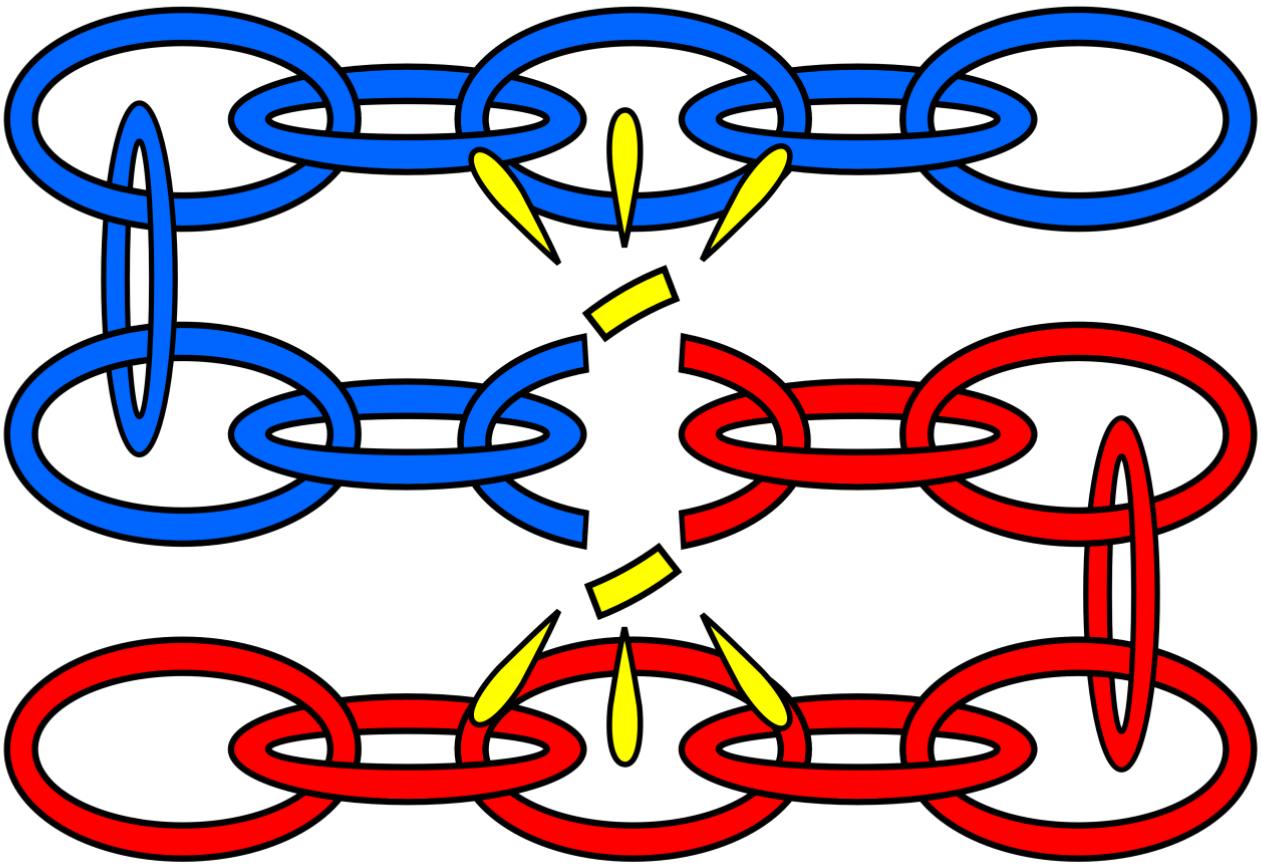
THE *Art* OF



TROUBLESHOOTING

Jason Maxham

Table Of Contents

© 2023 Jason Maxham

Title: The Art Of Troubleshooting
Author: Jason Maxham
First Edition: May 8, 2014
This Edition: October 15, 2023

Visit the companion web site at:
https://artoftroubleshooting.com/

Send comments and feedback to the author:
bookfeedback@artoftroubleshooting.com

Dedication

The Art Of Troubleshooting is dedicated to my Grandfathers. One was an artist, the other a troubleshooter:

- **John H. Maxham** (1915-1986)
- **Gerald W. Quade** (1923-2012)

Both owned and operated their own shops. My Grandpa Maxham was a commercial artist, making signs and other advertisements in Lebanon, New Jersey at Maxham Signs. My Grandpa Quade was an auto mechanic who ran a repair shop called Safety Service in Fairmont, Minnesota.

Each also served in World War II and was a member of the Greatest Generation, whose legacy I strive to uphold and extend.

Meet The Troubleshooters

While doing research for this book, I conducted interviews with ten exceptional troubleshooters. These people are problem-solvers that I deeply respect: their insights were invaluable in extending and refining my original thesis. Quotes from these interviews are scattered throughout the book; as you'll see, they said some very pithy things on the subject of fixing things. Note: quotes have been edited for clarity as necessary.

Many thanks to this group for enduring my endless questions about how they do what they do. I learned much, about machines and people, talking to them.

Their names and areas of troubleshooting expertise are:

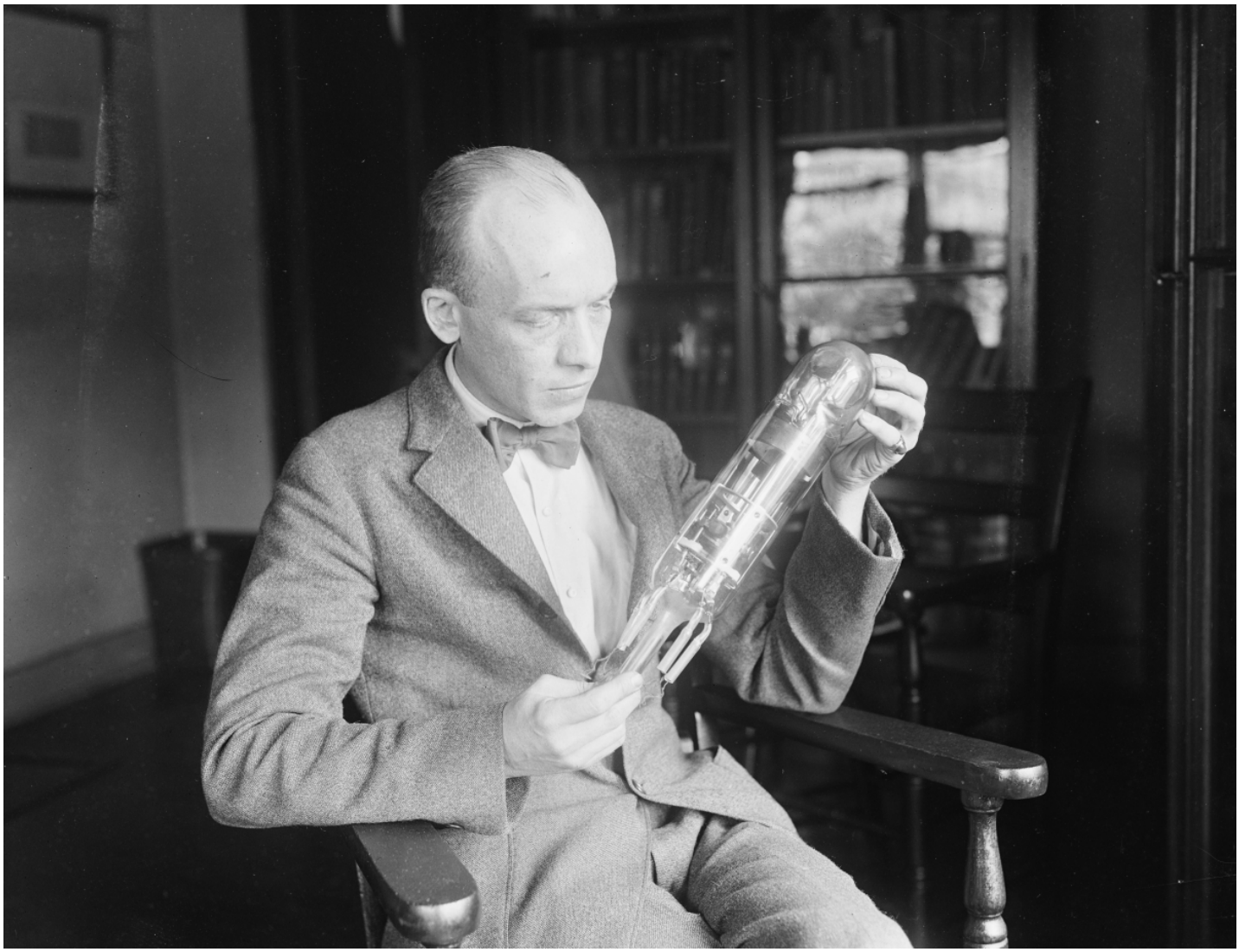
- **Alex Chaffee**, Programming
- **Ken Fechner MD**, Medicine
- **Jamie Karrick**, Mechanics
- **Rich Kral**, HVAC (heating, ventilation, and air conditioning)
- **Karl Kuehn**, Information Technology
- **Dan McCormick**, Mechanics
- **Mike McCormick PhD**, Information Technology and Scientific Equipment
- **Austin Quade**, Information Technology
- **Gerald Quade**, Mechanics
- **Jeremy Sheetz**, Mechanics

Part 1: Introductions

Let's throw this switch and get started!
(image: Jack Delano / Library of Congress)

> I always look out for the best interests of the client, not for myself. By doing that, I have unconditional trust. That's what I like, but it's a heavy burden. When people are putting all their trust in you, you want to make sure you're giving them the best of what you have to offer.
>
> **Rich Kral**

The Big Idea



> Everything fails.
>
> **Rich Kral**

Exciting or mundane, every day your life is filled with purpose. You wake up, attempt to advance your goals, and then fall asleep again at night. All around you are machines, amazing inventions that can amplify your intentions; it is probably difficult to imagine your life without them. We've developed a symbiotic relationship with the systems at the heart of our advanced industrial civilization. When they work, you are able to accomplish amazing feats, experiencing things and enjoying a standard of living that would dumbfound your ancient ancestors. When they don't…well, that's why you are reading this, right?

When a broken machine is standing between you and your goals, it's time to take action and get your hands dirty. Troubleshooting is about getting back to normal, about making something work again. Clever inventions enhance our lives in so many ways, but when they break down you can be left feeling powerless. I want to put you back in control by giving you the tools and mindset needed to have a healthy and productive relationship with the machines in your world. It doesn't matter if troubleshooting is your profession, as we all are dependent on machines and are negatively

impacted when they fail.

Why become better at troubleshooting? First off, it might be a matter of life and death. There are times when we literally place our lives in the hands of a machine: breathing from an artificial respirator in the hospital, or the smooth operation of the engines on an airplane that is cruising along at 37,000 feet. In these cases, the benefits of a quick resolution in the event of a breakdown are self-evident. Even when lives aren't on the line, speedily getting a machine working again allows you to get on with living your life. Lastly, there's nothing quite like that satisfying "aha!" moment when you get to the bottom of an issue and fix it yourself. Being effective in this way stirs the soul.

A toaster…
(image: Tom Hart / CC BY 2.0)

Maybe you'd like to be the hero at home by being able to fix things yourself. On the job, machines can either help or hinder your work objectives; understanding them from a troubleshooter's perspective can be the key to your livelihood. With the right attitude and knowledge, you can easily save the day at the office, factory, farm, or shop as well. Tights and a cape are optional. For professional troubleshooters, what I will describe are *the* fundamental skills to keeping your job and moving up. If you manage troubleshooters (and, if you manage humans, you do), then knowing the core strategies will allow your team to do more with less. Solving problems that make others cry will be especially good for your reputation and your wallet. Don't worry, I'm not going to ask for a cut of your inevitable pay raises. The fact that you learned it from me is payment enough.

The Crucible Inside The Cubicle

The origin of *The Art Of Troubleshooting* can be traced back to 2002, when I started a software company with three friends (it was called Discovery Mining, if you're interested). Looking back, I can say that the most interesting part of my job as CTO was troubleshooting really complicated problems. And, Discovery Mining had the perfect recipe for monster-sized issues: millions of lines of computer code, nearly 1,000 servers in multiple data centers, teams that were thousands of miles apart, and demanding clients spread across the globe that were always pushing the limits of our capabilities (sample quote: "I *only* tried to put 10 billion documents in a folder…do you think that would slow it down?"). As you can imagine, we had to deal with some very tricky problems.

Our company, a complicated web of interconnected systems, both human and machine-based, gave me a peek into the surprising complexity of our world. Surprising because, simple machines, when interconnected, can behave in unexpected ways. Oh, and that "simple machine" is probably more complex than you thought, once you start to carefully observe it.

The methods described here were largely discovered in the cleansing fire of a startup experiencing tremendous growth. Some are profound, some are so simple they border on the obvious, but they all are the result of butting up against the same problems over and over again. Eventually, I began to take notice of what worked and what didn't; from there I started to abstract some general principles of troubleshooting. While I feel our company's high-pressure environment was unique in driving me to think deeply about troubleshooting, I'm not claiming I'm the first human being to ever use these problem-solving methods. Many were taught to me by colleagues. Others were discovered accidentally when nothing else worked.

Naming Names

Doing research for the book, I interviewed some great troubleshooters, asking them explicitly about their methods. I've also asked plenty of questions about fixing things in casual conversation. Yes, I've been known to bring it up at a party—but only after I'm buzzed—pressing a slightly intoxicated stranger for the finer details of their latest computer or car breakdown can be highly amusing. Over time, I began to notice that many of the strategies I had discovered through my own trials were also known to others.

Discovering that the essence of what I knew was in use elsewhere, in a variety of occupations, was a big breakthrough. This confirmed my hunch that there was an opportunity to describe troubleshooting in general terms, *across* fields and industries. However, one thing I noticed was that even the great troubleshooters I met weren't very self-aware of their processes. That is, they were good at problem-solving, but had a hard time explaining why they were good. I've begun to speak using terms like "isolation," "narrowing," and "prerequisites for operation" (a category that includes troubleshooting's most famous question: "Is It Plugged In?"). These concepts have come up when speaking with other troubleshooters, but rarely as *conscious* strategies being explicitly employed. Therefore, I think that just by bringing the core concepts to the surface and assigning names to them, we can raise the quality of troubleshooting in the world. Because often giving a *name* to something is enough to bring it into your awareness and adopt its use. I believe that how we talk about things matter, so nothing would please me more than propagating a lexicon of terms that troubleshooters everywhere can use when discussing problem-solving.

The Goal

Unlike other works that may deal with troubleshooting a specific system or ones that have tips geared towards a particular field or industry, I'm interested in identifying the general principles needed to bring **any system** back to "working." That is, an actual set of practical strategies that will help you fix anything from a toaster to a nuclear reactor. Human beings have been depending on machines for hundreds of years now, so it should be no surprise that almost every scheme you can think of to fix them has been tried! Of course, not every strategy is equal and so you'll find the ones I've chosen to be tops for simplicity and effectiveness. These are the best of the best.

The other area in which I believe I can make a significant contribution to the troubleshooting arts is what I'll call "the human side." If there is a deeper lesson to be gleaned from my experience, it's that **all machine problems are human problems.** To some people, that will be disconcerting, especially if you think that the "world of machines" is different, an area separate from humans. At first, I resisted this insight because I saw machines as a refuge from the petty goings-on of the human race. Machines might be unintelligent, but at least they weren't malicious and inane! But woe to the troubleshooter that ignores the human side of the equation: your effectiveness will be severely limited. You can go so much further if you take into account this critical dimension of troubleshooting:

- By becoming an expert in untangling the language people use to describe their problems.
- By leveraging the psychology of problem-solving.
- By getting inside the head of an operator experiencing a problem.
- By knowing the economics behind what you produce and how that affects the resources available for repair.
- By looking at the *context* in which troubleshooting takes place: not just the surrounding and supporting systems, but the team and the larger organization of which it is a part.

Lastly, and perhaps most importantly:

- By understanding yourself, and how you best function as a troubleshooter.

…and a nuclear reactor. The underlying principles of fixing them are the same.
(image: Bjoern Schwarz / CC BY 2.0)

Implicit in communicating all this hard-won wisdom is the belief that anyone can learn how to be a good troubleshooter. I certainly wasn't born to fix machines, it was a skill I've acquired over time. How much quicker my development would have been if I had access to the ideas I've collected for you! All the principles I've written about, I've personally seen in action. I know that troubleshooting can be learned: everything here I have either successfully taught to someone else or learned from others. If I thought that troubleshooting was just something you were destined to do, I wouldn't have bothered writing this.

Further, I want troubleshooting to be recognized as a field in its own right. I hope some day that being acquainted with the principles here will be considered a basic life skill for anyone wanting to better themselves (like knowing how to drive a car or use a computer). The funny thing about troubleshooting is that everyone does it, but there is little formal training or even awareness of it as a discipline. I've never seen a class offered that was devoted to the topic in a general way. Consider this a first attempt at cataloging the entire field, a Linnaeus-inspired work to establish a

framework for others to extend. I dub thee *Homo Fixitus*.

It's beyond the scope of this work, but the principles here also contain lessons for problems in everyday life. If "all machine problems are human problems," you will see applications far beyond your workshop. I leave it up to you to figure that out on your own…

Defending The Obvious

I include a lot of "simple" strategies, like asking "Is it plugged in?" However, I also point out that this simple strategy is representative of a bigger concept called "prerequisites for operation." Even so, you may think, "All he's doing is pointing out the obvious!" If all I accomplish here is to give you a fresh appreciation of what's in front of you, I'll take that as a compliment. Troubleshooting is about the solution staring you in the face, if you'd just get out of your head and focus on the situation. Unfortunately, what's "obvious" is often ignored! If I can merely acquaint you with what's already known to work, you'd be well on your way to mastering the art of troubleshooting. On top of that, the original contribution I hope to make is to show you the rich and complicated implications of these "simple" and "obvious" strategies alongside a panoramic view of our advanced industrial civilization.

About The Terms I Use: Systems, Devices, and Machines

For the purpose of this work, the terms "system," "device," and "machine" are interchangeable. You may think of a mechanical contraption when you think of a machine, but here a "machine" can also be digital, or have no physical presence at all except as an abstract process (like a computer program). This broad definition encompasses a wide variety of things: assembly lines, computers, internal-combustion engines, network routers, airplanes, mobile phones, water heaters, nuclear reactors, software, etc. **Basically, anything that accomplishes work and can malfunction.**

Also, remember that "systems," "devices," and "machines" are themselves composed of smaller subsystems (often "machines" in their own right).

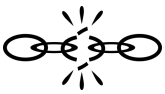
References:

- Header image: Harris & Ewing, photographer. (1924) *First radio vacuum tube. Carl W. Mitman, Curator of Engineering, US Nat'l Museum, holding what is believed to be the 1st radio tube.* United States, 1924. [June] [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016887335/.

The Big Idea was originally published September 13, 2011.

Notes:

One-size-doesn't-fit-all



> You are thinking: something used to work and we're trying to get back there.
>
> **Ken Fechner**

I have often stopped and wondered: "Could all the strategies in *The Art Of Troubleshooting* be condensed down to a single troubleshooting script?" Is there One Recipe to Rule Them All?

Many people I've talked to about troubleshooting have asked the same question. So, I went and figured it out. If you want the script, here it is:

The Universal Troubleshooting Recipe

1. Find the problem.
2. Fix it.

Okay…but it's not really that useful, is it? Sorry, that's the best I can do. I've tried to expand this beyond a 2-step process, but it gets too specific too quickly. Also, there are just too many "if this happens, then do this" clauses. For example, take "duplicating the problem": pursuing this strategy is often a necessary step to discovering the cause of an issue. Or, not: some problems can be duplicated with 100% reliability and you won't be that much closer to finding a solution (for example, a car that hasn't run in 50 years will be *very* reliably broken). Given enough time and resources, it's theoretically possible that all problems could be duplicated. However, it would be a waste of resources to single-mindedly pursue that strategy to the exclusion of others that could provide a quicker resolution.

Similarly, you might be able to "copy one that works," but then again you might not have an extra *one* on hand and so that would be a dead end. Rebooting or restoring the default settings may magically resuscitate a machine in just a few minutes—I've witnessed this hundreds of times. But, I've also seen this tactic have no effect. Other problems may require a rigorous data collection program executed over weeks or months to identify and fully understand the underlying issue. Also, sitting above any given strategy is the possibility of not fixing something, which has its own considerations. Presumably, the tests to *forgo* troubleshooting would also need to be included in our Universal Troubleshooting Recipe. Finally, economics will have the last word: there may be paths that are very effective, but can't be considered because of the cost.

I think you get the point: there's not a single troubleshooting strategy that consistently produces results *all* the time in *all* situations. Therefore, I think that a universal theory of troubleshooting can't be reduced to a single recipe on a card—it's more like a whole box of recipe cards. When it comes to food, depending on what you're craving and what ingredients are on hand, you try to choose the right recipe. If you have apples and want something sweet, apple pie is a good choice. If you're making dinner and you have a hunk of beef and some vegetables, then a beef stew is a tasty match. Choosing the right recipe for your ingredients and occasion is the key to good cooking: you can't make apple pie with beef, and a beef stew made from apples is going to taste like…something's missing.

Just one recipe for everything? That would be boring…
(image: liz west / CC BY 2.0)

Within a discipline or industry, there will be opportunities to develop a single troubleshooting recipe. If you only work on a certain make and model of car every day, you'd inevitably hone your routine for maximum efficiency, including only the most effective strategies and placing them in a set order. Those refinements may only be a narrow subset of all the ideas presented in *The Art Of Troubleshooting*—the rest being superfluous. I guess the analogy to cooking would be a chef that only worked with a single main ingredient: if you only had apples and made the same apple pie every day, there wouldn't be much use in learning how to filet a fish or debone a ham.

However, life isn't that predictable: the problems you'll need to troubleshoot will come in an infinite variety of forms and won't care about your set routines. In other words, be prepared to handle not only apples, but also pears, bananas, fish, potatoes, beans, rice, and anything else that can be eaten! In that way, choosing a troubleshooting strategy that is complimentary to your problem is just like cooking: the problem is the ingredients and the strategy is the recipe. Depending on the context, a given machine failure may need duplication, or a change of sequence, or a reboot, or something to be plugged in, or something completely different. Mix and match as needed and *voilà!*

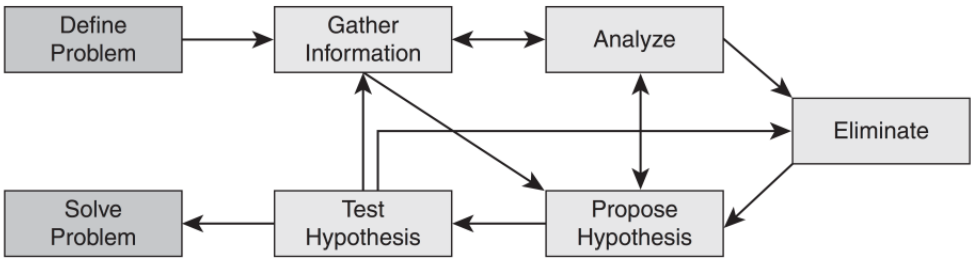
Until the particulars are known, you can't say for sure which strategy is going to be best suited for a given situation. This is why I believe troubleshooting is an *art*: deciding which strategy is most appropriate will be a judgement call, aided by your experience and intuition. The key is familiarity and flexibility: intimately knowing the core

troubleshooting strategies will allow you to select the one you think is going to have the biggest payoff. Add to that a flexible mindset which compels you to switch strategies if your first choice isn't working. That is, if your vision of a pie doesn't work out, maybe those apples could go in the stew after all. *Bon appétit!*

Fitting Into A Structured Approach

I believe that my methods, as formulated in the strategies, are the quickest way to a resolution for the vast majority of failures. However, there may be some situations that require a very rigorous approach to troubleshooting. I've worked on some extremely complex, intermittent failures that have benefited (or would have benefited!) from a formal process. For these rare cases, my strategy and "question-based" approach (as employed in my one-page Universal Troubleshooting Guide) may not be enough to satisfy your need for structure.

By now, I've seen a fair number of generic problem-solving processes. In addition to the reasons above, I want to address them because you might be curious, or your organization might promote their use. Here's one such example of a "structured troubleshooting approach" from Amir Ranjbar's book *Troubleshooting and Maintaining Cisco IP Networks*:



Excerpt: "Flow Chart of a Structured Troubleshooting Approach" from Troubleshooting and Maintaining Cisco IP Networks by Amir Ranjbar. [1]

Ranjbar's method has the following elements in this order:

> **Step 1.** Defining the problem
> **Step 2.** Gathering facts
> **Step 3.** Analyzing information
> **Step 4.** Eliminating possibilities
> **Step 5.** Proposing a hypothesis
> **Step 6.** Testing the hypothesis
> **Step 7.** Solving the problem
>
> ***Troubleshooting and Maintaining Cisco IP Networks*** [1]

In the abstract, this is a great way to think about troubleshooting. For very tough problems you might need to be this rigorous, documenting your progress through these steps as you inch towards a solution. However, just like my 2-step process above, it's a bit too general to be helpful as a quick troubleshooting recipe for an actual problem. Which facts do I gather? What possibilities do I eliminate? How do I choose a hypothesis to test? It's going to take some thought to translate these steps into specific actions.

You may be wondering how my methods relate to a generic problem-solving process like the one above. The answer is that the strategies I present are a kind of shorthand: they include all of the above steps, often in combination, in an implicit way. For instance, let's examine the "What's changed?" strategy within the framework of Ranjbar's process. The brief summary of this strategy is to find recent changes to a machine (or its environment), with the idea that one is causing a failure. Discovering the changes incorporates steps 2-3 ("gathering facts" and "analyzing information"). Choosing one, rolling it back, then observing the result combines steps 4-6 ("eliminating possibilities," "proposing a hypothesis," and "testing the hypothesis"). The power of a strategy like "What's changed?" is that it takes the generic approach and fills in the blanks, swiftly nudging you in a direction that has been very profitable for others.

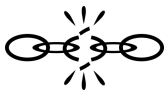
References:

- Header image: Harris & Ewing, photographer. *ARMY, U.S. NEW FIELD SERVICE SHOE.* [Between 1911 and 1917] [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016853755/.
- [1] Amir Ranjbar, *Troubleshooting and Maintaining Cisco IP Networks* (Indianapolis: Cisco Press, 2010), pgs. 32, 41.

One-size-doesn't-fit-all was originally published March 30, 2013.

Notes:

The Economics Of Troubleshooting



> What's the simplest thing I can do that's going to have the greatest effect in the least amount of time?
>
> **Dan McCormick**

When fixing things, I often find myself thinking about economics. All work involves economic considerations, but I feel there are unique aspects of repairing machines that deeply touch the core principles of the "dismal science." Economics are inescapable for the troubleshooter and will set the boundaries for the "if," "how," and "when" of fixing something.

We employ machines to fulfill our worldly wants, which are without end. Satisfying our infinite desires with our limited means has been called the "fundamental economic problem":

> Yet despite the comparative abundance of products and services emanating from the process of social cooperation, the economic problem remains: Wants continue to exceed the means or resources for their attainment. The persistence of the problem of scarcity means that even in a modern, highly developed, and productive society decisions have to be made regarding how the various scarce resources should be directed to the satisfaction of the more urgently felt wants of society's members.
>
> **Thomas C. Taylor** [1]

Look at the contents of any junkyard or landfill and you'll realize troubleshooting is a stellar example of this

"fundamental economic problem." Broken machines vastly outnumber the resources to fix them! Given the disparity between the quantity of breakdowns and the means to mend them, the end result is: what gets fixed is subject to a harsh but necessary triage based on people's most pressing needs. Only the most important systems will be worthy of being fixed.

When a machine breaks down, the entirety of the economic calculation that gave rise to its origin will be thrown into stark relief. Questions arise: What was its purpose? What need was it fulfilling? Is the need still present? If so, what resources will be diverted to fix the machine? Are there other, more pressing needs that I (or my organization) must satisfy first? What combination of labor and capital should be used to fix it? Will you use your own human resources to make the fix or outsource the labor to a repair shop? Will you choose a cheap, quick fix or go with a more expensive, longer-lasting solution? Or, should the machine be replaced instead? And, will it all be paid for with cash or credit?

The issues behind these questions are always present for the business owner, but they are easy to ignore when a machine is happily humming away. Once a system is installed, people tend to forget about the underlying economic motives behind its acquisition—that is until a malfunction occurs. Troubleshooting is deeply linked with economics because choosing a course of action demands an answer to the above questions. You can see that, if the original want is to continue being satisfied, an *economic* decision will need to be made in conjunction with the *technical* matters of fault finding and correction. In fact, the two influence each other: what is discovered by the troubleshooter informs the economics (i.e., "this is what is wrong and this is how much it'll cost to fix") and the economics dictate what is possible for the troubleshooter (i.e., "you have these resources with which to discover the problem and make a repair").

Scarce Resources

A wide array of means are available to the troubleshooter: tools, colleagues, consultants, spare parts, manuals, etc. Each of these may be optional, but time is needed for all repairs:

> A man's time is always scarce. He is not immortal; his time on earth is limited. Each day of his life has only 24 hours in which he can attain his ends. Furthermore, all actions must take place through time. Therefore time is a means that man must use to arrive at his ends. It is a means that is omnipresent in all human action.
>
> **Murray Rothbard** [2]

The smart troubleshooter understands the spectrum of possible fixes and what resources each requires to be properly executed. The resolution of these two opposing forces, the desire for the best possible fix and the limited means to pay for it, is achieved through negotiation and compromise. This dance is most prevalent in outsourced repair work, like an auto repair shop. The person bringing in their malfunctioning machine wants to get it working again by spending the fewest possible resources. Professional troubleshooters have their own incentives: to make a living and to pursue only those repairs that will result in long-term customer satisfaction (and therefore repeat business). This tension is healthy and ensures that both sides are left better off from whatever transaction is finally negotiated. For the machine owner, the value of the repair must exceed its cost. The troubleshooter must be adequately compensated and feel like a project is worth their effort (i.e., there isn't something better they could be spending their time on). Sometimes these opposing forces will not be able to find mutual satisfaction, and this means that a whole spectrum of repairs will *never happen*.

That reminds me of a situation I found myself in as a teenager back in high school. It began when I ran over a concrete-filled tire rim with my beloved first car. No one (especially my parents) believed me that the rim was knocked loose by the wind and that it rolled right in front of my car. But, that's what happened. Anyway, the "spatial conflict" with this "flying saucer" (my Dad's chosen term) left a giant hole in my exhaust system. There wasn't much troubleshooting involved in the diagnosis—the source of the deafening noise that announced my arrival from miles away was obvious. What was hard was getting the damage repaired on my tiny budget of "almost nothing" (I think it was about $50). My goals were modest: to shut up that insanely loud car long enough to sell it to another reckless teenager like myself. Getting it fixed almost didn't happen at all. I took the car to every repair shop in the area, explained the situation and my modest means, and heard many a derogatory chuckle in reply: "You want me to fix that for $50?! Ha ha ha!" I finally found a student mechanic who agreed to make the noise go away for the meager sum I could offer. If I had only $40 to spend instead of $50, or if the damage had been greater, my car would have ended up in the junkyard and been just another example of where resources fall short of what's required to make a repair.

For some troubleshooters, this negotiation of wants and means will be partially hidden. If you maintain machines internally for your organization on a salary, then someone decided, before you were hired, that the value of maintaining those systems was worth more than the the salary they offered you for the position. If the value was less, the hire would not have occurred at all! But even in this situation, the economics of troubleshooting are not far away and it's in your best interest to be nimble in your ability to offer a range of solutions. Troubleshooters can also act like entrepreneurs, envisioning *new* solutions to maintenance problems that save money for their organization or its clients.

Like Snoop Dogg said, this should be on your mind—especially when you're fixing something!
(image: Elembis / Wikimedia Commons)

Opportunity Costs

When working on a troubleshooting project, I like to keep the idea of "opportunity costs" in the back of my mind. In the preface to his book *Cost and Choice*, the Nobel Prize-winning economist James M. Buchanan illustrates the concept in a hilarious way:

> You face a choice. You must now decide whether to read this Preface, to read something else, to think silent thoughts, or perhaps to write a bit for yourself. The value that you place on the most attractive of these several alternatives is the cost that you must pay if you choose to read this Preface now. This value is and must remain wholly speculative; it represents what you now think the other opportunity might offer. Once you have chosen to read this Preface, any chance of realizing the alternative and, hence, measuring its value, has vanished forever. Only at the moment or instant of choice is cost able to modify behavior.
>
> **James M. Buchanan** [3]

I guess the same goes for the time you spend reading my writing. I hope you didn't forgo a hot date! The economics of troubleshooting may be an interesting topic, but it's not *that* interesting.

Anyway, if it wasn't clear from Buchanan's antics, opportunity costs are the result of making choices about how we spend our resources. If you decide to read an illuminating tract like this, you can't also use the time to go to the movies (by the way, I applaud your decision). Money earmarked for a new TV can't also be spent on a new refrigerator. When you divert your means towards a specific end, an infinite number of other possibilities go unfulfilled. The benefits that

could have been, from the pursuits that weren't chosen, are called "opportunity costs."

If you had unlimited time to muck around with a broken machine, then the choices you'd make about how to repair it would mean little. This strategy or that strategy, who would care? You'd eventually figure it out, but there'd be no emotional weight to either a brilliant solution or a meandering slog. However, our time to make repairs *is* limited, likewise are our other resources. You can't be turning a wrench, reading a manual, calling technical support, and shopping for a replacement all at once. You must make a choice about what direction a repair is going to take and forgo the rest. Speaking of choice, every troubleshooting situation involves a staggering amount of decisions: repair or replace, pursue a long-term fix or a short-term hack, outsource or do the work yourself, purchase used parts or new, and on and on. Even routine maintenance involves opportunity costs: the time and money you allocate in order to prevent future problems cannot be spent elsewhere now.

The takeaway is: in order to make sure your resources are continually deployed in the most efficient manner possible, periodically ask yourself "What else could I be doing with my time and money?"

References:

- Header image: Lee, R., photographer. (1939) *Auto parts store. Corpus Christi, Texas.* United States, Nueces County, Corpus Christi. Texas, 1939. Feb. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017782247/.
- [1] Thomas C. Taylor, *An Introduction to Austrian Economics*, Chapter 2. Social Cooperation and Resource Allocation (pg. 14).
- [2] Murray Rothbard, *Man, Economy, and State*, Chapter 1. The Concept of Action (pg. 5).
- [3] James M. Buchanan, *Cost and Choice: An Inquiry in Economic Theory*, Preface.

The Economics Of Troubleshooting was originally published May 28, 2013.

Notes:

The Right Tool For The Job

> I've used all kinds of things for tools... I remember being stuck on a bicycle race course and 3 of the 5 bolts that held my chain ring on fell out. I didn't have an Allen wrench, but I found a twig in the forest that was the right size. You can make your own tools to solve a problem, but it all comes back to the fact that your brain is the best tool.
>
> **Mike McCormick**

Stanley Kubrick's *2001: A Space Odyssey* contains cinematic images that are so powerful, you find them flashing before your eyes years later. That happened to me as I sat down to write this: I had a vision of my favorite sequence from the film, called "The Dawn of Man." In it, we see a monkey sitting amongst a pile of bones. The ape picks up a large femur and begins to swing it around, tapping it here and there on the skeletons that lay on the ground around him. His initial strokes are playful and tentative, but they quickly gain momentum. Soon, ribs are flying and skulls are being crushed in a frenzied rage. An "Aha!" moment occurs for the monkey: "This isn't just a useless piece of bone! I can use it to strike harder and faster than my hands will allow." With this invention, new food sources are now possibilities. Kubrick drives home the point for the audience by cutting to a large beast (a Tapir) hitting the ground. The horns in Richard Strauss' *Thus Spoke Zarathustra* rise to underscore the importance of what would normally be just an animal playing around with some bones: this ape has discovered the first tool! Later on, we see the sinister side of this discovery, that tools can be used for good or evil. That same bone has been turned into a weapon (a kind of tool) and used against a rival pack of apes to deliver a vicious beating.

2001 is fiction, and it could have happened differently… But, imagine being there at that moment, when primitive Man first picked up a rock or a bone or a branch and realized it was more than just a rock, bone, or branch. Every time you wield a screwdriver or a hammer, you are part of that amazing human legacy of technological innovation that stretches all the way back to the discovery of that first tool.

Today, the number of tools available to help you accomplish your goals number in the millions. General ones, like the venerable hammer or Swiss Army Knife, can be used in a wide variety of situations. Others, like a strawberry stem remover, can be amazingly specific and narrow in their application. Also, tools aren't just things you can hold in your hand: there exists a whole slew of things to help you find and fix problems in the digital world too (debuggers, log analyzers, etc.).

But before we talk about having the "right tool for the job," maybe it's a good idea to look at the pain of choosing:

The Wrong Tool

I've had moments in my life that made me feel even *more* primitive than that monkey in "The Dawn of Man": they usually involve having the *wrong* tool for the job. One such frustrating example occurred when I first tried to wire a computer network for one of the fledgling companies I co-founded. I was stringing network cables around our office: the goal was to connect wall outlets with a patch panel on the other end.

Terminating these wires without the right tools will drive you insane.
(image: ChrisDag / CC BY 2.0)

First off, using a razor blade, just stripping the insulation from the wires was its own challenge. I'd sometimes nick the conductor and have to start over by cutting the whole thing off (yes, I know now that there's also a tool for removing wire insulation). But, that was nothing compared to getting the wires into the posts on the patch panel or wall outlets. Using just a screwdriver, it was so hard to keep the wire in line with the top of the post. I would have it lined up, and then it would slip off. Getting the screwdriver to penetrate in just the right place was also challenging: it frequently ended up embedded in my thumb! Cue: swearing that would make a sailor blush. I tried everything I could with the tools I had on hand. In my mind, I built up a picture of network cable installers as supernatural beings: master craftsmen who could make those damned little wires rest neatly within their posts. After an hour, I was only able to do **one** before throwing up my hands in frustration and retiring for the night. Even so, the one cable I was able to terminate was a mess, with the posts bent and deep scratches from my screwdriver everywhere.

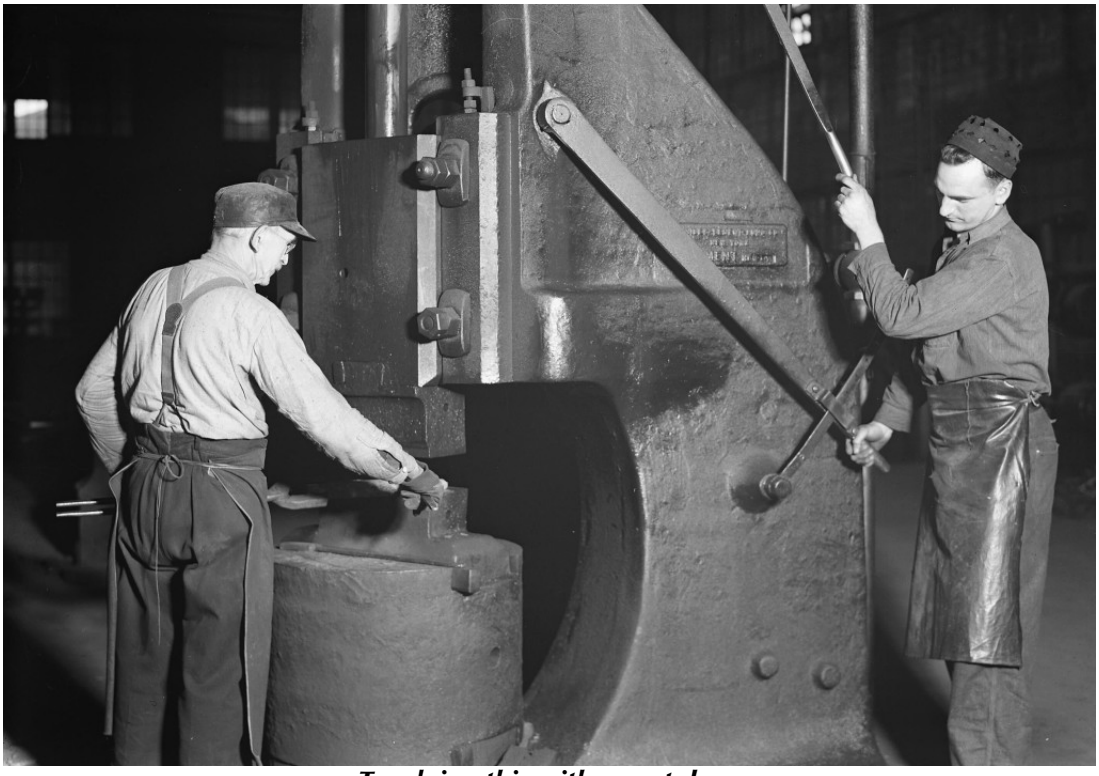
The next day, I went back to the store where I bought the cabling materials and had a chat with the owner. I told him my tale of woe, how terminating network wires was clearly the realm of divine beings. He walked me back to the aisle with the network cables and thrust a "punch-down tool" in my hand. I brought it back home and punched down my

first connection with my new tool. **The difference blew my mind.** What was incredibly frustrating before was now the simplest thing in the world: even a child could terminate network cables. The punch-down tool even made a satisfying "ka-ching" sound as it did its work!

The Right Tool…

Tools are born from the frustrations of pioneers who try to do something for the first time. Then they're honed by those who come after and try to make a living doing what those first-movers have set in motion. Like plants and animals, tools evolve over time, incorporating revelations of better ways to accomplish a task. When you buy a tool, you're buying more than just an object: you're buying the sum of what humanity knows about how to best do a job. Be grateful that you didn't have to endure the pain that lead to the creation of a tool, you get to tap into this fount of hand-held knowledge with just a swipe of your credit card. As you can see from my experience with terminating network cables, the right tool can make a job go from "impossible" to "easy."

If you're doing something unique and off-the-shelf tools aren't available, consider making your own. But, I don't have to suggest this, necessity will force you! You'll know when to become a toolmaker: it's when you need to repeatedly do a time-consuming or frustrating task. Consider improvising with tools from other trades or industries. I've often taken things that weren't necessarily made for my particular situation and adapted them for my purposes with great effect. Of course, my accomplishments pale in comparison to the undisputed King of Tool Improvisation: MacGyver. I still remember the episode of *MacGyver* ("The Prodigal") where he made a harpoon launcher from a telescope. That's world-class tool improvisation!



Try doing this with a spatula…
(image: Lewis Hine / The U.S. National Archives)

You'll recall that there are two steps you need to take in any troubleshooting exercise: 1) finding the problem and 2) executing the fix. You might associate tools with just step #2, but they are critical for the entire process. Diagnosing a problem may require *different* tools than the repair phase, but they are tools nonetheless. Tools for problem discovery usually include information gathering devices like meters, gauges, and probes. There will be some tools that are required for both phases: that's because finding the cause of a problem often requires disassembly to allow the observation of internal components. When it comes time for reassembly, chances are you'll need the same tools.

…At The Right Time

Possessing the right tool is only half the battle. The competent use of a tool also requires:

- Knowing how the tool works and being proficient in its use.
- Knowing *when* to use the tool.
- Having the tool available when it's needed.

Any one of these pitfalls could be a reason why the best tool for the job is not used. How many times have you been on the roof and discovered the tool you need is inconveniently located 3 floors below in the basement? Or, it was sitting in your toolbox just a few feet away, but you forgot that it existed? Good "toolsmanship" requires the functioning of the most important tool of all: your mind. After all, a tool is just an inanimate object–**you** are the bridge that connects the reality of a situation with the most effective means available.

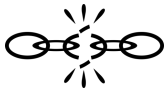
References:

- Header image: Highsmith, C. M., photographer. (2015) *Implements inside an old blacksmith's shop at the West Virginia State Farm Museum…* United States, Point Pleasant, West Virginia, 2015-05-09. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2015631952/.

The Right Tool For The Job was originally published June 11, 2013.

Notes:

There's A Fine Line Between Engineering, Invention, And Troubleshooting



> Essentially, all models are wrong, but some are useful.
>
> **George Box**

After writing about troubleshooting for a while, one of the unexpected questions that arose was: "Where does my subject end?" This prolonged mystery led to temptations of the "grass is greener" variety. When you write about something long enough, boredom will have you venturing off into related fields for some stimulation. Not-so-related fields too: when I grew tired of my chosen topic of repair, I procrastinated by starting two other books that had nothing to do with troubleshooting.

Now, there's nothing wrong with expanding the horizon of your thesis by introducing related topics. Those bordering territories are interesting places to take readers and give them context for your main argument. However, if you indulge your tangential whims too much, you will eventually get a work about life, the universe, and everything. I thought I showed restraint by sticking close to my chosen topic, but not everyone agreed. One reader's review said: "It [*The Art Of Troubleshooting*] sets out to be a sort of generic manual for everything. I see where Mr. Maxham is coming from in this grand idea, but it doesn't work and I think it never could." Of course, I disagree.

When I finally understood the boundaries of my topic, it was a relief because I wrote for much longer than I originally intended. What happened? I've found that writing is an unpredictable process of discovery, one that can take you far away from your intended destination. The act of organizing my thoughts, explaining them clearly, and researching supporting material often led me to unexpected places. The whole process was *expansive* because it exposed the gaps in my reasoning, as well as the missing context required for a reader to understand my arguments; likewise, new implications of my thoughts would unexpectedly appear and demand an explanation.

When it comes to the topic of troubleshooting, two neighboring fields that I like to visit are engineering and invention. But, where are the precise border lines between them? When I've mentioned my work on troubleshooting, people will often launch into their favorite stories of inspired tinkering or problem-solving. The fact that many of these good yarns don't involve repair means to me that a fog obscures the distinction between the three disciplines. I think this is partly related to the fact that the people who are good at one are often good at the others. But, it also speaks to the fluidity of our own selves: to get along in life you need to be part fixer, implementer, and dreamer. You may forget which role led to which result, so it's easy to understand the confusion. Therefore, I want to address the differences directly and in-depth. Not for a love of semantics, but because I think the distinctions are interesting and useful.

Thomas Edison is one of history's great problem-solvers, mixing imagination with organized effort to make new ideas come to life. Making something work for the very first time and then fixing it when it breaks both rely upon complimentary skills.
(image: Library of Congress)

The Need, Above All

You may have heard someone say that "necessity is the mother of invention." Allow one of history's greatest inventors to elaborate:

> None of my inventions came by accident. I see a worthwhile need to be met and I make trial after trial until it comes. What it boils down to is one per cent inspiration and ninety-nine per cent perspiration.
>
> **Thomas Edison**

It may surprise you that this "worthwhile need" drives not only the inventor, but also the engineer and troubleshooter. They all serve necessity, but with differing *means* to the same end. With this as our starting point, we'll see how these three roles advance our goals in different but complimentary ways. Let's start with some basic definitions:

- **Engineer:** "a person who has scientific training and who designs and builds complicated products, machines, systems, or structures."
- **Inventor:** "one who creates or introduces something new."
- **Troubleshooter:** "a person who finds and fixes problems in machinery and technical equipment (such as computers)."

Here's my quick summary of the three disciplines that gets at the heart of their differences:

- **Engineers** implement existing Machine Models, often adapting them to novel conditions.
- **Inventors** create and improve Machine Models.
- **Troubleshooters** restore a specific instance of a Machine Model.

What is a Machine Model? Read on.

Lest we get too wrapped up in our abstractions, it's good to remember that "the map is not the territory." Likewise, the model is not the machine.
(image: A. Ruger / Library of Congress)

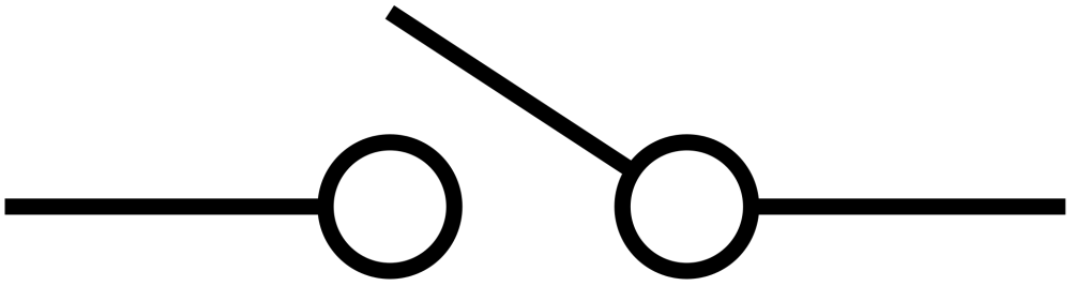
The Machine Model

The process of fixing a machine is made with constant reference to what I call the **Machine Model**, a *conceptual* understanding of a system that includes its component parts, how they are arranged, and the ways they

should interact. (Please note my expansive definition of the word “machine” that I introduced in the Big Idea: **“anything that accomplishes work and can malfunction.”**) A given Machine Model can be very broad, incorporating whole classes of systems (e.g., “cars,” “computers,” or “printers”). At this level of conceptualization, the parts and their arrangements may be similarly abstract. For cars, the model might simply be “4-wheeled transportation with a chassis, gasoline or electric powertrain, steering wheel, brakes, seating, and control pedals.” This sweeping Machine Model includes millions of cars, past and present, including the Ford Model T, Rolls Royce Phantom, Toyota Camry, Dodge Dart, etc.

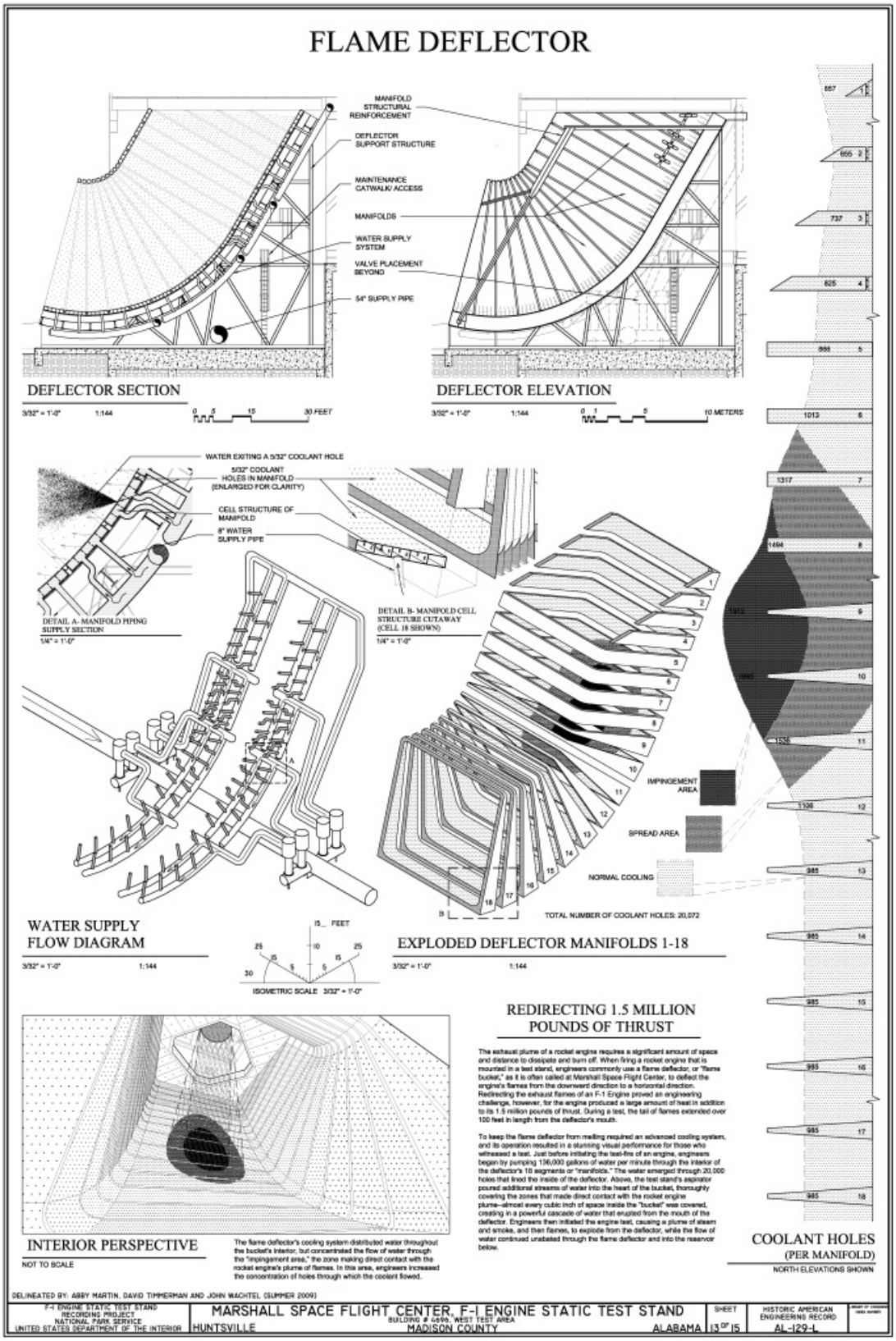
Machine Models can also be very narrow and extremely detailed, pertaining to only a specific manufacturer’s product (e.g., a Boeing 777, Toastmaster 1B14, or IBM RS/6000). In these cases, the model can include schematics, diagrams, manuals, statements of “description and operation,” and long lists of parts. The term can also refer to things used as components in the creation of larger systems, but which are still machines in their own right (e.g., switches, engines, gears, batteries, etc.).

There can be Machine Models which are elemental and include just a single function:



A simple machine with a complicated name: the single-pole, single-throw switch.
(image: Iainf / Wikimedia Commons)

There are also those that are extremely complicated, incorporating many smaller subsystems:



The "flame bucket": a simple name for a complicated machine.
(image: Abby Martin, David Timmerman, John Wachtel / Library of Congress)

As they are created by people, Machine Models can be sparse, erroneous, incomplete, or of little practical value. The Black Box is a good example of a model that isn't very helpful for troubleshooting. The term implies a closed system, where the only known parameters are inputs and outputs. Although the Black Box qualifies as a conceptual understanding of how a system works, it's not one that can easily be used for problem-solving.

Lastly, in case you think that you've never relied on a Machine Model, I will note that owners and operators are perhaps the most ubiquitous users of these forms. One consequence of a conceptual understanding of a machine is the

creation of *expectations* for how it will work: every time you put the key in the ignition of your car or press the power button on your computer, you are anticipating it functioning in a specific way. This goal-oriented perspective may be superficial compared to the original designer's understanding, but there's no doubt you are relying on a Machine Model as an end user.

Engineers, inventors, and troubleshooters all interact with the Machine Model; it is the common thread that binds these three disciplines together.

A mechanic straightens a fender: making a system function again is at the core of troubleshooting. That a machine once worked, representing an ideal to restore, is a crucial distinction separating fixing from invention and engineering.
(images: Esther Bubley / Library of Congress [left, right])

The 3 Fields Interact With The Machine Model

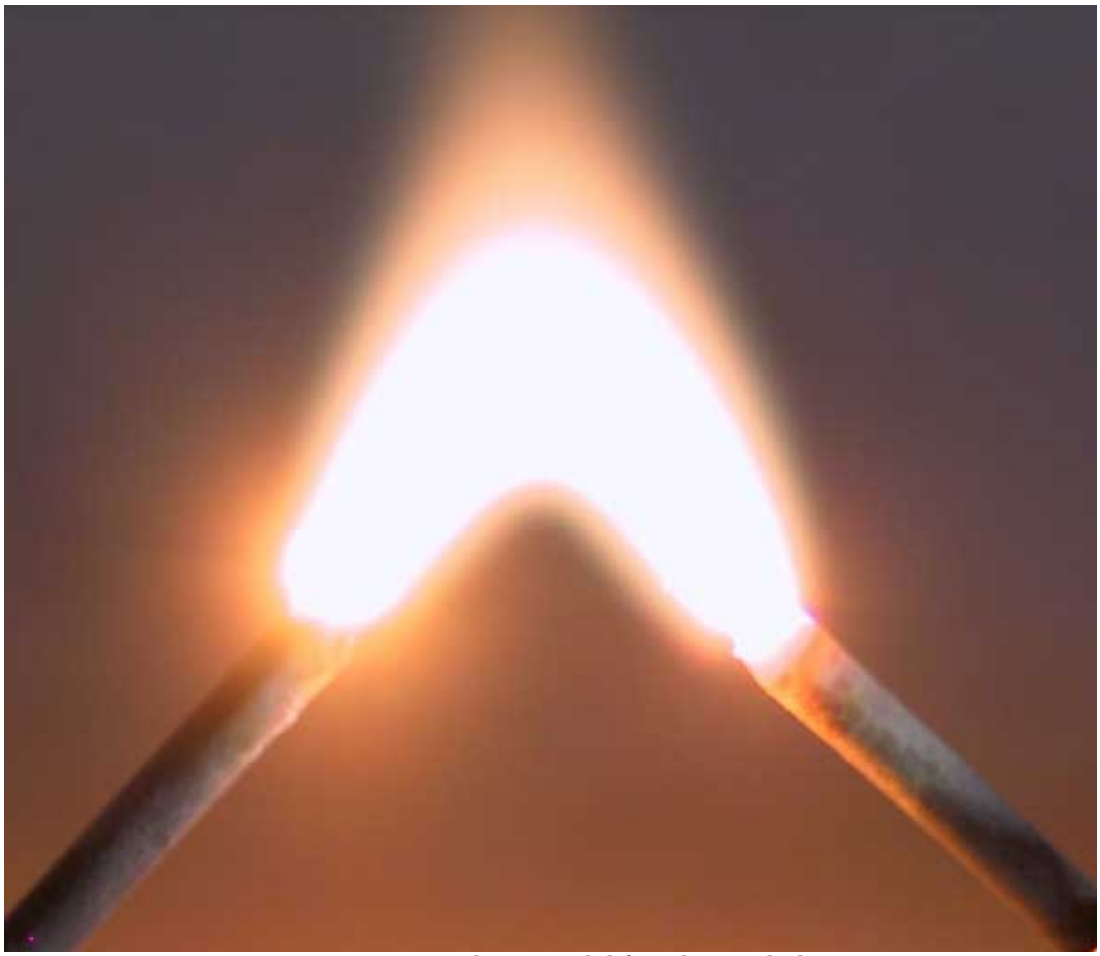
The Machine Model, whether it's broad or narrow, complete or partial, is the framework that guides the repair process. If a malfunction represents a deviation from the model's ideal, then repair is an attempt to restore that ideal. Fixing something would be impossible without these forms because they provide the necessary information for "how it should work." As they are abstractions, good models distill the important parts of a system into an efficient mental package. Their compact nature allows them to be easily shared, both for replicators (manufacturers) and fixers (troubleshooters).

Troubleshooters are guided by *existing* models to perform their restorative actions on broken systems. Engineers also employ these same Machine Models, but deploy them in new contexts, straddling the known and unknown with their work. The adaptation of a specific model for new projects is the organizing principle for many engineering firms. There are those that design airplanes, some may draft plans for fantastic new toilets, and others churn out schematics for computer chips. Again, these successful system frameworks are extant, but the challenge of engineering is making them work for fresh applications.

Now that we understand the concept of models and how engineers use them, we can further distinguish engineering from invention. While there's no doubt that engineering is a highly creative endeavor, taking the wondrous bounty of science and applying it to novel situations to satisfy our desires, the creation of radically new machine models is outside its purview.

Let's consider the design and construction of a water desalination plant. The specifics of its construction may be totally unique: the location of the building, salinity of the water supply, energy source, capacity, etc. After its completion, there may not be another facility in the whole world that is exactly the same. However, if the plant uses existing technology, perhaps one like reverse osmosis, you can't call the process that leads to its creation an act of invention.

What then is invention? Well, new Machine Models have to come from somewhere! Their origin is in the minds of inventors, who assuage necessity just like their brethren, but in wholly original ways.



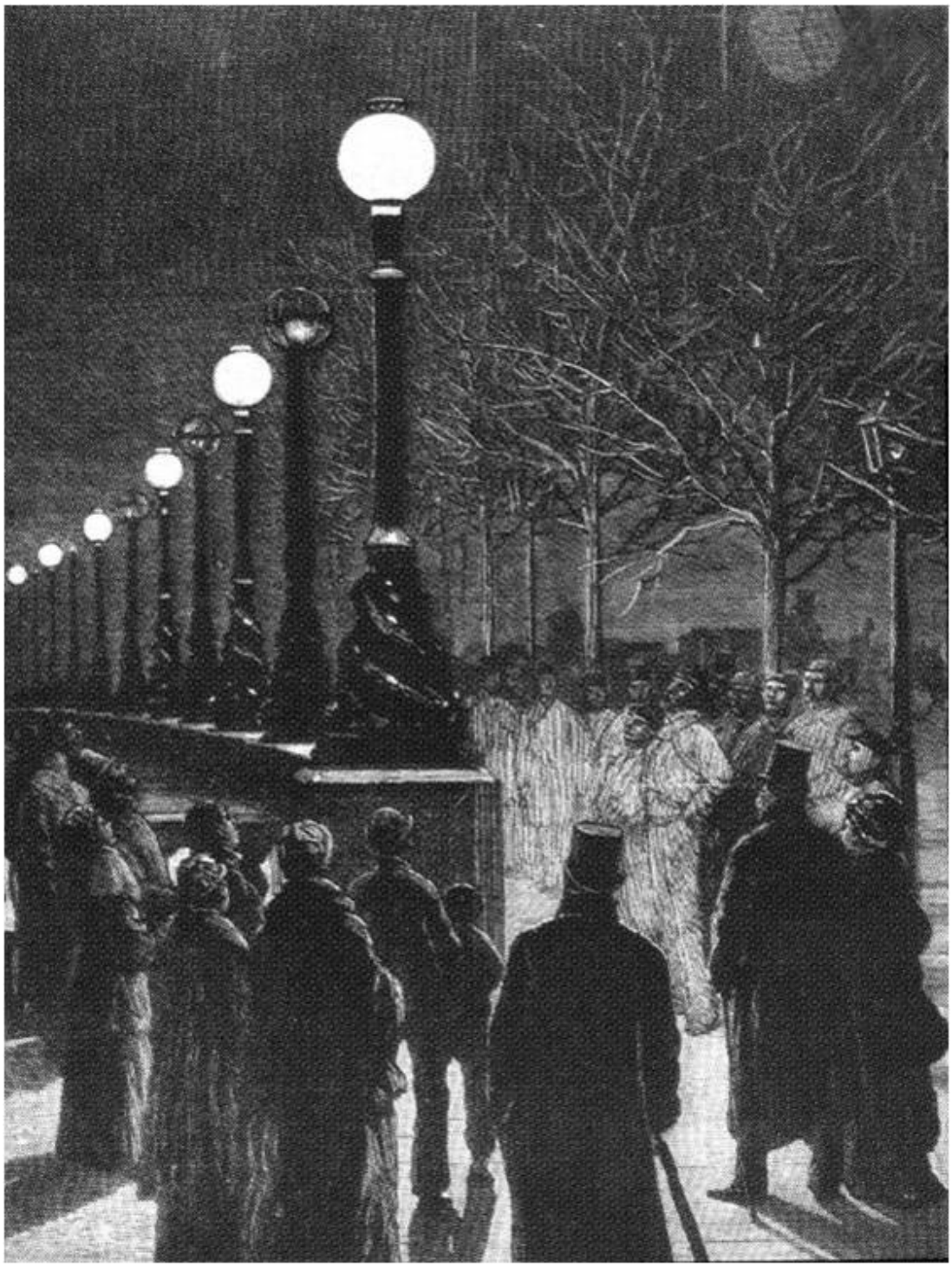
A primitive Machine Model for electric light.
(image: Achim Grochowski / CC BY 3.0)

A Light Turns On

Engineering, invention, and troubleshooting all use the Machine Model in different ways, making the relationship between these three fields rich and complicated. To show you why, let's examine one of the most useful creations ever conceived: electric light. You may think of Thomas Edison when you think of this technology, but the legend of the "Wizard of Menlo Park" may have obscured the fact that Edison didn't invent artificial light or even the lightbulb. While Edison deserves his lauds for making electric light commercially viable on a massive scale, the groundwork was laid well before his time.

To the best of our knowledge, the first person who deliberately created usable electric light was Francis Hauksbee—in 1705! Hauksbee's experiments with vacuums, mercury, and static electricity led to the creation of the gas-discharge lamp, which eventually became the warm neon lights that dot our cityscapes today. You could argue that the first person to create incandescent light was Alessandro Volta, who made a wire glow when he hooked it up to a battery. Pro tip: it's easy to be the first when you just happen to invent the battery too. That observation was a result of Volta's experiments with "voltaic piles" (also repeated by me in the 4th grade with a Duracell, which resulted in a burnt finger). While Volta is credited with inventing the electric battery, I haven't found anything that indicates he made the link between that hypnotic glowing wire and the wondrous possibilities for mass-produced electric light.

The two people who made that connection are the true progenitors of artificial illumination, but it's likely you aren't familiar with their names: Humphry Davy and Vasily Petrov. These two scientists weren't working together, yet around the year 1800 they simultaneously discovered that it was possible to produce substantial light with an electric current. Davy, a British chemist, hooked up a battery to carbon sticks and watched it glow. Petrov, a Russian physicist, did similar experiments, even noting the potential for electric lighting in a paper published in 1803 ("News of Galvanic-Voltaic Experiments"). The commercial result of their work was the carbon arc lamp, which was used in many of the early deployments of public lighting:

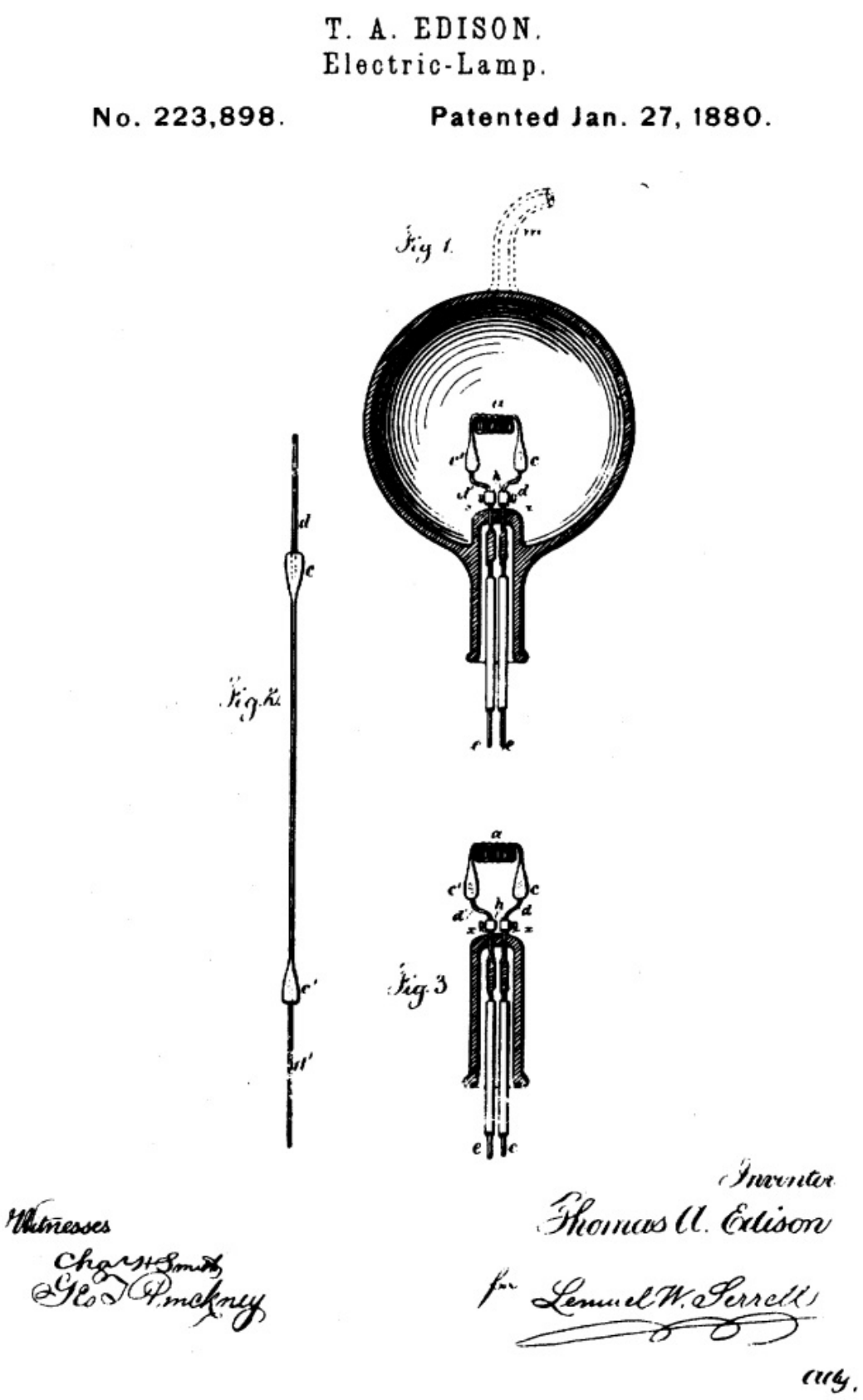


Let there be light! ***The Victoria Embankment in London gets electric street lights in 1878.***
(image: Wikimedia Commons)

By the 1870's, Davy and Petrov's simple model for generating light (passing current through a suitable medium) was still the basic framework around which researchers were innovating. Work centered on the main issues preventing electric light from being a consumer product for the masses. For their part, carbon arc lamps were too bright, used too much power, and were dangerous: they emitted harmful UV rays, along with hazardous sparks and heat. All these factors made them unsuitable for the smaller scale and confined nature of a home. The known alternatives for incandescent mediums were also problematic: they would be spent quickly, melt, or ultimately start on fire.

To prevent combustion of the filament, one solution was to enclose the apparatus in a glass globe devoid of oxygen (the "bulb" part of the lightbulb). However, by the time Edison started tinkering this was already "old" technology, first being patented back in 1841 by Frederick Mullins. Unfortunately, using an enclosure created further issues: it was difficult to achieve a good vacuum and even slight combustion of the filament would generate a black soot that would cloud the bulb, blocking the light. Creators of all stripes will be familiar with the sometimes frustrating "one step

forward, one step backward" nature of design decisions like this: often by solving one problem, another one will appear as an unintended consequence!



Edison and many other brilliant inventors painstakingly refined the model of the lightbulb. It continues to evolve today…

(image: US Patent 223898)

Edison focused his efforts on perfecting two critical parts of the prevailing lightbulb model: the filament and the vacuum in the glass globe. The story of how he happened upon bamboo is serendipitous and legendary: on a hot day Edison raises a hand fan to cool himself, notices it is made of bamboo, and decides to give it a try as a filament (after having rejected thousands of other candidates). Japanese bamboo turned out to be a high-resistance material that burned for over a thousand hours, a huge improvement over previous mediums. On top of that, Edison devises an ingenious way to improve the vacuum in the bulb by simultaneously heating the bulb while using a Sprengel pump to remove the air. It's this one-two punch of a better filament and improved vacuum that makes the light bulb commercially viable for the first time.

But, what do we call this achievement? The centrality of the Machine Model makes it easy to confuse fixing, creating, and implementing. Given that Edison's work on the lightbulb was within the boundaries of pre-existing forms, shouldn't we call it "engineering?" After all, he was taking the discoveries of scientists (Davy and Petrov) and utilized a well-established concept of electric light. On the other hand, trying to make a specific *thing* work might seem more in line with troubleshooting. Isn't swapping out various filaments like changing a tire or replacing a fuse?

Examining the Machine Model that prevailed at the time allows us to answer these questions. Before Edison's improvements, the existing concept of a lightbulb included a short list of sub-optimal filament materials (carbon, platinum, and iridium) enclosed in a partial vacuum. The arduous work at Menlo Park (truly in the "99% perspiration" category) to discover a new material and improve the vacuum stands outside the definition of engineering.

The unknowns dealt with by engineers are of a narrower type: the specific variables of the time, place, and purpose that will see the application of a given technology. The systematic research that finally identified bamboo as suitable medium for incandescence, along with the utilization of the Sprengel pump, took humanity far beyond the scope of its current knowledge. Edison *altered* the model of the lightbulb, creating something wholly original. Any time a Machine Model is modified or born anew, an undeniable act of *invention* has occurred.

A Fluid Relationship

Given the ties that bind them together, it should be no surprise to find:

- Troubleshooters who are adept at building new things.
- Engineers conjuring up wonderfully imaginative solutions to routine problems.
- Inventors that are skillful at fixing systems.

Why? Necessity is the mother of all these fields, and the Machine Model is the common and repeatable form that helps them meet those needs. Once this is understood, even subconsciously, it's easy to switch roles. My writing has focused on troubleshooting, but my career has included a healthy dose of engineering (and a little bit of invention). I was very good at fixing the systems I designed—because I was the model maker!

Fixers, creators, and implementers each view the Machine Model from a unique perspective that fosters an easy transition between the three disciplines:

- **Troubleshooters** have the most pragmatic approach because their restorative actions serve the underlying need so plainly. As they are often asked to work with a whole class of machines (e.g., an automobile mechanic will be familiar with most vehicles in common use), they typically understand a particular Machine Model in a wide variety of instances. From this base of practical experience, it's a short distance to building and deploying those same forms in new contexts (engineering). Troubleshooters also easily morph into inventors when the pain of repetitive breakdowns prompt improvements to the Machine Models for which they care.
- **Engineers** have scientific training for a deeper understanding of how a particular Machine Model works. However, because their designs are meant to be deployed in the service of actual customers, they are no strangers to hands-on work. Engineers acquire solid troubleshooting skills by testing their schemes before they are fully realized. This learning continues as customers identify and demand resolution to problems found long after a design is released into the world. Engineers' theoretical training, along with client pressures and competition from other firms, also prime them to imagine fantastic new additions to the Machine Models they utilize.
- **Inventors** may dream up entirely new designs, but their foundation is typically a deep understanding of existing Machine Models. Edison was the quintessential example of this approach, being grounded in the state-of-the-art while simultaneously letting his imagination soar. He built upon previous models, leveraged existing technologies, and his ideas about electric light didn't emerge from a vacuum (*ba-dum ching!*). Being conversant with existing

models allows inventors to easily cross over into engineering. Also, just like engineers, inventors must test their designs before they are ready for manufacturing. This vetting process turns them into capable troubleshooters.

Lastly, I want to point out that these roles extend well beyond the workshop. Joseph Campbell may not have written about them, but the troubleshooter, engineer, and inventor are vital human archetypes that are essential to living life. When it comes to surviving in this world, often we would just like to fix what is already in place. Other times, we need to build along the lines of what we know to be true. There's also a time to be guided by our dreams, boldly pushing into the unknown.

> If we did all the things we are capable of doing, we would literally astound ourselves.
>
> **Thomas Edison**

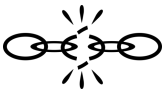
References:

- Header image: Highsmith, C. M., photographer. *Contraption on wheels in Alaska.* United States, Alaska. [Between 1980 and 2006] [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2011630866/.

There's A Fine Line Between Engineering, Invention, And Troubleshooting was originally published April 4, 2015.

Notes:

Beginnings, Middles, And Ends



> The only perfect program is an empty file.
>
> **Alex Chaffee**

Recently, I was sitting in the hot tub at a hotel in Laughlin, Nevada. The spa jets were on a timer and, after a relaxing soak, the timer expired. This returned the jacuzzi to placid stillness. The chilly morning air, coupled with the reset button that was seemingly far away, prompted the following conundrum: should I brave the cold to turn the jets back on? Or, maybe I liked it better without them running? Yep, deep thoughts… I sat there and hoped someone would happen by and turn them on for me, saving me from my first-world dilemma.

As I was pondering this, the air jets briefly spurted back to life! Had my prayers been answered? Alas, it was a false victory because after only a few seconds they fell silent again. I assume this last gasp was the system balancing the pressure that had built up while the hot tub was running. I also noted that this was something you'd only see at the *very end* of the hot tub's timer cycle.

This throwaway event got me thinking about the numerous beginnings, middles, and ends you will see as a Troubleshooter. Specifically, we're interested in examining these 3 stages during **normal operation** and over a machine's **lifespan**. Because different and unusual things happen in each of these phases, recognizing which stage a machine is in can be extremely useful.

Beginnings are important.
(image: Nationaal Archief)

Beginnings

The important institutions of humanity, whether religions, cultures, nations, or organizations, all have accounts of their origin which are studied with reverence:

- "In the beginning God created the heaven and the earth."
- "The Tao gave birth to One. The One gave birth to Two. The Two gave birth to Three. The Three gave birth to all of creation."
- "When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another…"
- "Bill and Dave…first met in the early 1930s while studying radio engineering at Stanford University in Palo Alto, California. Both avid outdoorsmen with a rabid fascination for electronics, the two became fast friends, spending many weekends camping and fishing in the wilds of the Colorado mountains."

People are naturally curious about the backstory of something they deem important, whether it's the tale of how their parents met, a famous person's upbringing, the inspiration for a bestseller, or the key ideas that preceded a major scientific breakthrough. For many comic book fans, the story behind their favorite superhero's powers is the most intriguing part of the character.

I think origins fascinate us because we are prompted to play a mental game of connect-the-dots. Whenever I read a news report involving violence or an unfortunate accident, I want to know the preconditions: he was drunk and walking by himself late at night in a bad neighborhood, she hired the cheapest skydiving instructor that could possibly be found, etc. We like to believe that by knowing how something bad started, we could prevent it from happening to us. Likewise for positive outcomes: she went to Harvard before making partner at her law firm, he was almost hit by a bus and then bought a winning lottery ticket, etc. In life we can sometimes read too much (or too little) into beginnings, but they are vitally important to those who fix machines.

When a machine is assembled and run for the first time, some special things need to happen. The many prerequisites

for operation need to be fulfilled, including adding consumables like gasoline, oil, ink, or electricity. Then, the machine needs to be configured to do useful work. Lastly, a break-in period may be required, running above or below a target level of usage, to ensure the longevity of certain components. Many mass-produced items are expected to simply work "out of the box," so these initial steps are frequently done as part of the manufacturing process. If a machine is re-made by your effort, you must manually take these initial baby steps that would normally be done at the factory.

Just like it was thrilling to discover that Peter Parker's powers stemmed from a chance bite by a radioactive spider, you should learn the backstories of the machines under your care with a similar level of enthusiasm. Geek confession: I followed the histories of the systems in my startup's infrastructure with the zeal of an obsessed comic book fan. "This server has already been back twice to the manufacturer for crashing," "This keyboard was involved in an epic coffee spill," etc.

These kinds of origin stories may not be the subject of a big-budget summer blockbuster, but they are worthy of your attention if you are responsible for ensuring things work around your home or business. History is something to screen as you consider whether to let a machine into your Circle of Trust. CarFax is a good example of a company that is dedicated to the premise that backstories matter: for a used car buyer, accident history, odometer fraud, or title problems is drama best avoided.

In motion: a machine does useful work in the (hopefully long) middle part of its lifecycle.
(image: Jet Lowe / Library of Congress)

Middles

There's comparatively less to say about middles except that, like a caterpillar or centipede, I hope there's *a lot* of it for your machines. While beginnings get a machine ready, middles are where the work gets done. I find these are the most prevalent causes of problems in the Middle phase:

- **Lack of routine maintenance:** when a machine has a reduced lifespan, inadequate upkeep is often to blame. Routine maintenance keeps components within specification, and is essential for a machine to reach its expected longevity.
- **Resource exhaustion:** these are among the simplest troubleshooting problems to solve, as the remedy is to add whatever has been depleted: gas, oil, ink, batteries, etc. Operating a machine to do work consumes these means, which is why resource exhaustion is associated with the middle stage of operation.

Assuming any problems with setup and break-in were handled correctly, the middle part is where a machine just

cranks away, churning out widgets, highway miles, or bytes. I've come to really appreciate well-tuned systems in this stage. Perhaps you've encountered a machine like this, one that is broken in and running just right. With mechanical machines, this is an analog, physical presence. I've ridden motorcycles that have had this velvety quality: the brake lever and gear shifter are neither tight nor sloppy, the throttle moves smoothly, and the engine purrs.

Endings are a time to reflect and a prompt to move forward.
(image: Library of Congress)

So We Beat On, Boats Against The Current

> 'Oh, I don't know. I can't count days in Rivendell,' said Bilbo. 'But quite long, I should think. We can have many a good talk. What about helping me with my book, and making a start on the next? Have you thought of an ending?'
>
> 'Yes, several, and all are dark and unpleasant,' said Frodo.
>
> 'Oh, that won't do!' said Bilbo. 'Books ought to have good endings. How would this do: *and they all settled down and lived together happily ever after?*'
>
> 'It will do well, if it ever comes to that,' said Frodo.
>
> 'Ah!' said Sam. 'And where will they live? That's what I often wonder.'
>
> **J.R.R. Tolkien, The Fellowship of the Ring**

We might laugh at this, but Sam is right: a story can go on forever. Where to cut it off is a deliberate choice made by the author. Sometimes I feel like the denouement was perfectly executed (*The Great Gatsby*), sometimes I would have liked a resolution much sooner (*Lost*), and other times I desired the tale to go on and on (*Arrested Development*, which eventually was extended: be careful what you wish for).

The endings of machines are also wide-ranging: they can be clear-cut, ambiguous, planned, forced upon you, happy, bittersweet, sudden, or catastrophic. Sam's insight also applies because those concluding moments can be greatly

influenced by your actions: repair and preventative maintenance give you a significant degree of control over when and how a machine dies. Closure of a system's working life can also be temporary, as anyone who has prepared a boat or car for long-term storage knows.

For most people, picking the endpoint of a machine will be an economic decision (for more, see what I've written about the "repair or replace" dilemma). Also, an ending for you can be a beginning for someone else: sometimes the expiration of your relationship with a machine just means transferring it to someone else. Regardless of how a machine under your care meets its terminus, it's an invitation for reflection and a prompt to move forward. The world keeps on changing, and your needs along with it…

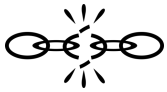
References:

- Header image: Ravenna, A., photographer. (1962) *Officials man the shovels with a vengeance at ground-breaking ceremony for Hunts Point produce market / World Telegram & Sun photo by Al Ravenna*. New York, 1962. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2006686002/.

Beginnings, Middles, And Ends was originally published May 28, 2015.

Notes:

You Won’t Guess The Hard Part



> Known problems come first, 'cause I've already done the work. I know what the problem looks like, I know how to test for it, I know how to fix it.
>
> **Karl Kuehn**

Recently, the lights in my bathroom stopped working. I would flip the switch and the bulbs would glimmer briefly, but they wouldn’t stay on. What more could a troubleshooter ask for? Here it was—a project!

I thought briefly of calling in the cavalry (i.e., a pro), but I’ve heard that it’s hard to get tradespeople to come out for “little stuff”. Smaller jobs like these are why many people rely on those who are “handy”: those jack-of-all-trades that can pass for an electrician, painter, plumber, or carpenter. Since I am my own favorite handy type of handyperson, I decided to dive right in.

First, I watched a few videos on how to replace a light switch. Cute cat videos aside, the learn-by-seeing possibilities of video sharing sites like YouTube make them one of the greatest self-education platforms ever created (also notable: libraries). Now, before I tell you any more, I want to share my mindset going into this project. Thinking about repair is something I like to do (obviously!). I may ruminate about fixing things more than the average person, but it’s only because my mind is saying: “Pay attention! You probably will end up having to write an article about this…”

However, as much as I’ve studied and theorized about fixing things, I’m constantly humbled by reality. In particular, guessing the **Hard Part**. You know what I’m talking about: whenever you contemplate a fix-it project, your mind starts spinning. You do your best to conceive of a plan from end-to-end, and your thoughts will invariably dwell on the tricky

parts of your imaginary process. Maybe it's something you don't understand, a task you perceive as strenuous, or something that's tripped you up in the past. Whatever the reason for my consternation, I'll say to myself, "This is *the* part of the repair that will cause me trouble."

Simply by slowing down, asking myself the right questions, and thinking about what I'm about to do, I've saved myself countless times. It's easy to rearrange the steps of a project in your mind, but once you start swinging a hammer, things get harder to reverse. Avoiding trouble, at the speed of thought, pays big dividends. Yet, dwelling on the imagined Hard Part seems to have consistently poor returns.

What I perceive will be difficult isn't usually just a little bit off, it's often not even in the same ballpark. In the case of this particular repair, I thought the most difficult parts of the project were going to be:

- Avoiding the dangers of live electricity.
- Diagnosing the problem: I'm not an electrician, so how am I going to figure out what is wrong? Is the problem the switch, the light fixture, the wiring, the bulbs, etc.?

Yes, these two concerns did have to be overcome, but it turns out they were among the *easiest* parts of the project. The DIY tutorials I watched all started with disabling the electricity to the circuit about to be worked on. Sure, it took a little trial and error, but it was just a few quick flicks on the electrical panel and I had identified which circuit was powering the lights. Done.

Okay, but I still had to complete the diagnosis. Surely that would be sticky, right? Yet again, this was easily overcome when I thought about the electronics class I took in high school. We learned that a switch is a very simple device: it just breaks or completes a circuit. To figure out if this particular switch was faulty, all I had to do was remove it from the circuit and see if the lights turned on. I'm no electrician, but this seemed like an obvious way to isolate the switch. It felt good to so easily figure this out: "challenge met, competitor bested, obstacle overcome!"

With the power off, I began to remove the switch, attempting a bypass to see if the circuit would complete in its absence. But here's where the actual Hard Part entered the scene: it turned out that I had a lot of difficulty removing the old switch. In the videos I watched, the wires were attached by screws on the *side* of the switch. Installed in this way, disconnecting the switch was just a matter of undoing the screws.

On my switch, however, the wires disappeared into the *back*. I pulled on them every which way. I cycled through my modest collection of pliers, from small to big, in an attempt to improve my grip. I considered cutting the wires, but ultimately didn't because I wanted to preserve the original wiring. Stuck here, I cursed the drywall that looked on and had witnessed the switch's original installation, quietly hiding the secret to this puzzle.

As I got more aggressive wrestling with the wires, I sensed that I was about to cross a line: after all, if you have to force it, you're probably doing it wrong. So, I decided to take a break and do more research. Why wouldn't the wires come out? If you know anything about common light switches, you probably know the answer to my dilemma. There are two ways to connect the wires: 1) posts with screws on the side and 2) gripping clips in the back. The clips are designed to operate one-way: you slide the wire in and the clips prevent the wire from coming out. To get the clips to relinquish their death grip, you have to insert a screwdriver into a little hole and hit the release mechanism. Once I recognized this, the wires detached easily from the switch.

From there, the rest of the project was downhill. I removed the switch, connected the wires that used to run through it, and turned the power back on. Shazam—the lights came on, bright and clear! This test isolated the switch and showed that it was likely the culprit. Therefore, the obvious next step was to replace it. That meant a trip to my new favorite place: the local big-box hardware store. Returning with a new switch, I attached the wires through the holes in the back, realizing why they have two ways to connect the switch. If you worked as an electrician wiring new construction, the gripping clips would save you a lot of time. They avoid the extra work of bending wires around the posts and screwing in the screws.

The new switch works great, and my flossing is back to being fully illuminated (lesson learned: don't floss in the dark). But, the trouble I had getting the wires out of the switch got me thinking about the Hard Part. Why is it so difficult for me to predict what's actually going to be difficult about a repair project?

Imagining Risk

Searching for an answer, I was reminded of an interesting story I read in *The Black Swan*. This book is a fascinating meditation on improbable events, with author Nicholas Taleb showing how we often fail to properly conceive of rare but significant hazards: from misused mathematical methods and faulty conceptual frameworks, to our own mental shortcomings that make it difficult to comprehend the very large and very rare.

While illustrating the mismatch between perceived and actual risks, Taleb recounts his visit to a prominent Las Vegas casino. On a tour of their facilities, he finds that the "casino's risk management, aside from setting its gambling policies, was geared toward reducing the losses resulting from cheaters." Along these lines, they show him an elaborate electronic surveillance system, which made Taleb feel like he was "transported into a James Bond movie—I wondered if the casino was an imitation of the movies or if it was the other way around."

However, when cataloguing and ranking the top incidents that *actually* came close to putting the casino out of business, he finds that—they had *nothing* to do with stolen cash, roving bands of card counters, or shady dealers pocketing chips! For example, one of the casino's star entertainers got bit by a tiger during a magic show (apparently, this alone was about a $100 million loss!). Then there was an injured contractor, so offended by a settlement offer, that he tried to blow up the casino (luckily, the attempt was thwarted). And, this doozy:

> …casinos must file a special form with the Internal Revenue Service documenting a gambler's profit if it exceeds a given amount. The employee who was supposed to mail the forms hid them, instead, for completely unexplainable reasons, in boxes under his desk. This went on for years without anyone noticing that something was wrong. The employee's refraining from sending the documents was truly impossible to predict. Tax violations (and negligence) being serious offences, the casino faced the near loss of a gambling license or the onerous financial costs of a suspension. Clearly they ended up paying a monstrous fine (an undisclosed amount), which was the luckiest way out of the problem.
>
> **Nicholas Taleb, The Black Swan** [1]

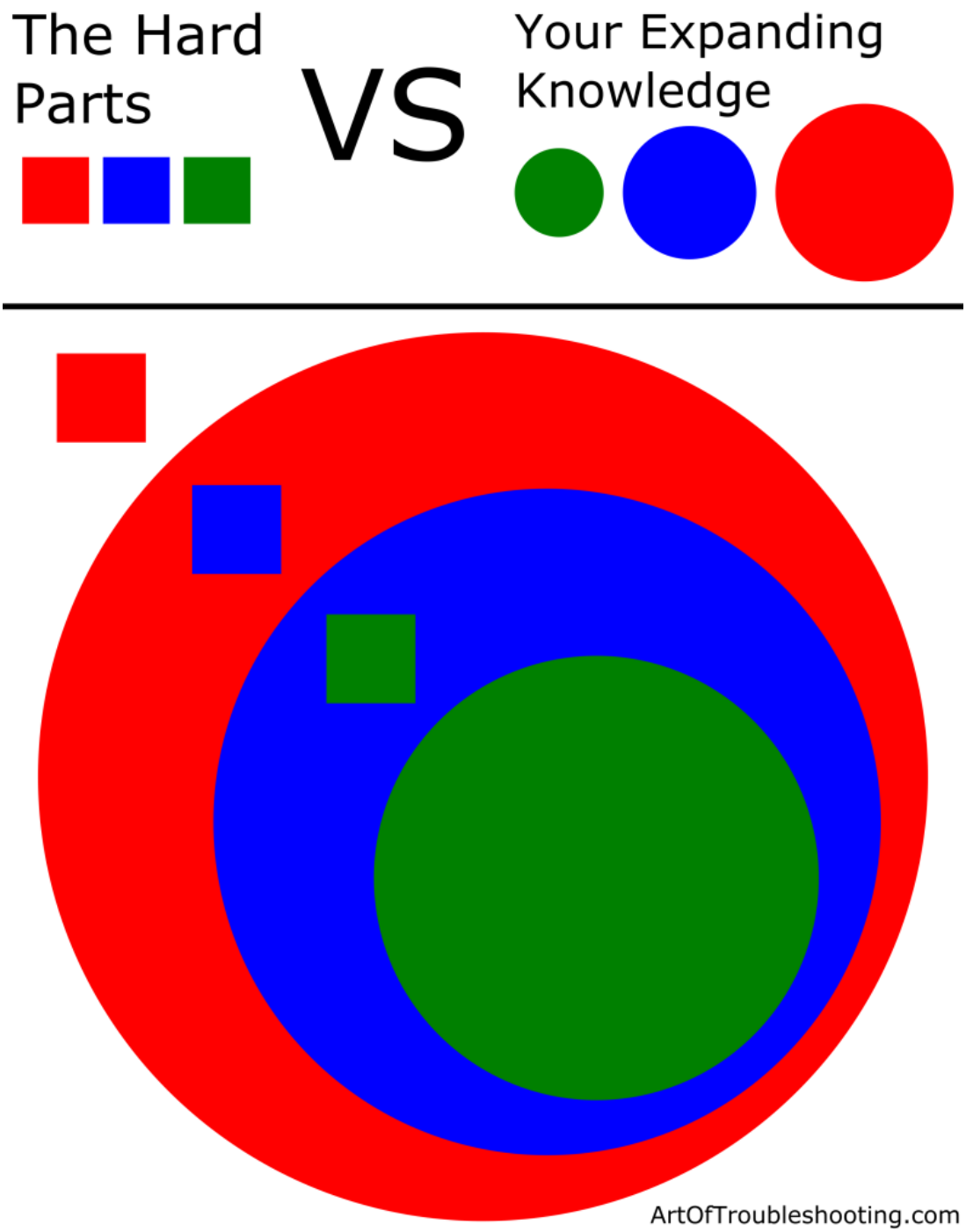
Average Fears

When it comes to guessing the Hard Part while troubleshooting, I'm like that casino risk management team. My fears are loosely based on the small sliver of reality that I know and think I can control, using forms that are familiar. In this sense, they're often *very* pedestrian. Back to the light switch project: a *truly* imaginative roadblock to the repair would rightly be dismissed as unrealistic by my mind and not considered (A nearby black hole causing a disturbance in the space-time continuum? An evasive and invasive Smurf hiding in the walls?). Examined closer, the things I thought were going to be hard about replacing the light switch were quite tame. After all, I can flick a switch to cut the power and use logic to isolate and make a diagnosis. I've done both of these things countless times!

Attempting a repair for the first time is stepping into something new, so it's to be expected that the Hard Parts from your past are not likely to be the Hard Parts of your future. This is true because past troubles are familiar; you've already either overcome or avoided them. This prior experience and knowledge automatically makes them less potent.

Even if a repair is routine, you probably still won't be able to guess the Hard Part, should it arise. A seasoned electrician would have laughed at my light switch repair problem: a person with those skills would obviously know how to release the clips holding in the wire. This simple repair would have been a 5-minute distraction for them. They might be able to do a thousand with no incidents…

Compared to me, a professional electrician has a much more detailed map of how a light switch replacement should go. But even though their mental model is larger and more complex, filled with the rich details of their training and experience, it's still just as finite as mine. That means that when the Hard Part finally arrives for them, it too will be outside their prior experience and conception. And, just like me, they will be surprised.



A never-ending process: the Hard Part lies outside your experience and knowledge. Every time you encounter an unseen difficulty, your circle of awarness expands to include it. But, the next Hard Part will be outside of that enlarged circle, etc.

(image: © Jason Maxham)

Your Imagination Needs Help

The Hard Part lurks outside the bubble of your experience and what you consider possible. It's therefore hard to guess, so you're going to need help! This is *especially* true if you're working on something dangerous (like electricity), where unforgiving Hard Parts can damage, maim, or kill.

Manuals, troubleshooting guides, and "how-to" tutorials are typically filled with references to the Hard Part. This is also the kind of thing that seasoned repair veterans are quick to point out, if you bother to ask them ("You're doing what?! You better watch out for…"). Between what is written down and what is in the minds of the more experienced, hopefully it's enough to keep you safe.

Yet there is no end to the process: acquiring more knowledge simply pushes the Hard Part further out into the unknown unknowns. That means there will always be accidents and unseen difficulties involving repair. Cultivating a curiosity about the wider world gives you a chance of avoiding these troubles, but be glad you can't extinguish all the mysteries. As long as the Hard Part doesn't harm you, these unexpected moments of discovery are part of the glorious wonder of being alive.

References:

- Header image: Wilson, William A., Photographer (1978). *Bulldozer Freeing Tractor Stuck in Mud.* May, 1978. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/ncr000652/.
- [1] Taleb, Nassim Nicholas (2007), *The Black Swan: The Impact of the Highly Improbable*, Random House.

You Won't Guess The Hard Part was originally published October 18, 2020.

Notes:

Is Troubleshooting A Science?

> To Gillet, it was all sort of an experiment, and "as it turns out I had just enough to do it." The results of the experiment are still inconclusive; Gillet guesses that if he attempted the crossing in the same manner ten times, that he would die on five of the attempts.
>
> **Dave Shively, The Pacific Alone**

When I was a guest on the *I Love Data Centers Podcast*, host Sean Tario broached the topic of methodology, asking me about the "art" in *The Art Of Troubleshooting*. He thought the methodology of fixing things I laid out in my writing seemed to be more formal and rigid than what was implied in the seemingly looser term of "art." I replied that repair can be *very* rigorous and structured. Not everything can be fixed with a simple reboot, so there have been times where I've meticulously collected data, formed & tested hypotheses, and attacked a difficult problem with a disciplined regimen worthy of a Rocky montage.

I also brought up the more human aspects of repair, saying "there's also a part of troubleshooting that's more about experience and intuition…" (54:07). Those processes might sound fuzzier than they actually are: while hardened repair veterans seem to be guided by a divine auto-pilot, it's actually just the deep internalization of their unique fix-it education. Over time, all the victories and setbacks, large and small, are eventually encoded into our unconscious habits.

During that interview, I used the S-word. Of course, I'm talking about **science** (what did you think I meant?). Since then, I've been thinking about the ways of the Troubleshooter; after all, method is very important to understanding how a discipline works. When I give advice about repair, I try to keep reason front and center. However, can we go further and say that troubleshooting is a "science" or that your approach to fixing things should be "scientific"?

Boldly Going Into The Unknown

Most people would rather not peer deep into the soul of a broken machine, meditating on its metaphysical qualities. Instead, they'd prefer to take the easy way out. If you could *instantly* tell someone *exactly* what's causing a malfunction and how to fix it, surely they'd like that. After all, the quicker someone can get on with the rest of their life, the better.

Troubleshooting, in this abbreviated version, can simply mean following the prompt of a machine-generated suggestion. Acting on a "refill paper tray" alert or a "low battery" warning light are low-effort paths that reduce fixing to the simplicity of being aware and following instructions. This relentless march towards making fault-finding an automated process is vastly preferred by consumers, who acquired a product for the *benefits* it confers on their lives, *not* for the joy of repair.

What happens when the most common fix-it problems become readily known and easily solved? I'll tell you: what's left will be the frustrating and interesting troubleshooting conundrums that will lead you into the unknown. Out on this ledge of uncertainty, the best course of action is *at best* a contested matter of opinion. Luckily, this is what makes life interesting: if it all was settled and known, there would be nothing new to learn or figure out. When there's no well-worn shop manual or checklist to follow, we grasp out in the darkness for a framework to give us structure, a general direction to start heading towards. At these moments of uncertainty, when we must choose from the myriad paths in front of us, we tend to fall back on the human virtues of the *art* of repair, like experience and intuition.

That might be fine for the best among us, but what about those with limited experience? What happens if your intuition is weak? I've tried to supplement this gap by becoming a collector of the best troubleshooting strategies, which are general enough to insert your problem into, yet specific enough to nudge you in a direction that has worked for other fixers. However, maybe there's a better way to repair, one based on science instead?

Science, Abused & Praised

In many of the contentious debates in the public sphere, you'll notice there's often a battle over who will claim the mantle of scientific approval. From nutrition to climate change to pandemics, there are points to be scored if you can convince people that the science is "settled" or "on your side."

Unfortunately, for those advocates whose interest in science is merely a tool of persuasion, support for it can be weak and easily crumble in the wake of a contradicting discovery. When the ends (political or social goals) are considered more important than the means (science), the means are easily jettisoned. Cherry picking—emphasizing facts that benefit your cause—is common too. Please leave that tasty harvesting to the orchard.

It's easy to see why there is a fight over who can claim scientific respectability. Science is generally well-regarded in the public's mind; we associate human progress with the rise of science. In an era of spin, propaganda, clever omissions and paid advocacy (both hidden and overt), it's advantageous to be associated with an impartial and verifiable truth. Saying that out loud, I'm getting excited and want in too! It seems we should at least consider that troubleshooting could be scientific…

But First, What Is It?

Let's lay some groundwork. The dictionary has a few different definitions of "science," but for the purposes of this article, I want to focus on the meaning that science is a body of knowledge acquired through the *scientific method*. From the big book of words:

> **science** noun
>
> 3a: knowledge or a system of knowledge covering general truths or the operation of general laws especially as obtained and tested through scientific method

> **Merriam Webster**

If people ask “Can we use science to figure this out?” or “What does the science say?”, this definition is the one that is implied. When you say “science” to the average person, it probably conjures up images of chalkboards, test tubes, rats in cages, and people in white lab coats. That is, people using the scientific method to run experiments, and drawing conclusions from them.

That may seem obvious, but there are broader meanings of the word “science” that include much more expansive notions like a “state of knowing,” or learning around an “object of study.” Especially when it comes to debating who has “the science on their side,” I think what’s usually meant is that a conclusion has been arrived at using the scientific method. But can troubleshooters use the same tools as scientists?

Sidestepping universal solutions: as a Troubleshooter, you don’t need to fix everything, just the broken thing.
(image: Nate Bell / Unsplash)

Troubleshooting Heads One Way

Repair projects proceed toward the specific. What I mean is that troubleshooting is a relentless pursuit to learn the *unique* set of facts surrounding a particular problem. Behind a broad problem description like a “power outage,” “computer crash,” or “engine failure,” there can lie an infinite number of causes and remedies. This means that actually repairing a *particular* instance of these maladies requires an understanding of the situation’s *specific* underlying cause.

When troubleshooting, initial problem descriptions tend to focus on symptoms and an end result that is being thwarted. The person experiencing a malfunction is typically *prevented* from accomplishing something: their car won’t

start and so they can't drive to the store, their oven won't get hot and so dinner is delayed, etc. But for any given symptom, there are an endless number of possible causes. That's because there are only a relatively small number of possibilities for a machine's parts to be correctly configured, while there are an *infinite* number of ways for it to be broken. Think of how many different possibilities could be behind a car not starting: a dead battery, empty fuel tank, faulty wiring, blown fuses, broken belts, etc.

You can see we already have a full plate here: we don't need to add any more to the endless possibilities lurking behind a group of symptoms. However, there's an added problem in simply communicating a malfunction to others: how do you begin to describe what you clearly don't understand? We naturally grasp for a way to relate the problem to something we and others might know, often trying to place the problem in a known category ("power outage," "computer crash," "engine failure," etc.). Using these problem groupings may get recognition from our fellow humans and give us some comfort that we're headed in the right direction, but they're still abstractions which will need to be turned into specifics.

Whatever the source of the haziness, the best fixers quickly recognize and drill down on these generalizations, working to discover the key details so that a solution can be found. This is inevitably a narrowing: you don't need to fix *all machines*, you just need to fix *this machine*. *This* toaster is burning the toast. *This* computer program that calculates the payroll is crashing. *This* nuclear reactor is too hot. The broken system, screwed up in its own unique way, is an obstacle that impedes a specific goal: making perfect toast, paying employees on time, generating electricity, etc.

While completing a particular repair is a drive toward specificity, how you get there is not always a straight line. Troubleshooting requires both the use of induction and deduction, a clever movement between the general and the specific. You may start with some bare facts, like the symptoms of a car not starting. From there, you can "step up", grouping and attempting to fit them into a broader form, like an "electrical problem". Doing so may put familiar tools in your hand and nudge your feet down a known path (e.g., getting out your multimeter, testing continuity, checking the battery's charge, looking for blown fuses, etc).

However, if you aren't successful, you'll have to step "down" again and look for another model to guide your actions (for examples of effective mental models for repair, see the numerous strategies I've written about). Theory leads right into practice: you might have reams of possibilities knocking around in your noggin, but Troubleshooters aren't rewarded for fixing things in their heads. Repair is an applied discipline, so abstractions are useful only to the extent that they rectify the malfunction at hand. "Does the machine work again?" provides a useful check to those who might favor theory over practice.

Science Goes The Other Direction

The output of science seems to run in the opposite direction, from the specific to the general. The most revered scientists are those that discover wide-ranging concepts that implicate huge swaths of reality. Think of Einstein's theory of special relativity or Newton's laws of motion: these are attempts to describe the workings of the entire universe. That's as big as it gets!

To make these broad generalizations, scientists have to be very careful about the particulars of the things they study. Observing just one forest, one human, or one planet may not be enough to confidently say something about *all* forests, *all* humans, or *all* planets. That's because just a single counterexample can bring down the whole edifice of a universal theory. If all swans are supposedly white, then you've got some explaining to do if you encounter a black one.

The underlying goal for most scientific experiments is to be able to take the specifics of a group of subjects and project them onto the larger world. Whether eggs are a good or bad dietary choice for just one person is of limited use to the rest of humanity (sorry, Bob). People want to know if eggs are good or bad for women. Or men (you're welcome, Bob). Or senior citizens (how old are you, Bob?). Or babies. Or high-performance athletes (time to hit the gym, Bob).

Scientists are an observant bunch who always think that they are on the brink of a REALLY. BIG. DISCOVERY. Given that they are watchful and motivated, they are vulnerable to the dangers of heroically extrapolating from what they see. Imagine if science was simply a matter of reporting what you saw, with your observations immediately generalize-able for all things and all times. Science would then just be a matter of reporting, like journalism (well, like *good* journalism!).

Firmly In Control, From A→B

Scientific protocols have safeguards to prevent hasty conclusions. Front and center are measures called "controls" that encourage scientific investigators to take a skeptical approach to their hunches that "A causes B". Merely observing a link between a cause and its supposed effect is only the beginning. If your experimental hypothesis posits that A *inevitably* leads to B, you must also verify that the *absence* of A will not also cause B.

For example, if you want to see the effects of a new drug, you might give the drug to a group of ill rats and note the effect. However, you also need to **not give** the drug to an additional group of similarly ill rats (the control group) and observe the results. Only by comparing the two groups can you know for sure that the drug **alone** is the source of the effects you observe. If both groups of rats recover from their illnesses at the same rate, then the drug's role as the agent of change is in severe doubt. Controls establish a baseline, a benchmark for changes to be measured against.

The use of experimental controls is a cornerstone of the scientific method, the kind of thing you might learn in an elementary school lesson and then promptly take for granted the rest of your life. Sure, there are other bulwarks to prevent the false conclusions of scientists from bothering us: there's always peer review and the possibility of duplicating experimental results in another lab. However, I always thought it was neat that skepticism is built-in to the discipline of science, always encouraging investigators to be circumspect. Designing a controlled experiment at least prompts the thought, "I guess there's a possibility that A might **not** cause B…"

Who's in control of this experiment?
(image: Marion Post Wolcott / Library of Congress)

Your Life Is Out of Control

Before we draw the connection to troubleshooting, I want to show the difficulty of applying the scientific method to a common human problem: choosing how to act in the face of uncertainty and scarcity. In your endeavors as an individual human being, you are singularly involved in an amazing *uncontrolled* experiment. Take a conundrum young people face: selecting a profession to pursue. Let's say you're thinking about becoming a lawyer and wondering if that would be good for you. Can *science* answer this question? What would it take?

I guess we'd start by stating the question a bit more scientifically: "Will becoming a lawyer achieve X for me?" As for X, you're free to pick any measurable metric you want: health, wealth, fame, defendants defended, cases won, homes bought, children begot, spouses divorced, etc. For our experiment, it would start easily enough: we'd follow your legal career from beginning to end, when hopefully you'd retire as Senior Partner from the Big Firm and get your gold watch (or is it a golden gavel?). We'd then assess what happened: did becoming a lawyer cause the desired outcome?

This longitudinal study of your life may be interesting and the tracked parameters could be completely objective, but it isn't science. Not yet, at least. We also want to know: did choosing to become a lawyer, a choice you had among

many professions, *solely* cause the various outcomes, good or bad? That's because, when deciding to become a lawyer, you're implicitly turning down other professions—which may be even better for you! The problem with just observing your career as a lawyer is that these alternatives are not being examined and compared.

Establishing causation from a single observation is problematic, because no variables are being manipulated. Merely noting the outcome of a process, without isolating and changing the inputs, invites speculation and disagreement on what would happen if you did—change the inputs! Luckily, isolating and measuring the effect from a suspected cause are exactly what a well-designed experiment tries to achieve. So, what would it take to get scientific with our career dilemma? Now, we're talking! Let's add some controls and play with some variables: we need to have you **not** become a lawyer (i.e., choose another career) and track those paths as well. In short, we want to compare the *lawyer* you with the *not-lawyer* you.

Our modest plan: we'd start by cloning you. Let's take 1,000 of yourself and send yous on your merry way to becoming a lawyer. We would obviously need to send you to different law schools, as it would be *very* awkward to run into yourself around campus. For 1,000 more clones, we'd choose a variety of alternative careers: hippie jam band festival organizer, plumber, repo man, museum curator, pet detective, etc. Oh, and add another 1,000 clones to sit home and slack off, just in case doing nothing somehow leads to life satisfaction.

Over the years, we'd periodically note some key statistics for all of the various yous: height, weight, marital status, offspring, annual earnings, drug and alcohol use, etc. Finally, at the end of 40 years, we'd analyze all the data. Given that we're going to send you back in a time machine, we would have the luxury of waiting until the very end to analyze all the results from all your clones (it's going to take a while to interview them). Plus, you'd be free to change your mind, at any point, on what it means to have a "good career": along the way, we could easily add data points to collect and questions to ask in our "exit" interviews. Great! Now, hop in, set the dial to the year you graduated from high school, accelerate to 88 miles per hour, and nudge your former self to make the best decision.

This all sounds like a bad sci-fi comedy, but the reason is that we've chosen the wrong tool: the scientific method seems unsuitable for this particular dilemma. You have just a single life to live and this singular nature is not well-suited to simultaneously testing multiple career paths. You can't ask, "Does science say becoming a lawyer is the best option for **me**?" and then forget to run a *controlled* experiment. That's a bush-league move: you'd have to hand over your white lab coat if you made such an amateurish mistake. Minus controls, we can't know the most basic of counterfactuals; therefore, science will have difficulty saying that becoming a lawyer was uniquely the cause of anything, good or bad, in your life.

Since we don't have time machines, another huge problem with "living scientifically" are the omnipresent opportunity costs of simply being alive. You can't go backwards in time for a do-over, so every moment expires all the possibilities of what could have been if you had taken a different path. Your teenage self that's seeking career guidance, needs it *now*. Knowledge gained through scientific experimentation may help you in the future, but it can't be retroactively applied to the past. The creation of scientific insights has a price: setting up an experiment takes resources and time, a cost which you may be unwilling to bear. Lastly, goal-oriented actors (like humans) constantly adjust to incentives and feedback. An inanimate object, like a rock, doesn't respond to its environment like we do (pro tip for scientists: check out those nifty rocks—at least they'll do what you tell 'em). If you found out that you hated practicing law, you wouldn't suffer through a 40-year career as a lawyer—just to complete an "experiment"!

Of course, that doesn't mean that we just throw up our hands; the question of a career path can be answered in many different reasonable ways that are feasible and affordable (and don't require a time machine). You can read biographies of famous lawyers, interview practicing attorneys, get an internship at a law firm, attend a trial at your local courthouse, and tour law schools. You can research starting salaries, analyze job placement rates for various law schools, join a moot court team, and talk to recent law school graduates. You can take a test that matches your personality type with the most suitable occupations, get the opinion of an industrial psychologist, and ask your friends and family if they think it's a good fit. And, if being a lawyer doesn't work out, you can always choose to pursue something else. You can do all of this, but it's not directly applying the scientific method to your question.

Let's see, we could try this spare. Or…conduct a double-blind randomized controlled experiment, then wait for our paper to be peer reviewed and for the results to be replicated by labs around the world. Your call.
(image: John Collier, Jr. / Library of Congress)

The Clock Is Ticking

When it comes to repair and science, you've probably already drawn the parallels with our confused career-seeker. For starters, troubleshooting problems are usually singular in nature. You might have one broken car, or one spewing printer, or one errant computer; it's rare to have 100 broken cars, 100 spewing printers, or 100 errant computers (the kind of numbers needed to run a controlled experiment and get a statistically significant result). Our resources are finite and we employ machines for our purposeful ends, so it's rare to have copious extras on hand to test counterfactuals and provide adequate experimental controls. Also, just like our aspiring attorney, we have time constraints: we need things fixed *now*. That broken machine is preventing you from accomplishing something important, so the favored solutions will be expedient (i.e., the opposite of the slow, careful introspection needed for scientific exactness).

I do want to point out that there are scenarios where repair can successfully interact with the scientific method. This typically happens in large-scale industrial situations, where you're dealing with masses of identical components. For example, imagine that you run a data center, with thousands of computers containing many more thousands of hard disk drives. You buy in bulk, so you have many identical drives of the exact same make and model being used in these servers. Plus, the sheer size of your deployment makes it economical for you to attempt and track repairs in-house. This is a great setup for a rolling, controlled experiment!

That's because a data center with thousands of hard drives will have failures—constantly. I managed a server farm that would be considered modest by today's standards, and not a week would pass without at least one disk giving up the ghost. You can see how this setup would be ideal to practice science: with each new failure, you could experiment with various types of repair (replace the platter? controller? connector? firmware?). If you were tracking operational data

on your drives, you could begin to compare the effectiveness of these different kinds of repairs as you put the drives back in service and monitored their health. Finally, that pile of broken hard drives in the corner—there's your control group!

Beyond end users, manufacturers have both the incentive and scale to pull off scientific troubleshooting studies. Again, if you're dealing with masses of duplicates, you have an ideal situation for the segmentation needed to test alternative repair theories. Before a new product is born, the prototyping and testing phase offers ample opportunities to track breakdowns and the efficacy of various responses to them. When a product is released "into the wild", real world usage by customers offers even more chances to collect and analyze failure data in a scientific way. Automobile makers are a great example of this setup: there might be millions of a particular model on the road, and a network of dealership service departments collecting data could be your "lab notebook". Just like a control group in an experiment, the identical nature of mass produced goods provides a clear lens to observe the result of various changes: over time, the completely homogeneous group of test subjects can highlight the effectiveness of different ways of dealing with a failure.

Another field that is conducive to a quasi-scientific troubleshooting approach is software. When I was writing code, there were many opportunities to use controls, just like in a real laboratory. I could take a malfunctioning program, attempt to fix it by making a single modification (note: the "change just one thing at a time" troubleshooting principle is straight out of the science playbook), and then test it against prior versions. Running these tests in the same environment (hardware & operating system version) provided consistency to the results; this would be analogous to making sure a lab was cleaned and set up the exact same way for each round of an experiment. Since I worked on a computer cluster, running thousands of tests simultaneously was both fast and easy. This way of testing software has all the elements of a controlled experiment: manipulating a single variable (making a change to the program), gauging its effects (examining the altered program's output), then comparing the results to the control group (noting differences to prior versions of the program, *without* the modification). This methodology was great at isolating the consequences of a particular change—exactly what scientific experiments aim for!

If you are fortunate to be working on a system that has genuine science-based troubleshooting data available, please take full advantage. But, I suspect that this won't be the case for most repairs you undertake in your life. Most fixes, even those recommended by manufacturers, are discovered through trial and error, or are simply the application of logic to a Machine Model ("this is how it's supposed to work, so do this when it breaks…"). It would be great if every fix could be verified through a controlled experiment, but again, scarcity has the last say. There is a cost to create truly scientific repair information, and likewise for you to seek out and employ it in your hour of need.

References:

- Header image: Rothstein, A., photographer. *A milk tester for the Triple B milk testing associations. His equipment includes test tubes, pipettes, a centrifuge and laboratory apparatus seldom found on the farm.* Black Hawk County, Iowa. 1939. Nov. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017779311/.

Is Troubleshooting A Science? was originally published March 5, 2021.

Notes:

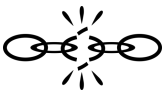
Part 2: Strategies

Plot a course—straight to fixed!
(image: Library of Congress)

> If it's broke, you should try to fix it.
>
> **Mike McCormick**

The Strategies are time-tested, practical recipes for troubleshooting. They are presented in no particular order: when troubleshooting an actual problem, don't expect to go through them serially (although you're welcome to do that as you're learning). Once you are familiar with them, you'll intuitively jump to the one you think is going to have the biggest payoff. Knowing where to start is the "art" in *The Art Of Troubleshooting*. Also, keep in mind that you might need to combine several strategies to actually fix something. Mix and match as needed! If there's any "magic" here, it's that the sum of these very simple ideas will make you a very good troubleshooter indeed.

The Order Of Things



> I think before you start troubleshooting, you have to understand the system, because you can waste a lot of time, checking this and checking that. But if you know that A comes before B, you won't mess around with B without checking out A first.
>
> **Dan McCormick**

The expression "there's more than one way to skin a cat" nicely summarizes the "change the order" troubleshooting strategy. The concept is to alter the sequence of steps being taken when starting up, configuring, or operating a machine. Among these multiple pathways there will be some combinations which work, and some that won't. Those pathways may be identical, except for the **order** in which things happen. By simply rearranging the progression of events, you can sometimes get a system working again.

When using this strategy, don't feel like you have to necessarily understand why one pathway works and another fails. Fix first and ask questions later. The ability to understand why one particular sequence works and another doesn't may be beyond even the understanding of the original designer. Given that so many products today are integrations of parts cobbled together from multiple manufacturers and outsourced teams splayed around the globe, this isn't surprising. That is, not all use cases or interactions among components may be well understood. As a troubleshooter, you should be focused on the outcome of getting things running again. Answering the question "Why?" has an economic component to it: sometimes there is a hefty cost in terms of time and effort to fully understand a problem. When knowledge is pricey, you may be better off in blissful ignorance!

Unless you've read the manual, messing with the startup sequence of an oil refinery is not recommended.
(photo: Walter Siegmund / CC BY 2.5)

On Startup

If you have a machine with multiple subsystems, interactions between them while they're starting up may cause a failure. Consider a simple web site that consists of a database, web server and application server. These 3 elements can be started up in the following different ways:

Startup Sequence	Order of Startup		
	1st	2nd	3rd
A	Application Server	Database Server	Web Server
B	Application Server	Web Server	Database Server
C	Database Server	Application Server	Web Server
D	Database Server	Web Server	Application Server
E	Web Server	Application Server	Database Server
F	Web Server	Database Server	Application Server

That's just 3 components and look how many different ways there are to initialize the system (6 to be exact, lettered A-F in the table above). The complexity grows exponentially, with the factorial of the number of subsystems (*n*) expressed as:

n!

Here's a table that shows you just how quickly the potential complexity of a startup sequence can grow as the number of subsystems increases:

# of Subsystems	# of Unique Startup Sequences
1	1
2	2
3	6
4	24
5	120
6	720
7	5,040
8	40,320
9	362,880
10	3,628,800

Astonishing, right?!

Back to our example: you've just upgraded the application server and, unbeknownst to you, the new version of the application server checks to see if the web server is running first. If it isn't, the application server will mysteriously quit without explanation. This means certain startup sequences will result in a failure (specifically: A, B, and C), which I've highlighted in red in the table below:

Startup Sequence	Order of Startup			Starts up successfully?
	1st	2nd	3rd	
A	Application Server	Database Server	Web Server	NO
B	Application Server	Web Server	Database Server	NO
C	Database Server	Application Server	Web Server	NO
D	Database Server	Web Server	Application Server	YES
E	Web Server	Application Server	Database Server	YES
F	Web Server	Database Server	Application Server	YES

However, sequences lettered D, E, and F will be fine. If you had encountered this problem "in the wild" and simply cycled through the various startup sequence possibilities, you could have fixed the problem.

Tweaking The "Change The Order" Strategy

As you can see from the table showing the growth in the number of startup possibilities, even with just 5 sub-systems, there are 120 unique startup sequences. That's a lot of things to try! A shortcut is to first make an educated guess as to the failing subsystem. In any troubleshooting exercise, there will frequently be supplementary evidence pointing to the culprit. In the example above, it could have been that the application server was missing from the list of running processes. After you've made your guess, try these 2 things:

1. Move the suspicious subsystem to the **first** position in the startup sequence.

2. Move the suspicious subsystem to the **last** position in the startup sequence.

Putting the suspicious subsystem first or last gets quickly to the underlying reason as to why the "change the order" strategy works: something needs to be present or absent for the subsystem to function. Putting it first ensures that, with all the other subsystems off, nothing can interfere with its ability to function. However, if the failing subsystem requires another subsystem to function, starting it up at the end gives it the greatest chance of meeting this condition because all possible dependencies will be satisfied.

For the ultimate example of a complicated initialization sequence, consider a modern oil refinery. It can take weeks to bring a refinery on-line from a completely cold start. Of course, if you tinker with that kind of startup sequence as a troubleshooting strategy, you better know what you're doing!

When Configuring

The same principle applies to configuring a machine; changing the order in which you apply your configuration options may make the difference between it working and it not working. Take the example of a network router, which you want to set up as such:

1. Add an entry to the routing table.
2. Set the default gateway.
3. Block connections from all outside networks.
4. Allow connections from select internal networks.

You notice that, after applying these options in the order listed above, the router will consistently crash and reboot itself after applying rule #2 (i.e., you never make it to #3). Let's say that, after playing around with the above order, you can get the router to work by placing rule #2 at the end of your configuration recipe. According to what you know about what the router should be capable of, there's no good reason why you shouldn't be able to do the configuration in your original order, but keep in mind that troubleshooting is dealing with reality. The reality of the situation is that no amount of hemming and hawing over how the device *should work* will make your original configuration stick. You tried it. It didn't work. Move on.

Also note that the math concerning the number of possibilities is the same: if you want to count the number of unique ways to apply a set of configuration options, it's also $n!$. Again, if the number is large you'll want to make an educated guess as to the option most likely to be causing the problem and move that to the front or to the end.

When Operating

This general strategy is also effective, for the same reasons, when changing the order in which a system does its work. You may have a collection of tasks (A, B, C) that need to be done, but the final result is independent of the sequence in which these tasks are completed. That is, A → B → C results in the same output as all the other possibilities: A → C → B, B → A → C, B → C → A, C → B → A, and C → A → B. While the order of a workflow may not matter theoretically, as we've seen, sometimes in *practice* it makes a big difference.

Imagine a computer program that validates data coming from a form on a web page, checking that a customer's name, address, and telephone number are all formatted correctly for a database. Offhand, you'd say that it shouldn't matter in what order these validations are made: all three fields must eventually be checked for the data to be made safe for the database. However, let's say that our telephone number validator uses a code library with a bug that causes corruption for the text validator. That means that the program will only work if the telephone number is validated last. Of course, this isn't how code libraries are supposed to work: they *should* be bug free and well-tested! However, don't let a fixation with "should" get in your way and blind you to considering options that achieve your end goal. Can you rearrange the steps of your workflow, achieve the same result, and do an end-run around your issue all at once? Many times, the answer is "Yes!"

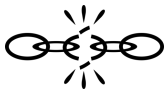
References:

- Header image: Lee, R., photographer. (1938) *House erection. First panel off the truck is made into an easel for the other panels. The panels will be unloaded in sequence for erection to avoid handling. Southeast Missouri Farms Project.* New Madrid County, Missouri, United States. May, 1938. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017736764/.

The Order Of Things was originally published September 28, 2011.

Notes:

Skillful Questioning, Part 1



> They would say, “It rattles in the back.” I would say, “That’s a lot of area.”
>
> **Gerald Quade**

Skillfully interviewing people to obtain meaningful information about a malfunction is an essential skill for the troubleshooter. Especially if you are called in to look at a complicated system, perhaps spread across geographies or teams, you may initially be spending a good deal of time simply talking to people and reading incident reports. Asking the right questions at this stage may mean the difference between a quick fix and a fruitless, time-consuming odyssey. Discovering where to attack is critical, a process that is aided by smart queries.

Let me introduce a powerful tool that goes beyond “asking the right questions,” one that will help you in all of your communications: insights of the “meta-model” from the field of Neuro-linguistic Programming (NLP). If you’re unfamiliar, NLP is the “science of subjectivity” and was born when its founders Richard Bandler and John Grinder studied three extraordinary therapists. These therapists (Virginia Satir, Fritz Perls, and Milton Erickson) were able to achieve amazing results with their clients in short periods of time. After closely observing them, Bandler and Grinder constructed a model of how their methods worked, the core of which was how they used language. The insights extracted by Bandler and Grinder have been applied to all kinds of problems: therapy, personal change, sales, management, etc.

For our purposes, we’re primarily interested in the contributions to NLP made by noted family therapist Virginia Satir. Satir was able to ask questions that uncovered hidden information and compelled her clients to change by revealing

their limiting beliefs. Bandler and Grinder distilled the very precise way Satir used language (in particular, her style of questioning) down to its essential elements and dubbed it the "meta-model." The meta-model is a series of verbal patterns and responses to them in the form of questions. The patterns indicate that information is missing, which the questions attempt to recover. The questions can also be used to find and confront discrepancies between a reporter's *beliefs* and reality.

Troubleshooting often starts like this, with an interview.
(image: Maj. Dale Greer / Wikimedia Commons)

You may be wondering why I'm introducing you to techniques from the world of therapy, when presumably this is a work about troubleshooting. Don't worry, you're not going to ask anyone to lie down on a leather couch and ask them about their mother, Freud-style. But, since the theme of this work is "machine problems are actually human problems," it's not surprising that the tools needed to mend broken relationships and dysfunctional families are also useful to the troubleshooter. The reason for this is that the entry point to nearly every troubleshooting exercise starts with a communication, usually from a human or from a (human-built) monitoring system. Your grandma calls you and says the Internet is broken, one of your tenants knocks on your door and complains that his lights are out, your phone buzzes with texts about a problem at the plant, etc. Machines don't talk, so you better get good at listening to those that speak on their behalf (i.e., humans). That means going deep in the ways people talk about their problems. As you will see, it really doesn't matter if we're talking about our frustrations with a family member or a broken-down car, because the difficulties people have describing their problems are universal.

Knowing the "meta-model" has been invaluable to me, both at work and at home. Understanding it will make you a much better troubleshooter and a more effective communicator in all situations. After you learn it and apply it to your life, you'll start to get a sense of what's missing when people are talking to you. When it comes to troubleshooting, you will find that people, either consciously or subconsciously, delete information when reporting issues. They may feel partly responsible for the breakdown of a system, be protecting a fellow employee or simply have a terse communication style, thinking they're doing you a favor by being brief. No matter the origin of such obfuscation, your job as a troubleshooter is to cut through it all. The meta-model is the knife that will help you do that, with the the underlying theme being the recovery of the hidden information and obtaining clarity from distorted descriptions.

So, let's walk through the components of the meta-model, with help from one of the first books I read on the subject: *Introducing NLP* by Joseph O'Connor and John Seymour. As we go along, we'll see how the meta-model relates to interviewing while troubleshooting:

Unspecified Nouns

> Unspecified Nouns are clarified by asking: "Who or what specifically…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [1]

Consider the two following descriptions of the same problem:

1. "It won't start."
2. "In my garage, when I turn the key in my 2011 Porsche Carrera 4 GTS Cabriolet, the following happens: I hear a grinding noise, the oil light on the dash flashes, the motor catches, runs for 2 seconds and then stops."

The two statements above are both grammatically correct. They also truthfully describe the same event, but one is infinitely more useful to a troubleshooter. The first description has omitted a lot of details about the "it" that is referenced. These details have been turned into specifics in the second description. The key to recovering information about unspecified nouns is to ask for more information: "What specifically…?" In this example above, what exactly is "it," to whom does it belong and where is it located? If the car was at the bottom of a lake or on the moon, would that be good to know and would you still be surprised that it didn't start?

You also may encounter the passive voice, which deletes the active subject of a sentence. This may give the impression that something just happened on its own, with the narrator being a helpless observer:

- "The paper tray was overstuffed."

Paper trays don't become overstuffed on their own. Who (or what) overstuffed the tray and on which printer was the tray attached?

Unspecified Verbs

> Unspecified Verbs are clarified by asking: "How specifically…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [1]

When troubleshooting, it's frequently of critical importance *how* something was being done before a failure. Almost every machine you can think of has a right way and a less-than-right-way of using it. Some machines have a wide latitude in how they can be operated and others require that strict procedures be followed.

- "She shut down the computer."
- "I'm trying to start the motor right now."
- "He replaced the alternator last week."

You can see that information about the "how" is missing from the examples above. Was the computer shut down by using the shutdown command from within the operating system? By pushing the on/off switch? By pulling the power cord? By shutting off a circuit breaker at the power panel? By an EMP from a nuclear explosion?

- "How specifically did she shut down the computer?"
- "How specifically are you starting the motor?"
- "How specifically did he replace the alternator last week?"

Comparisons

> Comparisons are clarified by asking: "Compared with what…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [1]

Buckle up for unspecified comparisons, because they can contain all kinds of assumptions and expectations.

- "The website is slow."
- "The engine is idling fast."

Saying "the website is slow" contains an implicit comparison: "slow"…compared to what? The way the website was operating last week? Slow compared to Google? Also, is their definition of "slow" based on a measurable and objective metric? Your job is to discover the missing context by asking for more information, and to corroborate these kinds of assertions with evidence. Knowing what something is being compared to can be critical, especially if are you being asked to restore it to an ideal that is unattainable.

- "The website is slow, compared to what?"
- "The engine is idling fast, compared to what?"

Judgements

> Judgments are clarified by asking: "Who is making this judgment, and on what grounds are they making it?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [1]

In a similar vein to comparisons, you may encounter implicit judgements when interviewing:

- "Obviously, a low voltage condition caused the failure."
- "Clearly, the burner isn't heating up fast enough."

If you replace the word "obviously" with "it is obvious," it becomes clear that there is critical information missing here: namely who this is obvious to and why! If you're going to accept someone's judgements when troubleshooting, do it willingly, not by having it smuggled in under the cover of vague language.

- "Who says that low voltage caused the failure, and how do they know that?"
- "Who says that the burner isn't heating up fast enough, and how do they know that?"

If you can't put your noun in here, it's probably a nominalization.
(image: Erich Ferdinand / CC BY 2.0)

Nominalizations

> A Nominalization is clarified by turning it into a verb and asking for the missing information: "Who is nominalizing about what, and how are they doing it?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [1]

Nominalizations are verbs describing processes that have been turned into nouns (e.g., education, communication, etc.). It's a bit of semantic magic to take a very *active* process (like "educate") and turn it into a *static* noun (like "education"), giving the concept a veneer of fixed permanence. You can apply the "wheelbarrow test" to find out if you're dealing with a nominalization. If it can't be put into a wheelbarrow, that is, if the noun can't be tasted, touched, smelled, heard, or seen, it's likely a nominalization. Pay close attention to nominalizations because they can contain vast differences between people's maps of the world. It's easy to see why, because an abstract concept like "education" is a collection of activities that could include: teaching, testing, learning, reading, doing homework, on-line courses, public or private funding, etc. Based on your background and beliefs, you'll likely have an opinion on what the essential components should be. Thus, "education" can mean very different things, depending upon who you're asking. Of course, nominalizations are extremely *useful*! It would be very tedious to say "teaching, testing, learning, reading, and doing homework" instead of "education" every time you wanted to talk about…education.

- "The investigation into the structural failure turned up nothing."

In the context of troubleshooting, just be aware of nominalizations and the vast amount of information they leave out. You'll want to find out the who, what and how behind the nominalization by asking for more details. Because, in the example above, an "investigation" could mean: "I made a phone call to some guy and left a voicemail," or "We had a team of 25 structural engineers studying the problem for 2 years that culminated in a 1,000-page report."

- "Who investigated what about the structural failure, how was the investigation conducted, and what conclusions were reached?"

The rest of the meta-model is covered in: Skillful Questioning, Part 2.

References:

- Header image: Trikosko, M. S., photographer. (1975) *President Gerald Ford taking questions from reporters during a press conference at the White House, Washington, D.C.* Washington D.C, 1975. Jun. 9. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2005684016/.
- [1] Joseph O'Connor and John Seymour, *Introducing NLP* (London: Thorsons/HarperCollins, 1990), pgs. 93-96.

Skillful Questioning, Part 1 was originally published October 4, 2011.

Notes:

Skillful Questioning, Part 2

> I don't put much stock in initial reports, except that something isn't working up to par.
>
> **Mike McCormick**

Let's continue describing a truly invaluable way of gaining clarity by challenging vague language (started in Part 1, which introduced the "meta-model"). In any context, but especially when interviewing people while troubleshooting, getting the right information about the problem at hand is critical to correctly directing your efforts.

I used to play a little game when I was learning the various parts of the meta-model. Each week, I would focus on one of the meta-model's language patterns and score one point whenever I observed someone making a statement that included the pattern. A few times, I got so excited that I would blurt out "unnnnnnspecified nouuuuuuuuuuun!" or "mooooooodal operator of possibility!" and do a little victory dance. Judging by the look on people's faces, I don't recommend this. I forgot the words of the master:

> Let your workings remain a mystery.
> Just show people the results.
>
> **Tao Te Ching (verse 36)** [1]

So, go ahead and play the meta-model game all you want, but win *silently*. Anyway, let's finish the list of meta-model language patterns (from *Introducing NLP* by Joseph O'Connor and John Seymour) and see how they apply to troubleshooting.

Sherlock Holmes always uncovered the right details. Clearly, he knew the meta-model.
(image: Mr. Frosty Man / CC BY-ND 2.0)

Modal Operators Of Possibility

> Modal Operators of Possibility – "I can't" – are clarified by asking: "What would happen if you did…?" or, "What prevents you…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Modal operators of possibility are statements concerning what is possible: they are words like "can," "cannot," "possible," and "impossible." But, they're just words! As such, they may or may not reflect the real world. As a troubleshooter, don't get sucked into someone else's reality and accept their ideas about what is possible or not possible without proof (see the virtue of skepticism for more on this topic). Especially on the "not possible" side of the ledger, you'll frequently encounter people who are locked into the well-rehearsed sequence of their workflow. Noting someone's routine is fine but remember that effective troubleshooting frequently requires deviating from the "official script" (for example, by changing the order in which things happen).

- "I cannot get the engine to start."

These words about possibilities, like the universal quantifiers discussed below, also don't allow for exceptions. Someone who says they "cannot" do something, or for whom a task is "impossible," is using language that is constraining and final. By the way, I'm not saying that you can overcome an actual, respect-the-laws-of-physics type of limitation simply by asking a clever question. The person who says they "can't" may very well be right about what is possible. However, if they are being held back simply by the way they're framing the problem, you can help them get unstuck by throwing a jarring question into the mix.

The way to counter modal operators of possibility is to say "Show me!" and ask, "What would happen if you did?" This

simple question is all it takes to pierce these limiting beliefs and get the mind working on alternatives.

- "What is stopping you from starting the engine?"

Modal Operators Of Necessity

> Modal Operators of Necessity – "I mustn't/I have to" – are clarified by asking: "What would happen if you did/didn't…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Modal operators of necessity concern needs and involve words like "should," "must," "ought," and their negations: "should not," "must not," and "ought not." Like the modal operators of possibility, these words involve presuppositions that may not be helpful or true. They also hint at a rule or procedure being followed, but only indirectly. You want to bring these implicit assumptions out into the open for examination.

- "You should turn on the backup battery before engaging the ignition."

Frequently, you will discover that the person has no better justification for their actions than "that's what I was told to do" or "that's the way I've always done it." The backup battery is always turned on before you engage the ignition because…the manual says there will be an explosion if you don't? A leprechaun told you to do it that way? Again, successful troubleshooting often requires straying from the well-trod path, so you should keep in mind that the "way we've always done it" is not necessarily the only way to do it.

- "What will happen if you didn't turn on the backup battery before engaging the ignition?"

Universal Quantifiers

> Universal Quantifiers are questioned by asking for a counter example: "Has there ever been a time when…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Humans like to generalize: it's faster, easier, and makes you sound confident. To a troubleshooter, however, exceptions will frequently point the way to a solution. When you hear words like "all," "every," "always," "never," or "none," recognize that someone might be generalizing. Universal quantifiers like these allow no room for exceptions. If they are true, that's good to know. If not, you'll want to understand the exceptions.

- "This stupid thing never works!"

To unravel a universal quantifier, ask for an exception. In this case: "Has there ever been a time when…that stupid thing worked?"

Operational data is another effective way to contradict universal quantifiers. Take a statement like: "The temperature in the warehouse has **never** exceeded 82 degrees in the month July." If you had a graph or table showing the recorded temperature during that time period, you could easily see if this universal, no-exceptions-allowed statement is true or false.

I hear universal quantifiers all the time when asked to help fix problems. People like to exaggerate and, particularly if they're angry that something is broken, they can really lay it on thick with words like "never" and "always."

Complex Equivalence

> Complex Equivalences can be questioned by asking: "How does this mean that?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Linking two statements together to mean the same thing is called a "complex equivalence"; the speaker is leading you (perhaps unintentionally) to make an association between two things. But is the connection real?

- "The workers are taking a break. The assembly line is stopped."

When it comes to troubleshooting, never underestimate the ability of people to jump to conclusions. Worse yet, *you* may attempt to fill in the blanks and reach false conclusions all by yourself (even if the speaker wasn't making that leap). The mind loves to fill in missing gaps with its own interpretations! In this example, it may be true that the assembly line is stopped and the workers are taking a break. But, that doesn't necessarily mean that one caused the other. The assembly line could be idle because of a work break, but it could also be the other way around: perhaps the workers are taking it easy because the assembly line is broken.

- "How does the workers taking a break mean the assembly line is stopped?"

Presuppositions

> Presuppositions can be brought into the open by asking: "What leads you to believe that…?" and filling in the presupposition.
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Making your way in this world demands reliance on beliefs and expectations. For example, imagine the paralyzing fear you would experience if you didn't simply trust in the Law of Gravity. It's much easier and less stressful to assume that gravity will behave like you expect. Unlike our experience of gravity, other beliefs we hold are highly contextual (i.e., they're only true in certain cases) or, sorry to be the bearer of bad news, they're simply false.

There's nothing wrong with presuppositions *per se* as they are essential to life. But, if you've been called in to fix something, it's likely that someone's expectations have been violated (unless, of course the expectation is that all machines eventually break down—now that's a very healthy belief!). Therefore, detecting false presuppositions are critical to your success as a troubleshooter. Words and phrases typically associated with presuppositions are: "since," "when," "if," "realize," "be aware," "ignore," and suggestive questions beginning with "why."

- **Statement:** "When this gauge reads 100 lbs./sq. in. of pressure, then turn the release valve." → **Presupposition:** waiting for 100 lbs. of pressure to build is needed before turning the release valve.
- **Statement:** "Why are you trying to wreck the motor?" → **Presupposition:** I'm wrecking the motor. And, I'm doing it intentionally!
- **Statement:** "Ignore the green wire. Are you going to cut the blue wire or the red wire to defuse the bomb?" → **Presupposition:** I must choose either the blue or red wire, as opposed to doing something else to stop the countdown timer. FYI, always choose the red wire.
- **Statement:** "Be aware of the fuel tank reading." → **Presupposition:** the fuel tank reading is accurate and important.

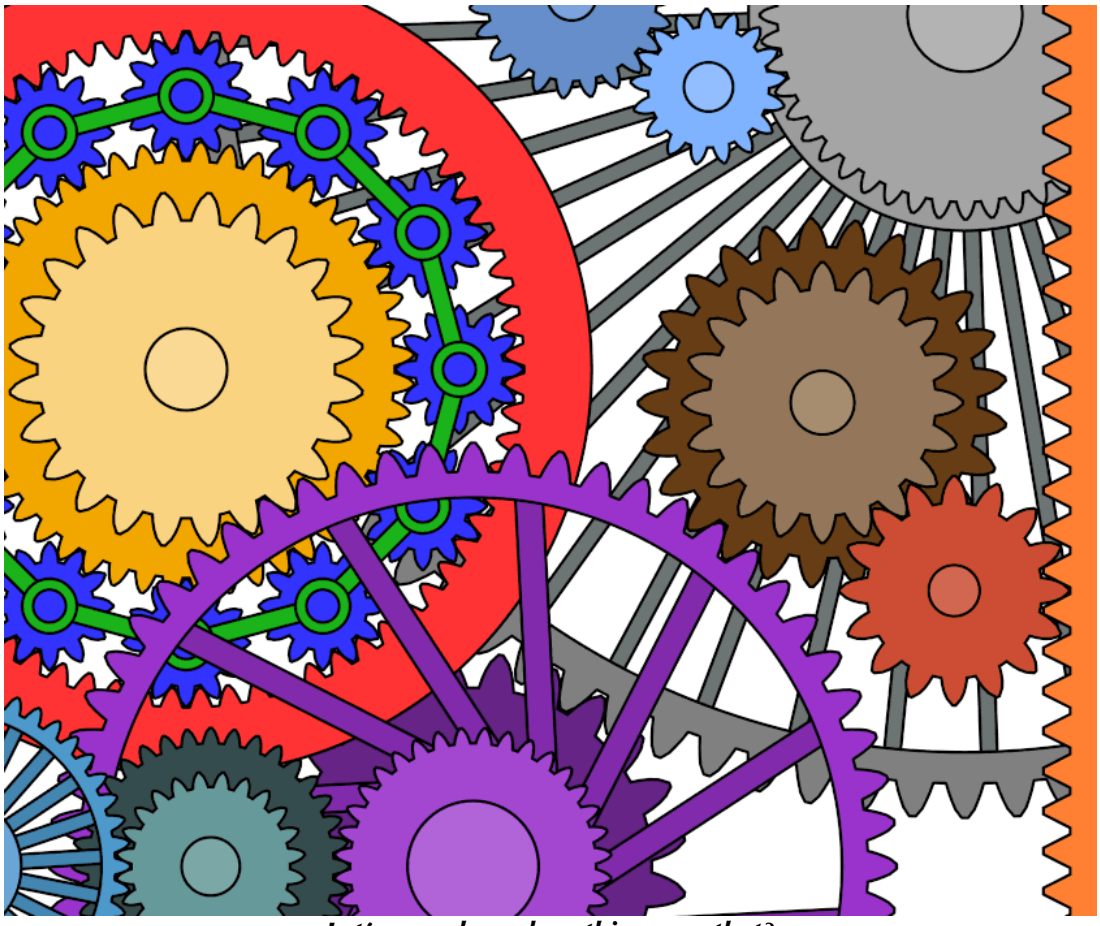
Cause And Effect

> Cause and Effect can be questioned by asking: "How exactly does this cause that?" or "What would have to happen for this not to be caused by that?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Cause and effect is a powerful way of explaining our world. If A causes B, then you've found the lever to prevent B, or make sure it happens all the time, depending on whether B is a good or bad thing!

- **Statement:** "The car would have started, but the battery was low." → **Cause/effect assumption:** the depleted battery was the sole reason the car wouldn't start.

- **Statement:** "The contamination resulted from a leak in Tank #12." → **Cause/effect assumption:** the only source of contamination was from Tank #12, *and* the leak caused it.
- **Statement:** "An abnormally high number of database operations used up all available memory, making the server unresponsive." → **Cause/effect assumption:** the database used up all the memory *and* the server crashed because of it.



Let's see…how does this cause that?
(image: Jahobr / Wikimedia Commons)

With respect to problem reports, "cause and effect" statements are a Big Deal. You'll want to thoroughly investigate them because, subconsciously, you will be drawn to them like Germany to David Hasselhoff. You won't be able to look away! If you're not on guard, these statements will burrow their way into your head and take over the troubleshooting process, leading you on the proverbial "wild goose chase."

Before that happens, step back and ask some skillful questions:

- **"How does this one thing cause the other?":** This may reveal new possibilities of causation, especially if coupled with the question "What would it take for A to *not* cause B?"
- **"Would you please describe the process that makes A cause B?":** The details revealed here may show a weak link (or, none at all!) between A and B. Additional contributing factors may also be uncovered.
- **"Has there ever been a situation where A didn't cause B?"**: If you can point to a situation with the same cause but not the same effect, there may more to the story than a simple A → B explanation.

Mind Reading

> Mind Reading is questioned by asking: "How exactly do you know…?"
>
> **Joseph O'Connor & John Seymour, Introducing NLP** [2]

Presuming to know what someone else is thinking or feeling can be filed under the category of "mind reading."

- "Jim doesn't care about the malfunction and won't know how to help us…"

In the context of troubleshooting, you may encounter this type of heroic extrapolation when brainstorming a list of people who could help you out. However, don't back down and pass over a potential ally without first questioning why. You may hear all kinds of weird reasons why you shouldn't contact the one person who can fix your problem!

- "How exactly do you know that Jim won't help us?"

Conclusion

We all have different models of the world that are reflected in the language we use. The meta-model gives you insight into those differences. By assuming you don't know exactly what a person's words mean, you can dig deeper for the precision that's so often needed when troubleshooting.

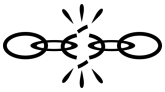
References:

- Header image: Bain News Service, P. *Tom Gibbons interviewed.* [No Date Recorded on Caption Card] [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2014716240/.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 36.
- [2] Joseph O'Connor and John Seymour, *Introducing NLP* (London: Thorsons/HarperCollins, 1990), pgs. 97, 99, 101, 102, 104, 105.

Skillful Questioning, Part 2 was originally published October 11, 2011.

Notes:

Put It Down And Come Back To It Later…



> That's why talking a walk or having a cup of coffee is an important problem-solving technique. Not solving the problem is an important part of solving the problem. Because then you come back and you're like "Oh, that's why! There's the answer!"
>
> **Alex Chaffee**

You've probably been told at least once in your life: "You should sleep on it."

It's a solid, common-sense thing to do before making a big decision. I sometimes imagine asking Wilford Brimley for advice and, besides sternly telling me to eat a healthy breakfast of whole grain oats, I bet one of his go-to nuggets is "sleep on it." As I've hinted at before, troubleshooting has lessons for life, and conversely life's lessons have something to teach the Troubleshooter. The benefits of "sleeping on it" are available in either realm: you gain perspective and insight by temporarily stepping away from your problems.

You may have implemented this strategy by accident. That is, you hacked away at something until you:

1. Got too hungry.
2. Got too tired.

3. Got a call from your sweetheart, asking if you were coming home soon.

Circumstances forced you to take a break and so you put down your tools and came back to the problem later. You might have noticed that when you did return to the issue, after eating or sleeping or doing damage control for being late to dinner, frequently the solution was staring you in the face! Why is this?

Believe it or not, this is a legitimate troubleshooting strategy.
(image: Timothy Krause / CC BY 2.0)

While you were distracted, your unconscious mind was doing some work on its own in the background. The "why" and "how" hasn't exactly been nailed down yet, given that the science on the unconscious mind's problem-solving abilities is an area of ongoing debate and research. Beyond a vague notion that some kinds of complex mental tasks may benefit from either conscious or unconscious processing, the understanding of why (especially on the level of what is actually happening in the brain) is probably a long way off. That's okay though, we don't have to understand how electricity works in order to benefit from its use. In the context of sleep, we know that unconscious processing is critical to the consolidation of memories in the "acquisition → consolidation → recall" model of learning. It also makes it clearer why your perspective is changed the morning after you've "slept on it": you probably assimilated a lot overnight! Troubleshooting can involve a lot of learning, especially if the failure condition or system is new to you, so it's not surprising that sleep is beneficial.

There's also a concept called "fractionation," used by hypnotists, that I also think is relevant here. Hypnotists know that to put someone deeper into a state of trance, it's useful to take them out of trance temporarily. Then, when you put them back under, they'll go in even deeper. Intense troubleshooting can closely resemble a state of trance: you're very focused on the problem, often to the exclusion of the world around you. Sometimes, I've been working on fixing something and been concentrating so intensely that I didn't even notice someone enter the room! However, a plateau will eventually be reached and the only way to get any further is to break your concentration, entertain a distraction and then go back. And, when you do, it'll be deeper.

You might also be "looping," trying the same thing over and over again with little progress. We usually think of being interrupted while working as a bad thing, and, in our high-tech world of fractured attention spans, most of the time I think that's true. However, if you're chasing your tail, an interruption can be very useful to breaking the pattern. In formulating this strategy, I was impressed by the number of times I'd be sitting in my office, "looping" on a problem and the phone would ring or someone would knock on my door. We'd chat for a few minutes and then I'd turn back to the problem and *BOOM!,* the answer would hit me.

When diminishing returns set in, can you punch out, relax, and let your subconscious do the rest?
(image: Marjory Collins / Library of Congress)

If I may add something original to the observation that being forced to step away from a problem is useful when troubleshooting, it's this: **do it intentionally!** Don't wait for yourself to get too hungry or too tired or for that angry phone call to get some much-needed distance from a problem. That's right, you will be a much more effective troubleshooter if you take regular breaks. Some people like to use a timer (e.g., the Pomodoro Technique) to know when to step away. I prefer a more open-ended approach based on how things are progressing: I now have a good sense for when diminishing returns of spending additional time on a problem have taken hold. For me, that's the best time for a break. This ties in nicely with the research about the role of conscious and unconscious processes while learning: there is a point when you'll become saturated on either front (most likely the conscious side). If you've noodled all day straight on a problem and are stuck, it's unlikely you'll make further progress without some intervening time away for unconscious processing (like sleep). To do that intra-day, I usually like to take a short walk. You can choose your own pleasant distraction.

If you're a Type A fidgeter who must be busy every second and don't think you can handle stepping away, try this instead: switch to a different aspect of the problem. Move on to another component or subsystem that needs inspection, read the manual, collect some data, or do some research on the Internet. You can also keep going by simply switching strategies which, after reading *The Art Of Troubleshooting* beginning to end, you'll have many options from which to choose. Basically, do anything that will distract yourself from the thing you're stuck on, so when you return to it you'll see it with a fresh pair of eyes. Changing course is also good for those cases where it would be, how should we say, unprofessional to take any sort of break. In the middle of a crisis you can't really tell a client who's losing thousands of dollars every hour that you'll be unavailable while you're "fractionating."

By the way, I followed my own advice while writing this: several times I came to an impasse, put it down, and came back to it later. I'm sure it's better for it.

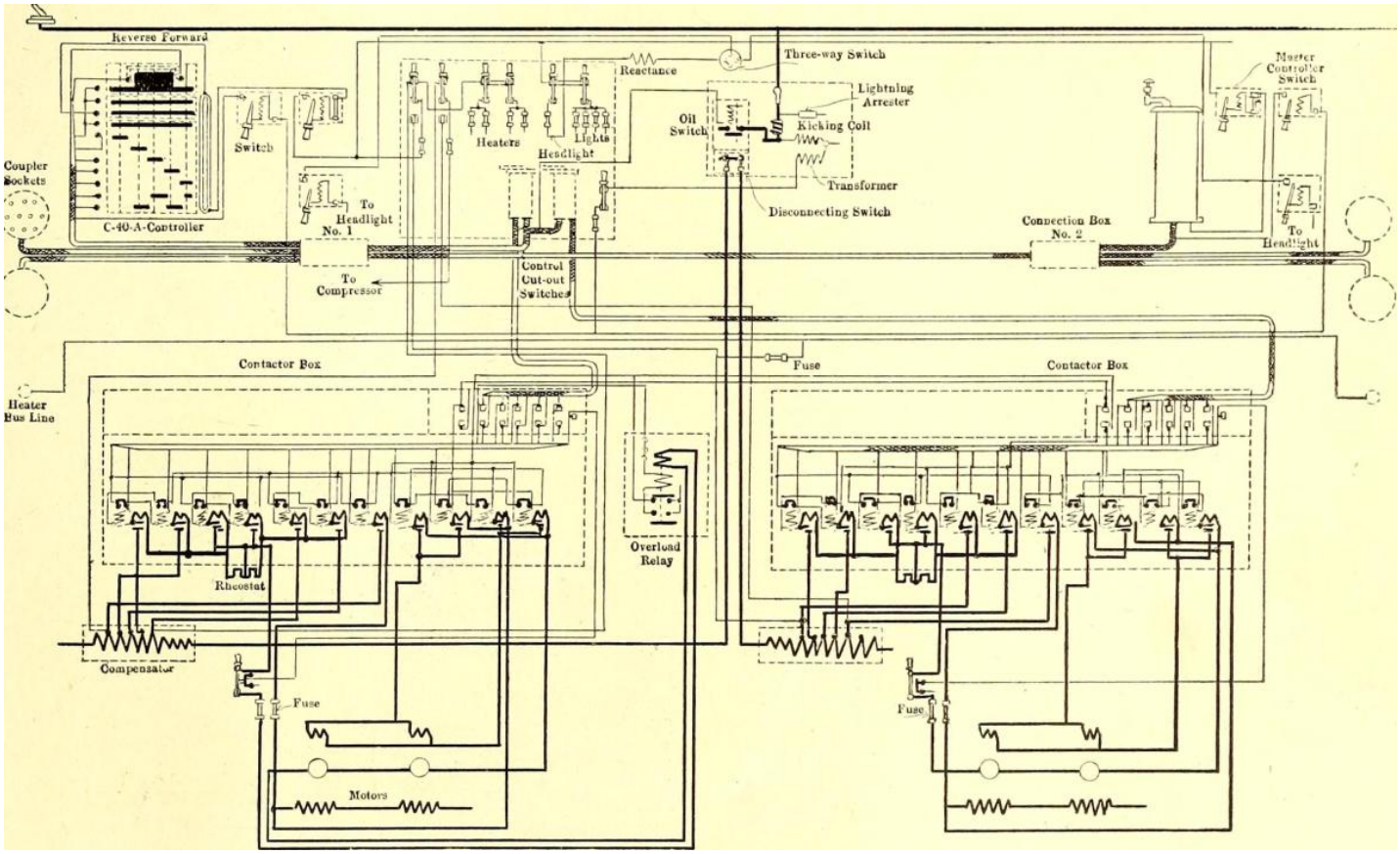
References:

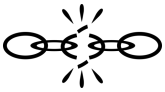
- Header image: Vachon, J., photographer. (1941) *Quitting time. National Electric Products Company, Ambridge, Pennsylvania. Man on right is company detective.* Pennsylvania, United States, Beaver County, Ambridge. 1941. Jan. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017811806/.
- Ap Dijksterhuis, "The Beautiful Powers of Unconscious Thought," APA Psychological Science Agenda, October, 2009.
- "Sleep, Learning, and Memory". Harvard Medical School (Division of Sleep Medicine).

Put It Down And Come Back To It Later… was originally published October 20, 2011.

Notes:

Follow The Chain





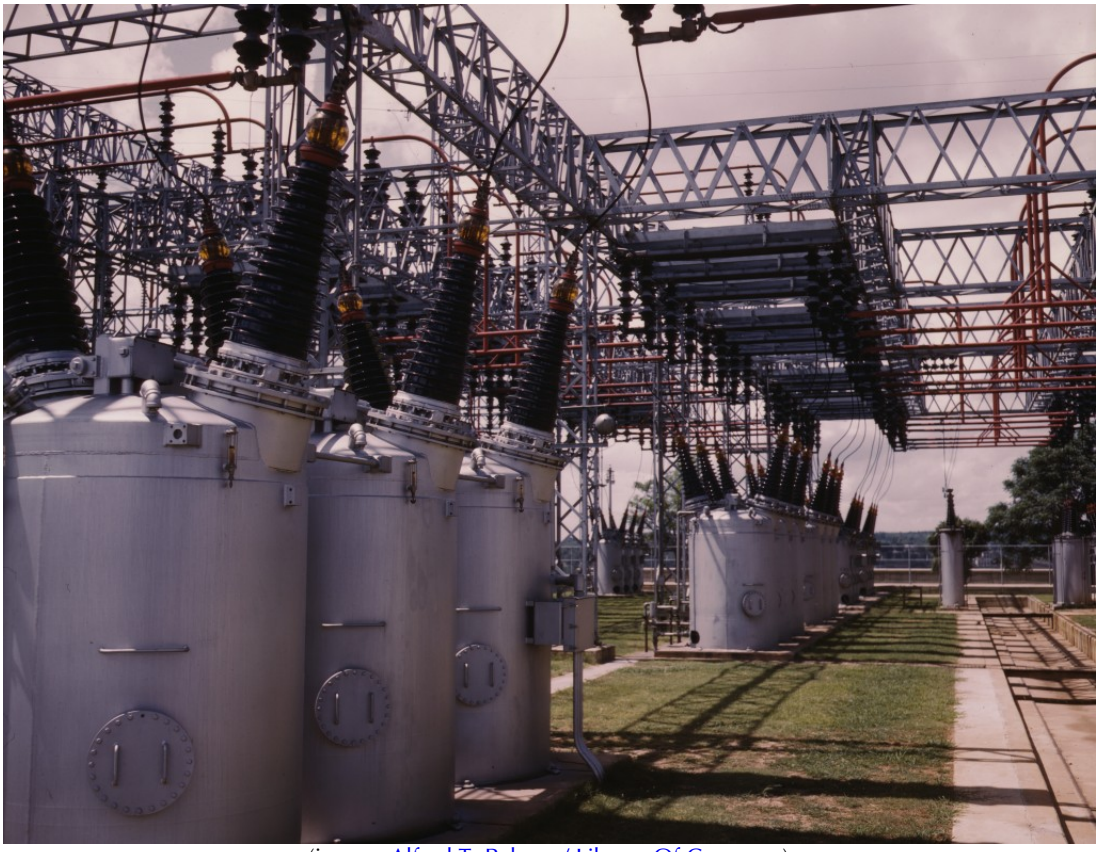
> Everything is about inputs and outputs. I just try to find the spot along the way where it's derailed.
>
> **Mike McCormick**

Many systems labor along a linear path and therefore lend themselves to a troubleshooting strategy I call: "follow the chain." These "chained systems" are everywhere—most machines have at least one component that falls in this category. That's because the essence of work, digital or analog, is transformative: you take an input, move it around, make additions or subtractions, and ultimately change it in a useful way. Because so many machines follow this A → B → C model, it's only natural that there is a corresponding troubleshooting strategy that mirrors this form.

To prime your mind for this topic, let's first look at some examples of chained systems:

(image: Steve Jurvetson / CC BY 2.0)

1) The Krispy Kreme doughnut production line: fryer → sugar glazer → cooling tunnel → my stomach.


(image: Alfred T. Palmer / Library Of Congress)

2) A pathway in the electrical grid: generating plant → substation → your home.

(image: johncarljohnson / CC BY 2.0)

3) The pipes and water heater that deliver hot water in your home: cold water supply → water heater → kitchen faucet.

Within transformational chains like these, the typical problem scenario is that either a **station** (place where work is done) or a **conduit** (pathway that moves material) will fail. Station failures may result in a "dumb passthrough" situation where material is still transmitted, albeit unchanged. For instance, if a water heater is malfunctioning, water may still get to the faucet, but it will be cold. Or, think about a network firewall that fails and stops filtering data, instead passing *all* Internet traffic along to your computer. A final tragic example: if that mesmerizing sugar glazer at your local Krispy Kreme runs out of glaze, the doughnuts *will* get to the end of the production line, though not with their addictive magic coating. Noooooooo!

Whatever the failure condition, you'll first want to examine the result at the **end** of the chain and work your way back from there. As mentioned above, the two typical scenarios will be:

- The output at the end of the chain will be flawed.
- There will be no output.

Since either a station or a conduit can be a point of failure, you should look for ways to isolate and test them independently. You can also measure the flow through the system by installing probes at key points along the transformational chain. Isolation and testing will tell you what needs to be replaced, while taking measurements will tell you what needs to be isolated and tested. So, the "follow the chain" strategy boils down to these two tactics:

1. Isolate and test each component of the chain.
2. Measure the flow through the system by installing probes.

Point Break-down

Most audio/video setups are chain-like systems and provide a great illustration of how to test both stations and conduits. The main parts usually include source components (DVD players, cable TV decoders, video game systems, etc.) and intermediate devices (switchers/selectors, pre-amplifiers, amplifiers, etc.), whose signal eventually winds up at an output device (TV, speakers, etc.). In our model, these are the **stations** in the chain, because they transform the audio or video signal in some way. On the way to your eyes and ears, the way these signals move is via wires (speaker wire, RCA/coaxial/HDMI cables, etc.) or radio waves. These are the **conduits** in our model.

Let's look at the "follow the chain" strategy in action. In this example, we'll examine a typical home video setup with 2 source components. But first, let's set the scene for our crisis: we're unable to watch our favorite movie *Point Break,* starring the incomparable Keanu Reeves. Using the principles of measurement and isolation, we'll take advantage of standardized connectors and cables to find the weak link somewhere in this chain: *Point Break Pure Adrenaline Edition* DVD → DVD Player → TV.

We first recognize that the DVD disc is itself part of the chain. Therefore, we take the *Point Break Pure Adrenaline Edition* DVD over to a friend's house to see if it will work over there. Four hours later, we can confirm that it does. "Four hours later…?", you ask. Listen, once you get rolling with a Keanu Reeves flick, you have no choice but to watch the movie the whole way through. Plus, there's a ton of special features included in the *Pure Adrenaline Edition*, including 8 deleted scenes (!) and a featurette titled "It's Make Or Break."

Here's what our setup consists of:

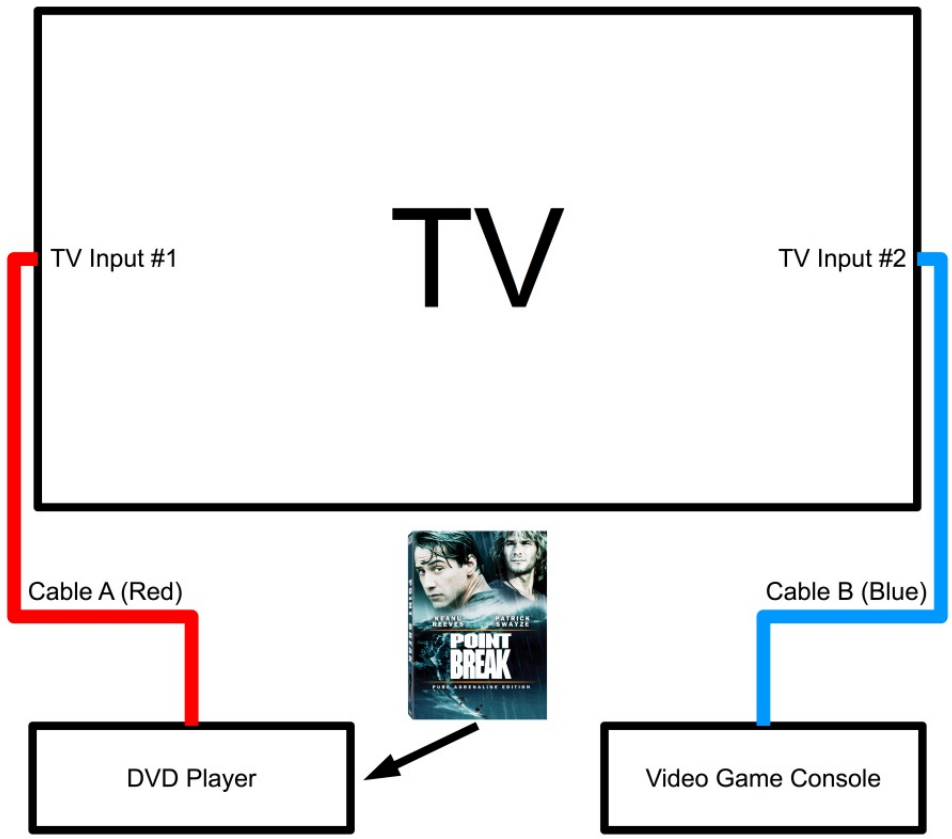


Diagram: a TV with two inputs, two source components, and perhaps the greatest movie of all time. A crisis builds as we realize that we can't view this masterpiece. We'll have to troubleshoot…
(image: © Jason Maxham)

In this system, the cable that connects the DVD player to the TV (Cable A, in red) is the same type as the cable used to connect the video game console to the TV (Cable B, in blue). Likewise, the two inputs on the TV are identical and will both accept this same type of cable. This is a crucial realization, because it will allow us to swap inputs and cables later on. Before we get any further, let's make a list of all our components and what we know about them:

Part	**Status**
Point Break Pure Adrenaline Edition DVD	OK
TV Display	?
TV Input #1	?
TV Input #2	?
Cable A	?

Cable B	?
DVD Player	?
Video Game Console	?

Starting out, you can see there are a lot of question marks in our list. Besides the disc, we haven't tested anything yet, so every part is suspect. By the way, this is a good way to begin troubleshooting any chained system: by not assuming anything!

Now that we understand the setup, we start at the end of the chain (the TV in this case). We power up the TV and the DVD player, select TV Input #1, insert the *Point Break Adrenaline Edition* DVD, press play, and…nothing. Having already eliminated the DVD disc as a suspect, the remaining candidates for the problem are: the TV display, TV Input #1, Cable A, or the DVD player. The problem must lie somewhere along this path! As I just mentioned, because the connectors, cables and TV inputs are interoperable, we can use the video game console and Cable B to test various theories about the source of the failure.

First, let's quickly verify the that video game console is working (that's the pathway of: Video Game Console → Cable B → TV Input #2 → TV). We want to be swapping around parts that have been *verified* to function, so it's imperative we determine their status at the outset. We turn on the video game console, select TV Input #2, and everything works just fine. Great, now we've got a line of known working components that we can use to find the failed component in the DVD player chain. We'll also update our list of working parts to chart our progress as we narrow down the list of suspects:

Part	**Status**
Point Break Pure Adrenaline Edition DVD	OK
TV Display	OK
TV Input #1	?
TV Input #2	OK
Cable A	?
Cable B	OK
DVD Player	?
Video Game Console	OK

Ultimately, we want to work through our entire list of question marks. All things being equal, it doesn't matter where you start. Of course, things usually aren't equal and so a good place to begin is with any "low-hanging fruit": the easiest parts to test. It's trivial to take Cable A from Input #1 and connect it to Input #2, so we'll start there:

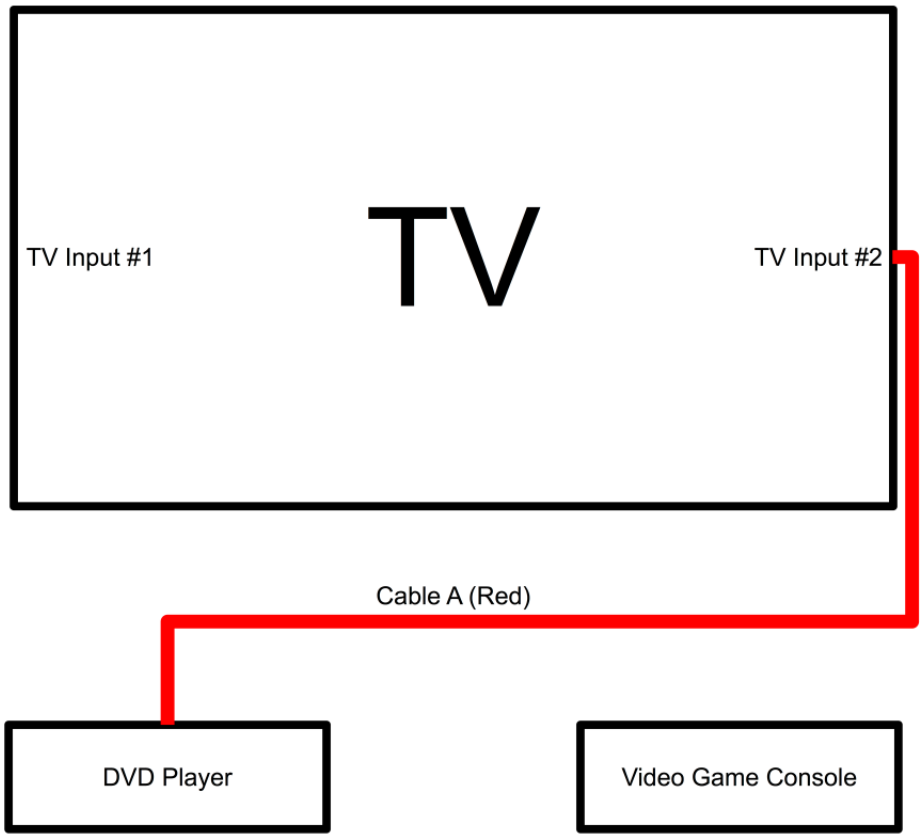


Diagram: testing the DVD player using Cable A and TV Input #2.
(image: © Jason Maxham)

When we hook up the DVD player using Cable A and Input #2, it still doesn't work. You may think this is a setback, but this is good information. Remember from our earlier test that the TV display and Input #2 are known to work. That means we have *isolated* the problem to either the DVD player or Cable A: the problem must be in one of these two components.

Another consequence of this test is to de-prioritize the testing of Input #1. Since we haven't tested it, we can't say for sure that it works and so it must remain a "?" on our list. However, remember the statistics involving multiple failure scenarios. We know for sure there is a problem somewhere within the combination of the DVD player and Cable A. It's possible, but highly unlikely that Input #1 is also failing at the same time.

Let's specifically test the hypothesis that the DVD player is malfunctioning. To do this, we'll hook up the DVD player using known working parts: Cable B and Input #2. The only question mark in this particular path is the DVD player:

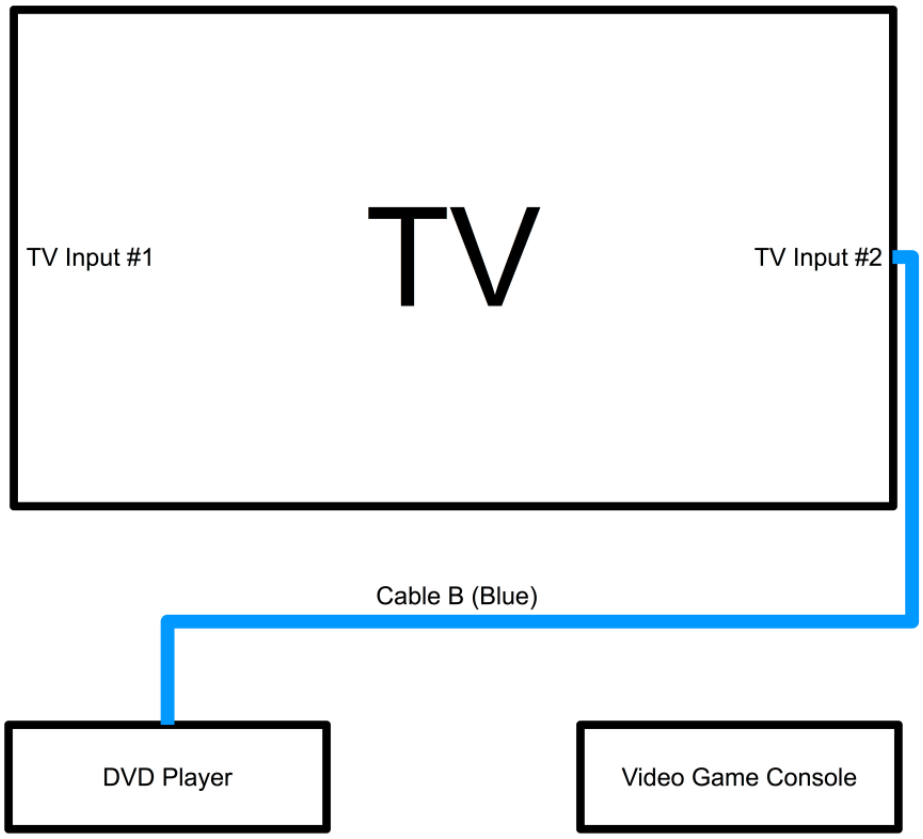


Diagram: testing the DVD player *using* Cable B and TV Input #2.
(image: © Jason Maxham)

Keanu flickers to life! Because this configuration works, we can update our table and mark one more component as "working": the DVD player.

Part	Status
Point Break Pure Adrenaline Edition DVD	OK
TV Display	OK
TV Input #1	?
TV Input #2	OK
Cable A	?
Cable B	OK
DVD Player	OK
Video Game Console	OK

Now we're down to just two suspects: Input #1 and Cable A. Since it means switching just one cable, let's test Input #1 like this:

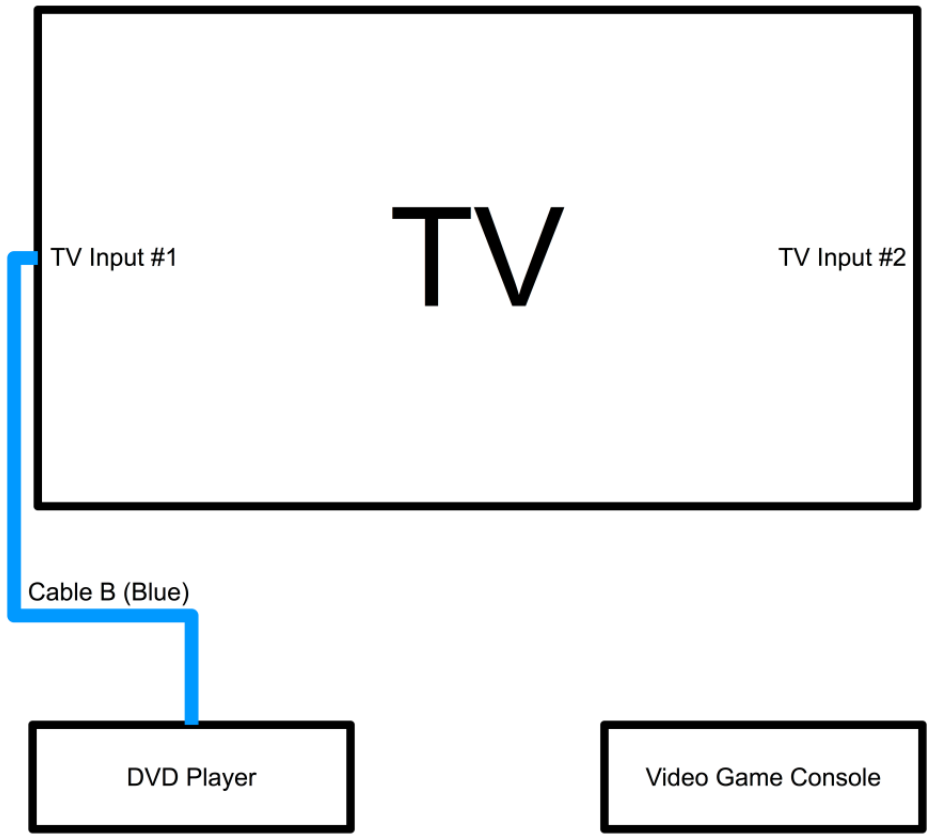


Diagram: testing Input #1 using the DVD player and Cable B.
(image: © Jason Maxham)

This configuration works too, and so we now know that Input #1 is okay (this was our hunch, but it's good to know definitively). Let's update our table one last time:

Part	**Status**
Point Break Pure Adrenaline Edition DVD	OK
TV Display	OK
TV Input #1	OK
TV Input #2	OK
Cable A	?
Cable B	OK
DVD Player	OK
Video Game Console	OK

We've done it, there's no longer a mystery as to what's preventing us from having an awesome Saturday night with our favorite movie. All the components in our system have been verified to work, with the exception of one. The culprit must be: **Cable A**.

Measurement And Probes

The other strategy for "follow the chain" troubleshooting is to put probes at key places in the transformational chain. Let's imagine a hot water system in an apartment building. Water comes from the cold water supply, enters the water heater, is heated and then flows to the apartments. Pretty pedestrian stuff:

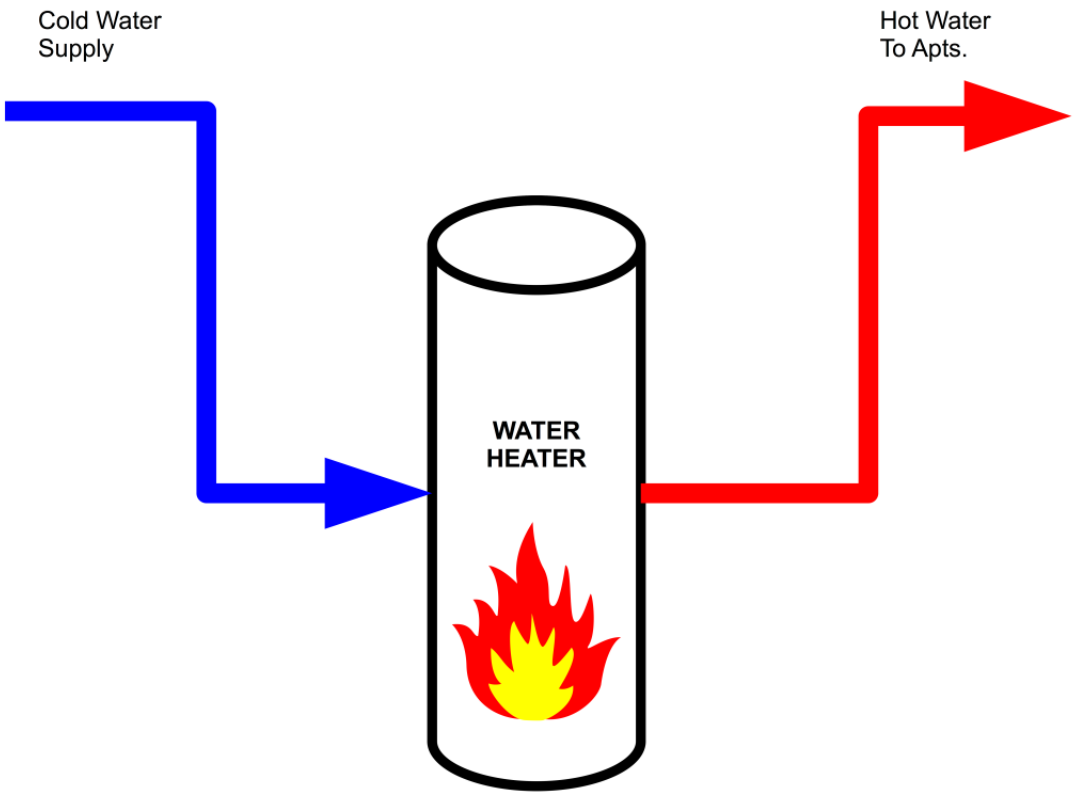


Diagram: water heater system.
(image: © Jason Maxham)

There are many ways this system can fail, but let's examine two common scenarios:

1. The heater malfunctions, and the cold water is simply passed through unchanged (a "dumb passthrough" scenario).
2. A pipe bursts or leaks, preventing water from getting to an endpoint like a shower or kitchen sink (the "no output" situation).

Putting pressure and temperature monitors in place, we can better understand what is happening as water flows through the system:

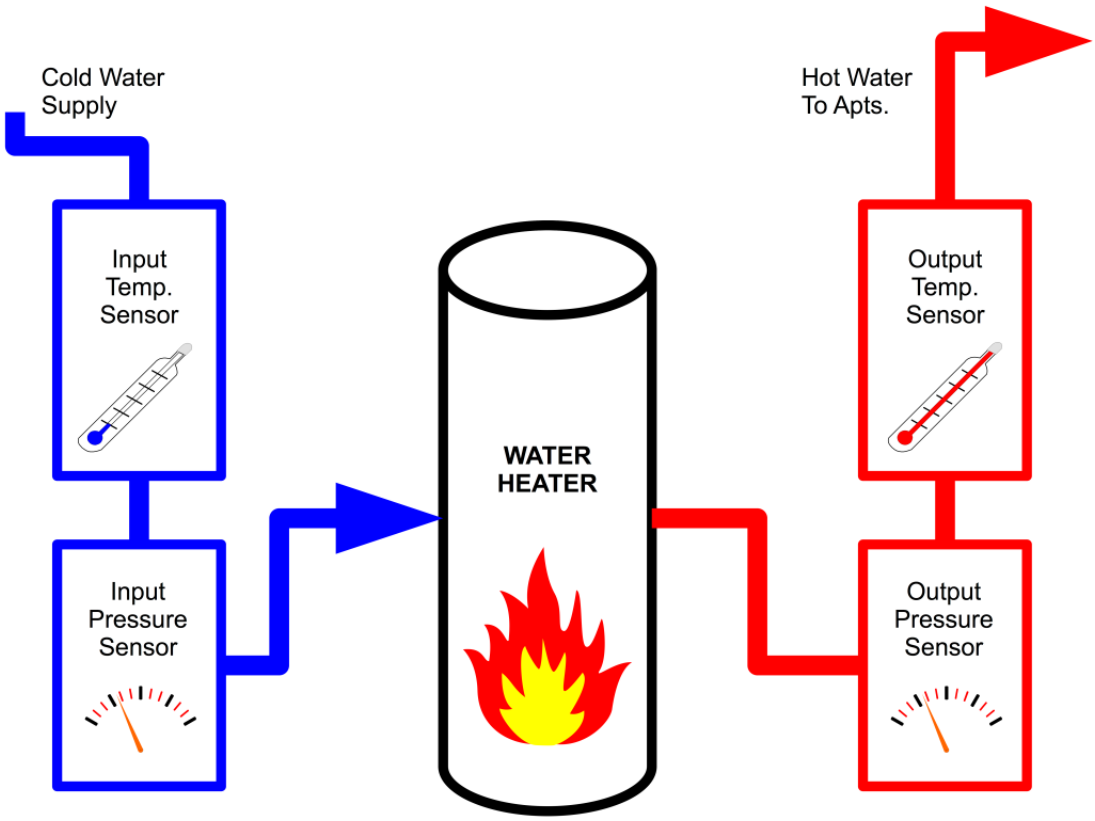


Diagram: water heater system with temperature and pressure sensors.
(image: © Jason Maxham)

Now, we have insight into a whole variety of failure scenarios and can make very quick inferences as to their cause. Let's look at a few examples of some possible readings from our gauges and the likely explanations:

Scenario	Input Temp. Sensor	Output Temp. Sensor	Input Pressure Sensor	Output Pressure Sensor	Explanation / Troubleshooting Ideas
#1	60° F	60° F	50 psi	50 psi	The water heater is just passing the water through without doing anything: the input temperature equals the output temperature. Is the heater turned on? What's the thermostat set at?
#2	68° F	68° F	0 psi	0 psi	There's no water pressure and the temperature is equal to room temperature. Has the water supply been shut off? Did you forget to pay your water bill?
#3	60° F	60° F	50 psi	30 psi	The loss of pressure means there's a broken pipe or the tank is leaking. There may be flooding! Our gauges allow us pinpoint a leak somewhere between Pressure Sensor #1 and Pressure Sensor #2. That narrows things down considerably.

#4	60° F	120° F	50 psi	50 psi	Normal operation. The input pressure is equal to the output pressure. The water temperature goes from cold (60° F) on the input to hot (120° F) on the output. This is what a water heater is supposed to do!

Do you see how adding these 4 gauges has given us a much better understanding of how this system is functioning? They can show us exactly where and what to look for when investigating a problem. Four critical parts of our water flow chain are now being monitored and we'll reap the benefits when we need to troubleshoot.

Any weak links?
(image: Taylor S / CC BY-ND 2.0)

Use Your Eyes

Sensors and gauges are great, but there are many cases where the process chain can be *visually* inspected (remember the lessons of "Listen Up": the importance of being engaged with the world around you and tuned in to your senses). Back to the Krispy Kreme production line: you wouldn't need a "glazing sensor" to tell you that the donuts aren't being glazed. You can see and taste it.

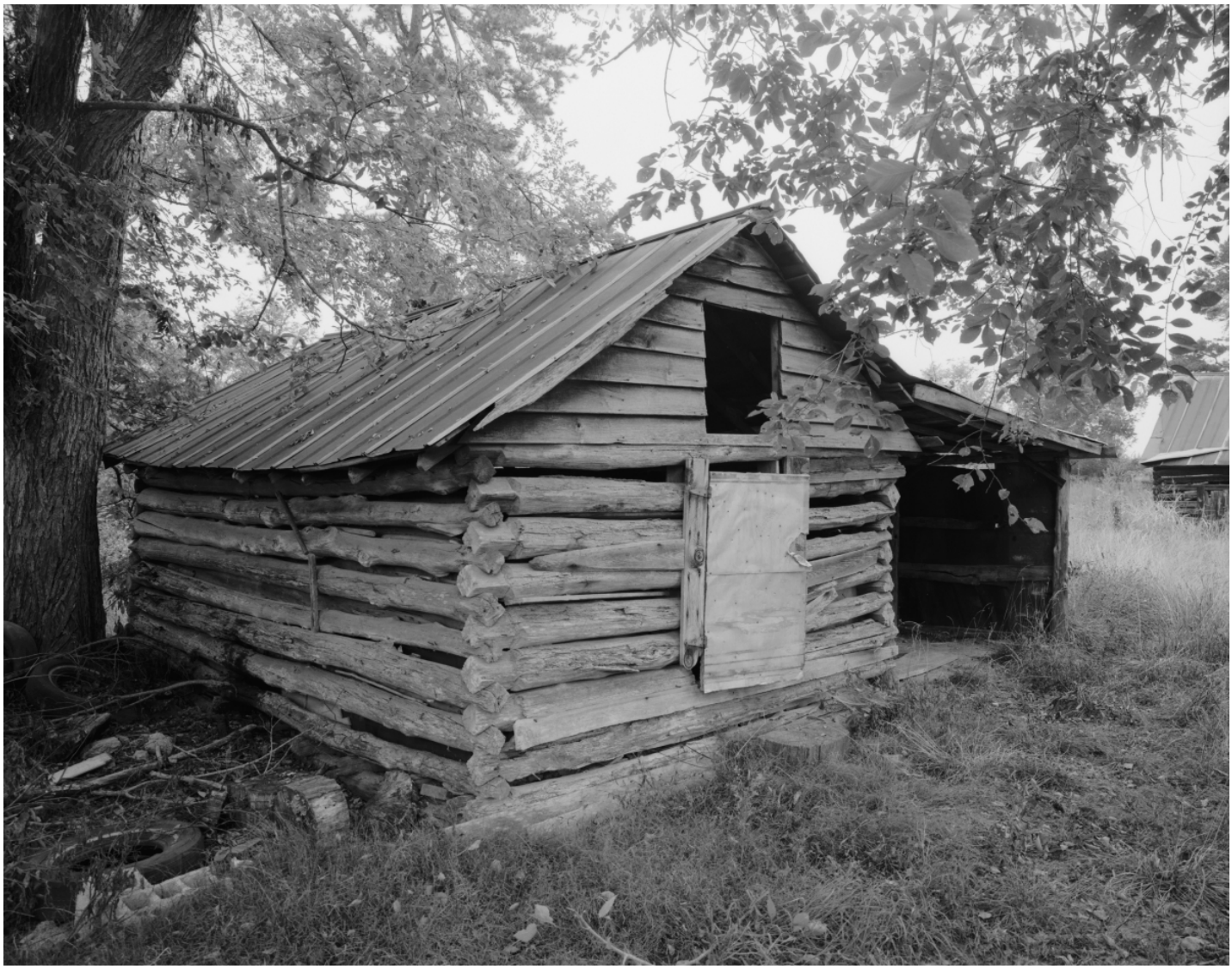
References:

- Header image: *New Canaan Electrification – Wiring Diagram of Electrical Apparatus.* Electrical Railway Journal. Volume 33, Number 20. May 15, 1909. Pg. 901. Retrieved from the Internet Archive.

Follow The Chain was originally published November 9, 2011.

Notes:

Bare Bones: Back To The Basics



> You don't know what's useless without first thinking about it.
>
> **Alex Chaffee**

I used to love assembling my own computers. I say "used to" because somewhere between my first and the 200th, the love has grown cold. It took a long time, but the novelty eventually wore out. Now, I'm definitely a "have it show up in a box and just plug it in" kinda guy. Also, the difference between a fun hobby and the "grind it out" nature of doing it for work was partly to blame. We were über-thrifty at Discovery Mining and I had an insatiable thirst for computer hardware at the time, so we built our first few computer clusters by hand. However, it wasn't long before we had outsourced this function: we simply couldn't have scaled fast enough if we had continued to build our own computers. They never said it out loud, but I also think my team was grateful for the change: for our first couple of builds, even the CFO was installing CPUs and RAM in our make-shift assembly line!

For the sake of this article, I will mentally revisit those bygone days of hardware bliss and try to convey the geek-tastic pleasure I would get from poring over lists of parts. I loved researching and choosing the components: the

motherboard, the RAM, the hard drive, etc. In general, I'm really into performance statistics, whether it's frames per second, horsepower, watts per channel, or sprinkles per square inch (donuts). After my orgy of product research was over, I'd finally order the parts and then anxiously….wait…for them to arrive. Since this was a "race to the bottom" where price was concerned, I used the sketchiest of Internet vendors, the kind of places that were probably selling your credit card number to the Mafia. Therefore, at least one component would take weeks to arrive and hold up the assembly. When all the parts would finally show up, I was itching to build the system.

When that last part arrived, I'd quickly assemble the computer—for all the waiting that preceded, it never took long. Then came the most exciting part: booting up the system for the first time. However, I can't tell you how many times my expectations were deflated like a pricked balloon when I pressed the power button and…nothing happened. This occurred a lot, until I became one with the Universe and discovered a troubleshooting strategy I call "Bare Bones" (I've also heard it called "back to basics"). The idea is simple and effective: you can often remove or disable unnecessary components in order to get a machine to work.

The owner of this car clearly understands the meaning of "bare bones." Doors and side panels? Optional.
(image: Mobilus In Mobili / CC BY 2.0)

Let's say you have assembled the following components for your sweet new gaming rig:

- Power supply
- Motherboard
- CPU
- RAM
- Video card
- Sound card
- RAID card (controls the hard drives)
- Hard drives
- Network card
- Keyboard
- Mouse
- Monitor
- DVD drive

You put it all together and press the power button. It doesn't work, and you experience the sinking feeling of "did I just waste all this money assembling a large pile of useless computer parts?"

Don't fret: just remember that not ALL of these things are required for the computer to work in its most *primitive* state. In fact, less than half of these components are needed to get the computer to boot. We can easily cut the above list down by over half to a list of the truly required components:

- Power supply
- Motherboard
- CPU
- RAM
- Video card
- Monitor

Usually, when I would unplug or remove all the extraneous components, leaving just the essentials, IT WOULD BOOT!

Even more subtly, you usually don't need *all* of a particular type of required component. For instance, you might have bought 4 sticks of RAM, but the computer will boot with one. Remember, for the "bare bones" strategy, you want the **absolute minimum configuration** which will work. If that means installing just one hard drive (as opposed to the 4 you eventually want to eventually use), just use one.

Now that you've got it to boot, you can go ahead and start adding in the rest of the components, one-by-one, to see which of them (and, there may be more than one) are causing the malfunction.

Bare Bones Everywhere

Bare Bones is a universal principle that applies to any machine that has subsystems or extra components that aren't needed for basic operation. In our advanced technological world, there's so many examples of things that have amazing "bells and whistles" that, while nice to have, simply aren't needed for basic operation. In the context of the automotive industry, the number of features that car companies deliver gets longer with each new model year. Think about a modern car with GPS navigation, power windows, heated seats, a 10-speaker surround sound system, etc. The list of truly Bare Bones equipment (steering wheel, engine, accelerator, brake, 4 wheels, etc. – see photo above) has come to represent a smaller and smaller percentage of the components in your average car. Yet the strategy of reverting to these basic functions remains available to you, should you need to turn to them for troubleshooting purposes.

Of course, "basic operation" depends on how you're using the system. Let's say you have a refrigerator truck that is inoperable and you can get it to run by going Bare Bones and disabling the refrigeration unit. If you're hauling meat around, clearly the refrigeration unit is a "must have" and therefore the truck won't be eligible for use in that capacity. That's okay, because Bare Bones can be used as a **triage technique**: if the truck is broken down in the middle of rush-hour traffic, clearly your main goal is simply driving it to the repair shop. In that case, disabling the refrigeration unit in return for "basic operation" would be a big win!

Flaming tomatoes: tasty and good to remember if you're a pilot.
(image: Jem Stone / CC BY 2.0)

Bare Bones In Aviation

As a pilot, I've encountered the principle of Bare Bones in aviation, codified in manufacturer manuals and FAA regulations. As strange and dangerous as it may sound, not everything on an airplane is required to work before you go flying! (Although, keep in mind that "minimum requirements" and "good idea" are not necessarily the same thing.) Studying to take the pilot exam, I learned the mnemonic "TOMATO FLAMES" to remember what must be **minimally working** in order to legally fly in clear conditions (VFR) during the daytime:

Tachometer
Oil pressure gauge
Manifold pressure gauge
Altimeter
Temperature gauge (liquid-cooled engines)
Oil temperature gauge (air cooled engines)
Fuel gauge
Landing gear position indicator
Airspeed indicator
Magnetic compass
ELT (emergency locator transmitter)
Seat belts

Mistakes in aviation are costly, so the list of what can be *broken* is also strictly proscribed in the confusingly named "Minimum Equipment List" (MEL), specific to each aircraft. Here's an example from the Cessna 182's similarly-used "Kinds of Operations Equipment List," which describes what must be working if you want to fly in various conditions:

KINDS OF OPERATIONS EQUIPMENT LIST (Continued)

System, Instrument, Equipment and/or Function	KIND OF OPERATION: VFR DAY	VFR NIGHT	IFR DAY	IFR NIGHT	COMMENTS
NAVIGATION AND PITOT-STATIC SYSTEM					
1 - G1000 Airspeed Indicator	1	1	1	1	
2 - Standby Airspeed Indicator	0	0	1	1	
3 - G1000 Altimeter	1	1	1	1	
4 - Standby Altimeter	0	0	1	1	
5 - G1000 Vertical Speed Indicator	0	0	0	0	
6 - G1000 Attitude Indicator	0	0	1	1	
7 - Standby Attitude Indicator	0	0	1	1	
8 - G1000 Directional Indicator (HSI)	0	0	1	1	
9 - G1000 Turn Coordinator	0	0	1	1	
10 - Non-stabilized Magnetic Compass	1	1	1	1	
11 - VHF Navigation Radio (VOR/LOC/GS)	0	0	A/R	A/R	As Required Per Procedure.
12 - GPS Receiver/Navigator	0	0	A/R	A/R	As Required Per Procedure.
13 - Marker Beacon Receiver	0	0	A/R	A/R	As Required Per Procedure.
14 - Blind Altitude Encoder	A/R	A/R	1	1	As Required Per Regulations.
15 - Clock	0	0	1	1	
16 - GFC 700 AFCS	0	0	0	0	

(Continued Next Page)

Excerpt: "Kinds of *Operations* Equipment List" for the Cessna 182.
(image: Cessna 182 Pilot Operating Handbook)

In this excerpt, you can see that the main airspeed indicator is always required (noted by a "1" in each of the columns to the left of the entry: "G1000 Airspeed Indicator"), but the standby airspeed indicator only needs to be working if you intend to fly through clouds or in low visibility situations (IFR).

While the "TOMATO FLAMES" and MEL lists aren't designed to be used for troubleshooting while flying (they're supposed to be consulted *before* taking off), I've read accounts of pilots going Bare Bones in emergencies by shutting off unnecessary systems to conserve precious battery life or quell fires (e.g., turning off a flaming engine on a multi-engine aircraft). Whether on the ground or in the sky, the principle is the same: not every subsystem is required for a machine to function. Conversely, the corollary is also true: a malfunctioning subsystem may be *preventing* a machine from functioning (and you may need to shut it down or remove it to restore system operation as a whole).

Bare Bones In The Digital Domain

I find that the Bare Bones strategy has its fullest expression in the digital world. That's because stripping an electronic device down to its bare essentials is usually just a matter of a few keystrokes. You can go from a very complicated configuration like this:

```
pager lines 24
mtu inside 1500
mtu outside 1500
no failover
no asdm history enable
arp timeout 14400
nat (inside) 0 access-list inside_nat0_outbound
access-group 101 in interface outside
```

to Bare Bones by simply adding something to the front of every line (sample commands taken from a Cisco network router configuration). In the parlance of programmers, this is called "commenting out" a line, an act that renders the command inoperable:

```
!--- pager lines 24
!--- mtu inside 1500
!--- mtu outside 1500
!--- no failover
!--- no asdm history enable
!--- arp timeout 14400
!--- nat (inside) 0 access-list inside_nat0_outbound
!--- access-group 101 in interface outside
```

Removing items from a digital device's configuration is typically very easy. Whether it's putting a "!" "#" or "/*" at the beginning of every line, or a clicking a checkbox on a web-based form, the principle is the same. Being able to turn on and off functionality with just a few keystrokes is a huge advantage over mechanical systems, where disabling functionality usually requires a lot more work (and the turning of wrenches and screwdrivers).

A very fruitful strategy is to bring a device back to its simplest state and then add back in your configuration options one by one:

```
pager lines 24
!--- mtu inside 1500
!--- mtu outside 1500
!--- no failover
!--- no asdm history enable
!--- arp timeout 14400
!--- nat (inside) 0 access-list inside_nat0_outbound
!--- access-group 101 in interface outside
```

until the offending option that is preventing the machine from working has been identified.

Conclusion

I think the importance of Bare Bones as a troubleshooting technique will only continue to grow. The prevalence of "feature bloat" in mass-produced products seems to only increase as time marches on. Tight QA budgets don't often allow for every feature permutation to be covered in a product's testing phase. Remembering that a machine has both critical and extraneous equipment allows you to bypass crippling interplay between the two by stripping a machine down to its bare essentials.

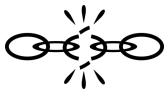
References:

- Header image: Historic American Buildings Survey, C. (1933) *Edgewood Farm, Log Shed with Lean-To.* Clover, Halifax County, Virginia, 1933. Documentation Compiled After. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/va1892/.

Bare Bones: Back To The Basics was originally published November 15, 2011.

Notes:

Does It Need To Be Fixed?



> In a lot of the problems I run into, it's "I can't fix this." So, I log the problem and toss it...and then I need to work around it.
>
> **Karl Kuehn**

When you're helping someone fix a broken machine and it seems like a resolution won't be forthcoming soon, one of the best questions you can ask is: "What are you trying to accomplish?" They may look at you funny and say "Duh! I want to fix this stupid thing!" Of course they do, but *why*? What means to an end is this machine providing? Identifying and focusing on the desired outcome will make finding workarounds easier.

Given that it's much better to problem-solve in a low stress environment, where you have ample time and no one's breathing down your neck, the troubleshooter has a tremendous incentive to provide alternatives. The strategies and virtues presented in *The Art Of Troubleshooting* will help you be that superstar under pressure, but as exciting as that may be, I think you will prefer the low-pressure option for your day-to-day routine.

Always frame alternatives with the sentiment of "I'm on your side, I'm committed to helping you solve your problem, and I'm trying to get you back in the game as quickly as possible." If you're required to help (i.e., it's in your job

description), people may think you're trying to avoid doing your job by offering a workaround. Showing you have their best interests in mind, along with a candid explanation of the situation, is typically the best combination: "Listen, I'm going to be totally honest with you, I don't know what's wrong and it could take a whole day to figure it out…" Most people will gladly accept a workaround if they know it's their only option to keep moving forward.

Original Problem	Desired Outcome	Possible Workaround
"The printer in my cubicle is broken…"	"I want to print my presentation for tomorrow's meeting."	"Can you use the printer in the conference room instead?"
"My car won't start…"	"I need to get to work."	"Can you take a cab, ride the bus or get a lift from a co-worker?"
"The magical Interwebs are broken…"	"I need to send this email."	"Can you go across the street to the cafe and use their Internet connection to send your email?"
"The lights won't turn on…"	"I need to see to be able to do my job."	"Can you temporarily move to a place where there's more light?"

Must We?

Asking "Do we have to fix this?" isn't evasion, it's a reminder that the way things *were* is not the same as the way things *should* or *could* be. The underlying premise of troubleshooting is that we want to get back to where we started from, to the way the world was before a breakdown occurred. But, just as in life, going back to the way things were isn't always a good idea. After the storm had passed, I've often been grateful that a breakdown occurred. I may not have appreciated how the introspection was thrust upon me, but failures were invitations to reexamine our systems and procedures.

Thankfully, repair is optional.
(image: Library of Congress)

As a troubleshooter, you're usually focused on the problem to the exclusion of all else. It may not enter your mind that your time might be better spent NOT fixing it. Woah…*not* fix it?! That's not a nod to laziness, but simply an invitation to consider devoting your scarce resources and energy to something better. Let's examine the scenarios where repair is suboptimal:

A Workaround Is Better

While investigating failures, I often see things that…just don't compute. "Why did they do it this way?" is the question associated with these moments. Candidly, I'll admit that it's frequently "Why did *I* do it that way?" A machine is installed to do a particular job, its place and specifications fixed, but the world keeps on changing. Later on, when it finally breaks down, what it is was doing or how it was doing it might not make sense anymore.

For this reason, you'll often discover redundancies and unnecessary steps while troubleshooting. These will be opportunities to bypass the broken part or machine entirely! "Why is our Internet traffic flowing through this broken switch and then through the router? If I patch it directly, I can restore service and eliminate a single point of failure."

But be careful, my legion of White Knights who ride in to save the day! Trying to be clever in the heat of the moment is like:

> …trying to take the master carpenter's place. When you handle the master carpenter's tools, chances are that you'll cut your hand.
>
> **Tao Te Ching (verse 74)** [1]

Of course, this doesn't apply if you just happen to be a master carpenter.

When I first discovered that I could simply decline fixing a downed system because I had found a better workaround, I thought I was pretty cool. But, reality always has the last word. The system you're working on was designed, installed, or assembled in a certain way, usually for a good reason: can you truly say you've been privy to all of the discussions, design meetings, prior incidents, etc. which resulted in the system you're now trying to modify on-the-fly?

That's why, if the stakes are high, this option is only for the 10th-degree-blackbelt-who-Chuck-Norris-calls-for-help kinda superhero. Maybe you do know this system like the back of your hand. Maybe you've been at every meeting, been involved in every design decision and understand why the resulting system looks like it does. If you're that sure, go ahead and pull the trigger, Rambo.

In a low-pressure situation, be open to improvisation. Again, you may discover a workaround that is superior to the original and doesn't require fixing the broken system!

You Were Planning To Upgrade Anyway

If you've been waiting for an excuse to upgrade, there's no better time than after a failure. Often, the broken system may not be able to be repaired, at least not in a reasonable amount of time. You needed downtime to upgrade? Again, you've got it on a silver platter. Seize the opportunities that life presents! Just make sure you employ sound procedures to ensure the upgrade goes successfully, especially if it's on the spur of the moment. A data collection program, regression tests, or checklists are all great ways to verify everything went smoothly.

Swap It

You don't have to impress me with your troubleshooting skills. Grab the spare (you do have spares, right?), replace the failed system, and move on. Depending on the severity of the problem, you may not be able to troubleshoot your way out. If you want to find the root of the problem, give yourself the luxury of doing it later, when the pressure's off. Or, send it back to the manufacturer and let them figure out what went wrong.

Always Have A "Plan B"

If you're going to pursue any of these "don't fix it" strategies, I highly recommend doing it as a parallel path to your troubleshooting efforts. Form a second team to investigate and implement your workaround. There's nothing like a little healthy competition and it respects the maxim: "Always have a backup plan."

References

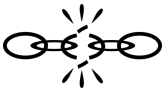
- Header image: "Papahānaumokuākea Marine National Monument where the Hoei Maru shipwreck lies in the shallow waters of Kure Atoll." NOAA's Office of National Marine Sanctuaries.

- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 74.

Does It Need To Be Fixed? was originally published November 22, 2011.

Notes:

The Phone Is Ringing, So Answer It



> Read the error.
>
> **Alex Chaffee**

How many times have I seen even seasoned technicians dismiss, ignore, or forget to seek out relevant error messages? A lot! So many machines in use today will tell you *exactly what is wrong* (should you be bothered to listen). It doesn't take a rocket scientist to tune in to this information, but you may have to snap out of the trance you're in to recognize what you're missing.

An example from my work life: I was with a co-worker and we were trying to look at some data on a portable storage device. After plugging it in, it was clear it wasn't working because the drive wasn't showing up in the filesystem browser. A few error messages popped up on the screen, which my friend, on auto-pilot, quickly clicked to dismiss. Like a pop-up box removal ninja, he killed the messages so fast that I could barely see them:

Me:"What did that say?"
Co-worker: "I dunno, it was in the way…"

Another error message popped up and I grabbed the mouse, insisting that we read it. The alert said that one of the hard drives in the array was missing. Hmm…that's interesting. We opened up the enclosure and, sure enough, found a disk that wasn't seated properly. We reseated the drive and everything returned to normal. **The device was screaming out what the problem was, but we weren't listening!**

You should answer that.
(image: Shannon Moore / CC BY 2.0)

The lesson is simple: unless you have a good reason not to, a good place to start any troubleshooting exercise is with a machine's own error codes. If the system has built-in diagnostics that will tell you exactly what is wrong, why aren't you using them? Remember, the Master Troubleshooter is looking to find the shortest path to a resolution. Not using the low-hanging fruit of built-in diagnostics is the equivalent of walking from LA to New York, after turning down a ride on a friend's rocket ship. Leave the hard path of self-denial to your personal quest for Enlightenment. When it comes to troubleshooting, take the easy way out and save your energy for the really tough problems.

In essence, this is just another reminder of the importance of being present when you are troubleshooting. I've been over the importance of listening to people, so I'll simply add: listen to your machines too!

References:

- Header image: *Opening of New Phone Line, 1/17/27*. 1927. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016842708/.

The Phone Is Ringing, So Answer It was originally published December 7, 2011.

Notes:

Duplicate The Problem

There was a sunroof on a brand-new vehicle. It wouldn't work and it was in the shop a number of times. It made no rhyme or reason and was completely intermittent. They sent out a factory rep and he said do this, this, and this. He told me to check the resistance to the grounds. We tightened things up and got the resistance down and he was sure we had fixed it. He leaves and says, "I'll write it up as fixed cause I know that's going to take care of it." I was very skeptical: the test, in my opinion, was not valid.

The car had been torn apart and I started putting it back together and the sill plate didn't fit right. I pulled it up and saw this little shiny copper spot underneath one of the clips that held the sill plate on. One of the wires that controlled the sunroof ran underneath the sill plate and when you'd pop the sill plate back on, the clip would hit the wire and short it out. So, it could be okay until someone stepped in the back of the vehicle or hit a bump on the road or whatever! I shorted the wire just to make sure and it stopped the sunroof from working. It was just sheer luck that I found it, because this one piece didn't fit the way it should.

Dan McCormick

The gold standard for troubleshooting is duplication of the problem. Simply put, duplication is the ability to reliably recreate a particular failure condition. The word "reliable" is key: it should be something you can summon forth on demand every time. Duplication is the beginning of a simple but powerful strategy that includes changing variables

one at a time until you find the offending part, subsystem, or configuration setting. Reliable duplication gives purchase to your attempts at isolating the cause, allowing you to verify after each change whether or not you've fixed the problem. Reliable failures (seems like an oxymoron, eh?) make certain that, when you make a change and it begins to work, you've found the cause!

Please note, the machine must be *nearly working* for duplication to be useful. To illustrate why, imagine a car that has sat unused for 50 years. When you go to start it, nothing happens. You try again, with the same result. Feeling good, you give it a try a third time and a smile creeps across your face. "Aha!", you think, "I've just duplicated the problem!" Yes, you have, but I'm afraid that it won't help you very much in the context of a car that hasn't run in 50 years. That's because the number of things likely to be wrong is so large that a duplication plus isolation strategy will only tell you that…nearly every single component needs to be repaired or replaced. A project like that is much closer to manufacturing (making something work for the first time) than troubleshooting.

Contrast the above scenario with a car that stopped working 5 minutes ago. And, its state of not working is 100% reliable. Now we're talking about a situation where duplication will be useful. A machine that stopped working recently is likely to have only a few things wrong. This is the type of scenario where you can start to make progress using a duplication and a "change just one thing at a time" strategy.

Duplication is a powerful force, but it won't help you fix this…
(image: barz51 / CC BY 2.0)

Dimensions For Duplication

If the failure scenario is robust, simply "trying it again" will be the only thing required to achieve duplication. However, let's say that your attempts at duplication are initially thwarted. Before you jump to the conclusion that it's an intermittent problem, give some thought to recreating the conditions that were present during the original failure. This may include the same:

- Time of day/week/month/year.
- Temperature, humidity, and other environmental conditions.
- Settings/configuration options.
- For mechanical machines: levels of fuel, fluids, batteries, etc.
- For digital devices: buffer levels, memory consumption, amount of network activity, other programs running alongside, etc.

- Speed/throughput/usage conditions.
- Operators: if the problem only happens when Joe is at the controls, then have Joe present during troubleshooting to do everything the same way (or have him show you so you can imitate).

Looking over this list, you may realize that you don't know what was happening within these categories during the failure. You may also come to the conclusion that you don't even know what the normal operating range is for these parameters in your environment. That may be an indication that it's time to start collecting data.

For Better Or Worse

While attempting to duplicate, also take notice of the things that make a problem better or worse. Especially when a failure is qualitative (i.e., "too much" or "not enough"), you need to pay attention to this aspect of problem replication. In my interview with Jamie, a former motorcycle mechanic, this technique came up while discussing electrical systems:

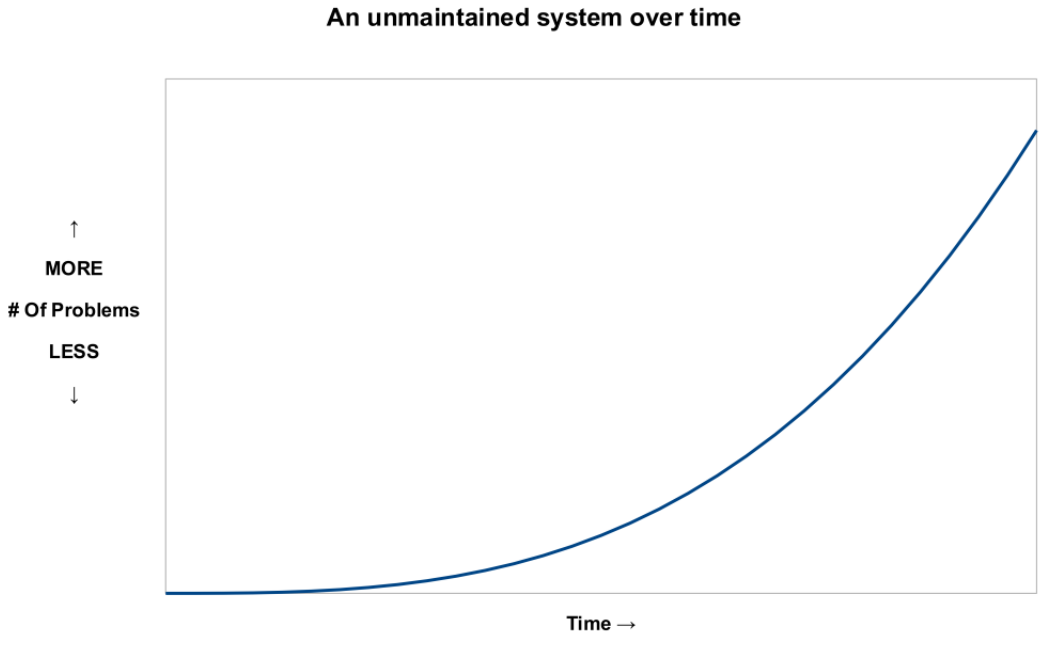
> If it's an electrical problem, I'll try to cordon off circuits that might affect what I'm focusing on. Like if it's an ignition problem and I'm not getting the spark I want. Well, let's disconnect the headlight, because that draws a fair amount of juice. Let's turn off the turn signals, or let's turn them on and see if the problem increases. Removing, but also adding.
>
> **Jamie Karrick**

You may be drawn to things that make a symptom better, but things that make a symptom worse are just as valuable to discover! Both provide valuable information and show a dependency in action. When you find a knob you can twist that affects the problem, good or bad, you are well on your way to understanding the issue.

Time, An Important Dimension

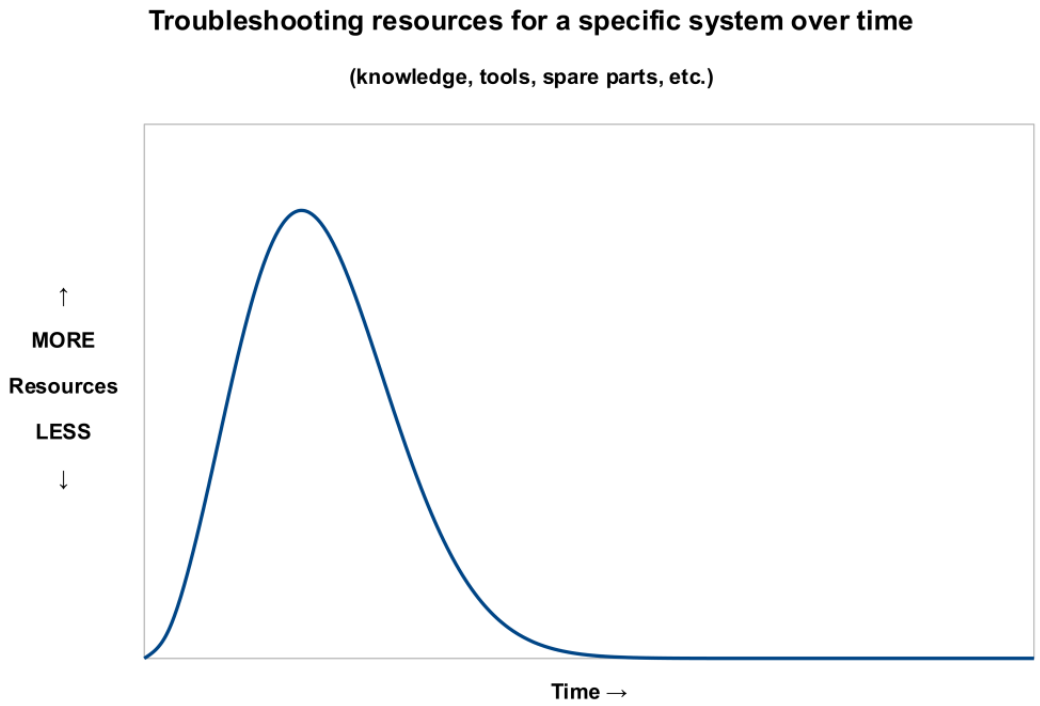
> What is recent is easy to correct.
>
> **Tao Te Ching (verse 64)** [1]

If a long period of time passes after a machine last worked, duplication will likely be an inefficient troubleshooting strategy. Entropy will transform any working system into a pile of worthless parts over the long run:



Graph: The degradation of an unmaintained system over time. Left to sit, a machine will have so many problems that duplication will no longer be a viable strategy for troubleshooting.
(image: © Jason Maxham)

I really want to impress upon you that time is a major factor in all aspects of troubleshooting. As a limited resource, time is something you need to budget carefully while searching for a solution. Time also affects the likelihood of problems (they increase with the age of the machine, as shown in the graph above) and the resources available to you (these decrease with the passage of time, as shown in the graph below):



Graph: especially for mass-produced machines, the resources available to help you diagnose and fix a problem will increase, peak, and then begin a long decline. At first, a machine is widely adopted and the resources for repair increase in response to market demand. Later, the machine is replaced by newer models and outmoded by new technologies, and available troubleshooting resources begin to decline.
(image: © Jason Maxham)

Over time, resources available to help you make a fix can vary widely. Number among these resources: community knowledge, technician know-how, manuals, tools, and spare parts. See the graph above for a visual representation of this phenomenon: with some lag, shortly after a machine is first produced there will be a peak in the resources available to help you troubleshoot. Take the Ford Model T as an example: between 1908 and 1927 over 15 million units of this iconic car were produced. At the very moment the first one rolled off the assembly line in 1908, there probably weren't that many people who knew how to fix them. However by 1927, when there were millions of these cars on the road, it's safe to say that nearly every auto mechanic working in the USA knew something about how to repair the Model T. They were everywhere! Not only that, but the tools and spare parts required were likewise ubiquitous: the simple economics of millions of these cars requiring maintenance meant that the market met this demand with abundant resources.

Finding slack-jawed admirers is the easy part. Spare parts and know-how, not so much…
(image: Carol M. Highsmith / Library of Congress)

Fast-forward to today and how many people know how to repair a Model T? How do you get parts? It's possible, but only a very small group of enthusiasts (antique car collectors and restorers) possess these resources. A hundred years from now, you can bet that this group will be even smaller! Anything can be repaired if you can throw enough resources at the problem, but realize that "fixing" a very old system will put you in a role similar to the original design engineers: prepare to pay the cost of discovering how to make it work, just like they did.

(Continued in Part 2: Failing To Fail)

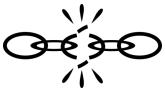
References:

- Header image: Mydans, C., photographer. (1935) *Row of identical houses off Eastern Avenue, in Cincinnati, Ohio, showing backyard outhouses. Ohio River Valley is in the distance.* United States, Ohio, Cincinnati, Hamilton County, 1935. Dec. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017759064/.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 64.

Duplicate The Problem was originally published December 14, 2011.

Notes:

Failing To Fail (Duplicate The Problem, Part 2)

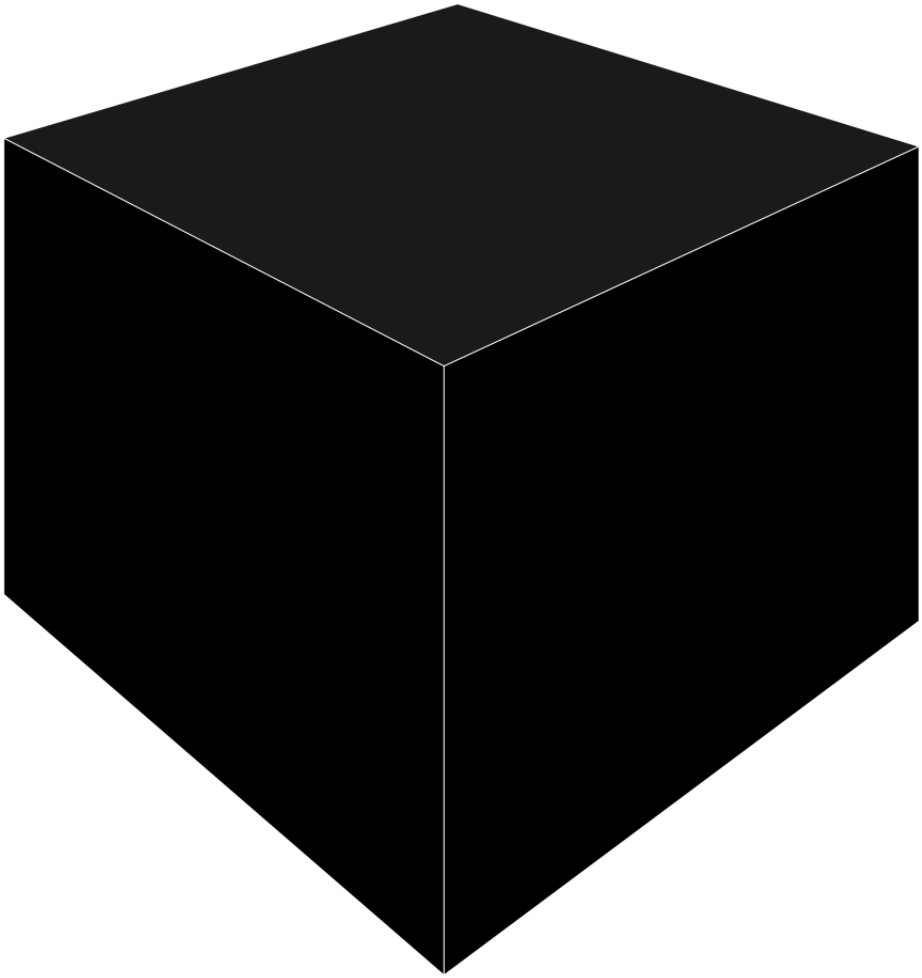


> Reproducing it is often the hardest part.
>
> **Alex Chaffee**

We've already been over "duplicating the problem" as a core strategy of the troubleshooting process. However, if you've been in the trenches long enough, you know that some problems resist duplication. During these moments, you may shake your fist at the heavens and shout, "Why won't you fail when I want you to?!" Fixing these tricky issues is when you'll really earn your pay. Eventually, you'll appreciate the rewards, both spiritual and material, of solving these tougher failures.

The Black Box

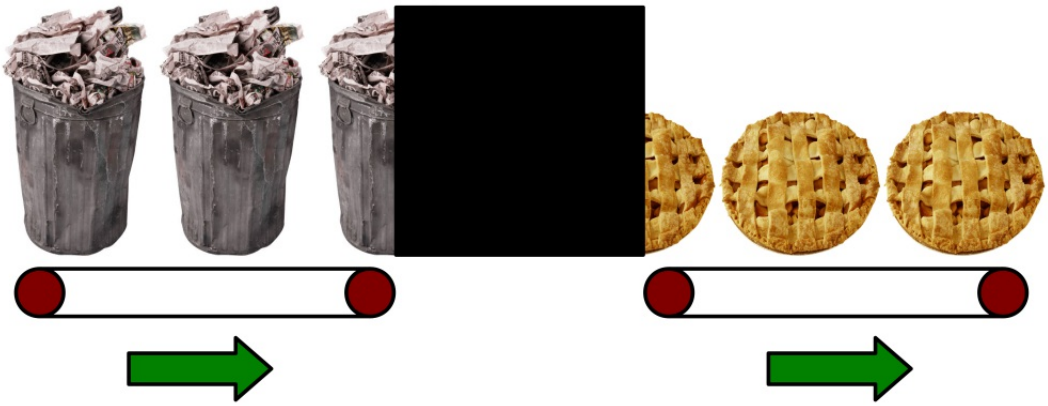
Let's say that, despite your best efforts, you're unable to reproduce a problem. Ugh. Well, there is a class of troubleshooting problems, simply known as "intermittent." They will be difficult to duplicate, even when you have attempted to painstakingly recreate the original environment in which the failure occurred. Of course, you may believe you've set up everything the same, but *something* is different. And that *something* is making a difference.



The Black Box: sometimes you care what goes on in here, and sometimes you don't.
(image: © Jason Maxham)

Based on my experience, the inability to recreate a failure is typically a variation of what I call the Black Box Problem. You may have heard the term "black box" thrown around in the world of engineering and product design: it may be used as both a damning slur or a coveted feature, depending on whether you're hands are covered in grease while cursing or reading a slick marketing brochure.

A Black Box system takes input on one end and magically produces the desired output on the other end. Of course, it's not really magic, it just seems that way when you don't know what's going on inside. For instance, even though I love audio gear, I really have no clue about the inner workings of the amplifier in my stereo. All I know is that I plug my music player in one end and sound comes out the other. Another example is Google, probably the greatest Black Box known to man: you type your search in a box and out springs millions of relevant web pages (although technically theirs is a Blue Box, not a black one). Calling something a Black Box doesn't mean the system is unknowable: clearly, the original designer of the Black Box knew what was going on internally.



The ideal Black Box works like this…
(image: © Jason Maxham)

Hiding complexity behind Black Boxes is essential to enable the average person to use technology in our modern civilization. I don't care to know the intimate details of what goes on inside my amplifier, I simply don't have the time or inclination to acquire that knowledge. However, I do want to listen to music in my living room. I might even dance, if the shades are drawn. Likewise, it would be exhausting to have to interact with all of those unshaven programmers at Google every time I wanted to search for something on the Internet. Black Boxes allow a broader range of society to participate in and benefit from technological innovation by making complicated things simple. "Yay!" and a slow clap for all that.

What's the downside? Well, for the same reason that Black Boxes makes our life easier (you don't have to know how they work to use them), they also present a problem for the troubleshooter. As long as it works flawlessly, you can be blissfully ignorant of what goes on inside a Black Box. Of course, all machines eventually break down and this is when you'll need to make the transition from ignorant to enlightened (or preferably beforehand, if you want to be prepared).

If you think that troubleshooters only encounter Black Boxes in the world of off-the-shelf consumer products, think again. The root of the Black Box Problem is a lack of knowledge, so any machine can be a candidate. You may have unwittingly created a Black Box, especially if you've cobbled together a Franken-system of parts you didn't build (Systems Integrators are especially likely to encounter this problem, given this is their job description). Creating a machine with your own two hands won't confer automatic knowledge of how it will be used, how it will fail, or what happens internally while it is operating. When you build something, you usually devote a lot of time to *testing*, which is the first-time discovery of these aspects. This learning continues long after a machine is deployed and people begin to report their successes and failures actually using it for work. Designing a machine involves many humbling surprises, so let go of the notion that creating something is the same as knowing it.

In addition to the systems we create, Black Boxes can also spring into existence if knowledge about a system is lost: consider the case where only one person in a company knows how to fix a machine and they retire or leave to take another job. If you're put in charge of its maintenance and repair, that machine just became a Black Box—to you!

Thinking Inside The Box

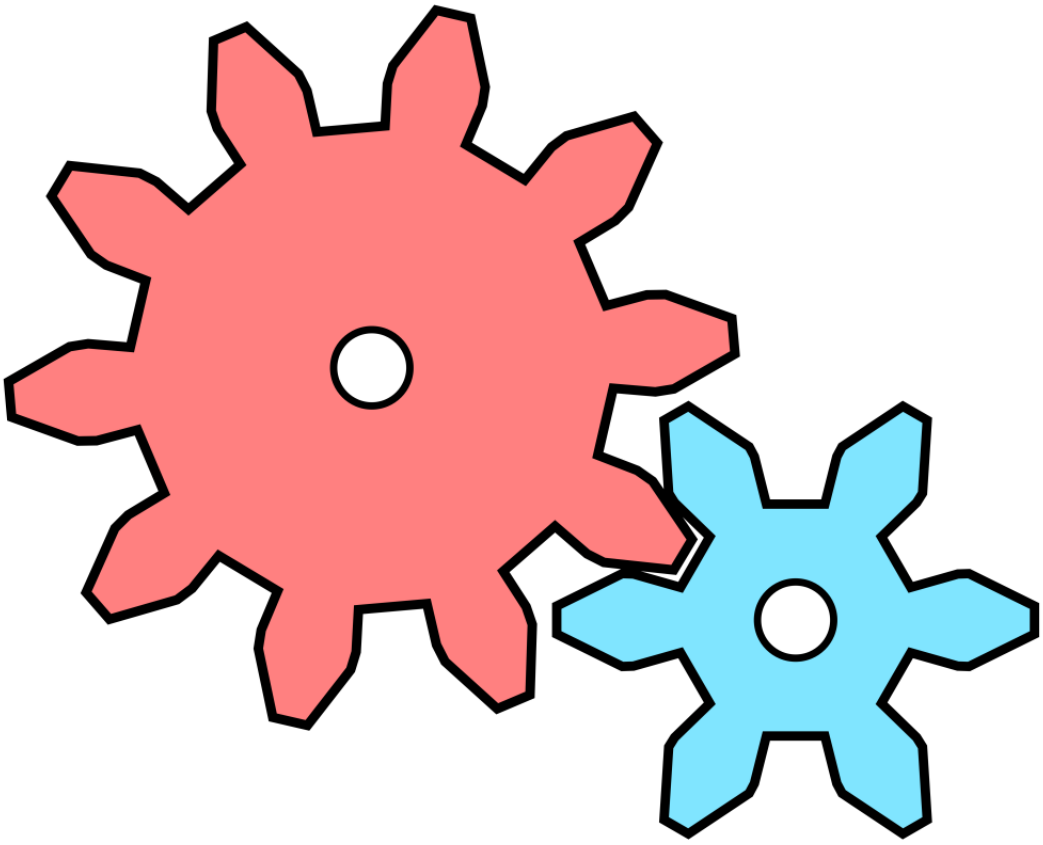
The solution to solving intermittent problems is to demystify the Black Box. This means inserting probes to observe the functioning of individual parts, subsystems, components, or stages along a transformational chain. From there, start to look for correlations among those readings to get ideas for further investigation. Looking at data from probes is always mind-expanding: the internal workings of even a machine you think you know well often astonishes. In fact, **expect to be surprised**: if you knew absolutely everything about the internal workings of a system you're repairing, you'd already have solved the problem, right? Right.

Circularity: Repeating Positions And Processes

A good start to understanding intermittent problems are what I like to call "circular" components. I put circular in quotes because we're not always talking about actual circles here, but rather any part that follows a pattern of returning to where it started during the course of doing work. Examples: wheels, fixed-digit counters (e.g., something that goes:

"00", "01"…"99", "00"), belts, gears, sprockets, chains, gates, switches, pistons, etc.

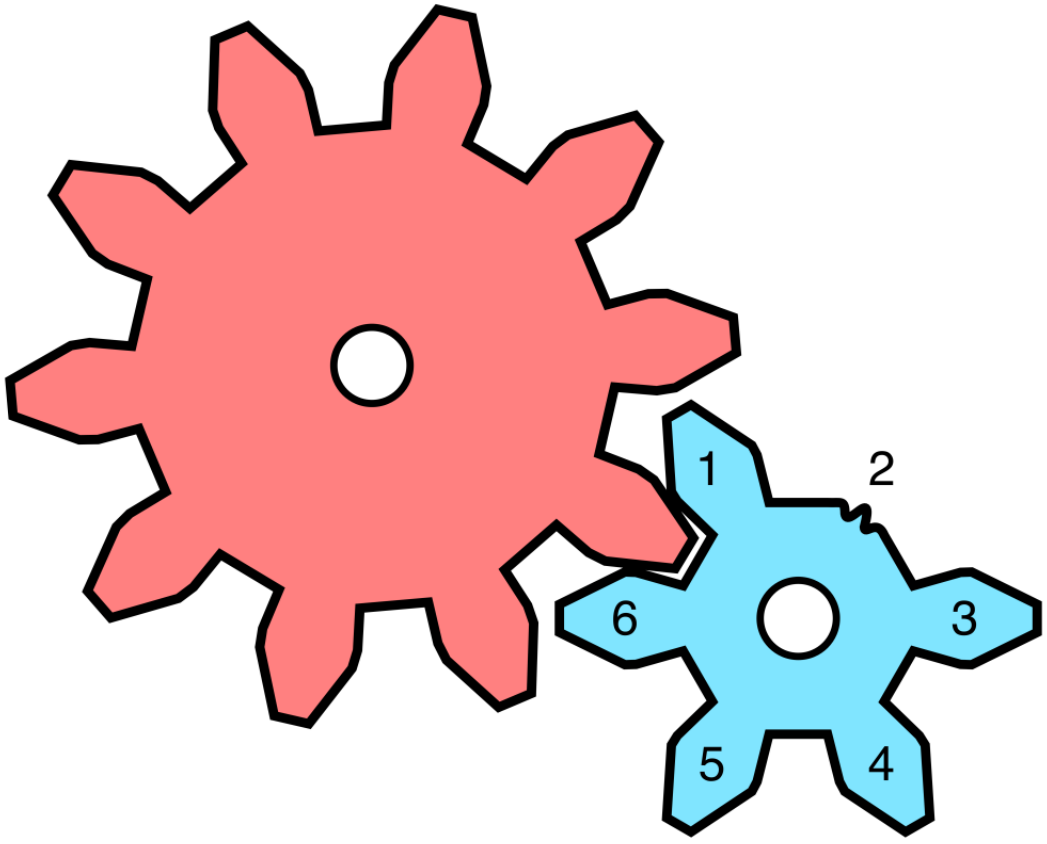
The relationship to intermittent problems is that one or more of the positions in the repeated pattern may be failing. Many circular systems are designed to keep going even though one of the steps in the cycle is failing. Meshed gears elegantly illustrates this principle:



Meshed gears: 10-tooth with 6-tooth.
(image: © Jason Maxham)

First, observe the circular nature of the smaller gear in this machine: it returns to where it started after cycling through all 6 of its teeth.

But, let's say that one of the teeth on the smaller gear breaks:



A broken tooth (labeled #2) on the smaller gear will intermittently move the larger one.
(image: © Jason Maxham)

If this machine is designed to do something with each partial turn of the smaller gear, the machine will fail intermittently because of this single broken tooth. Specifically, it will work 5 times in a row, followed by a failure, then work 5 times in a row again, followed by another failure, etc. I've numbered the teeth on the smaller gear in the diagram to show you what happens in each iteration:

Sequence #	Tooth Meshing	Status
1	1	OK
2	2	FAULT
3	3	OK
4	4	OK
5	5	OK
6	6	OK
7	1	OK
8	2	FAULT
9	3	OK
10	4	OK
11	5	OK
12	6	OK

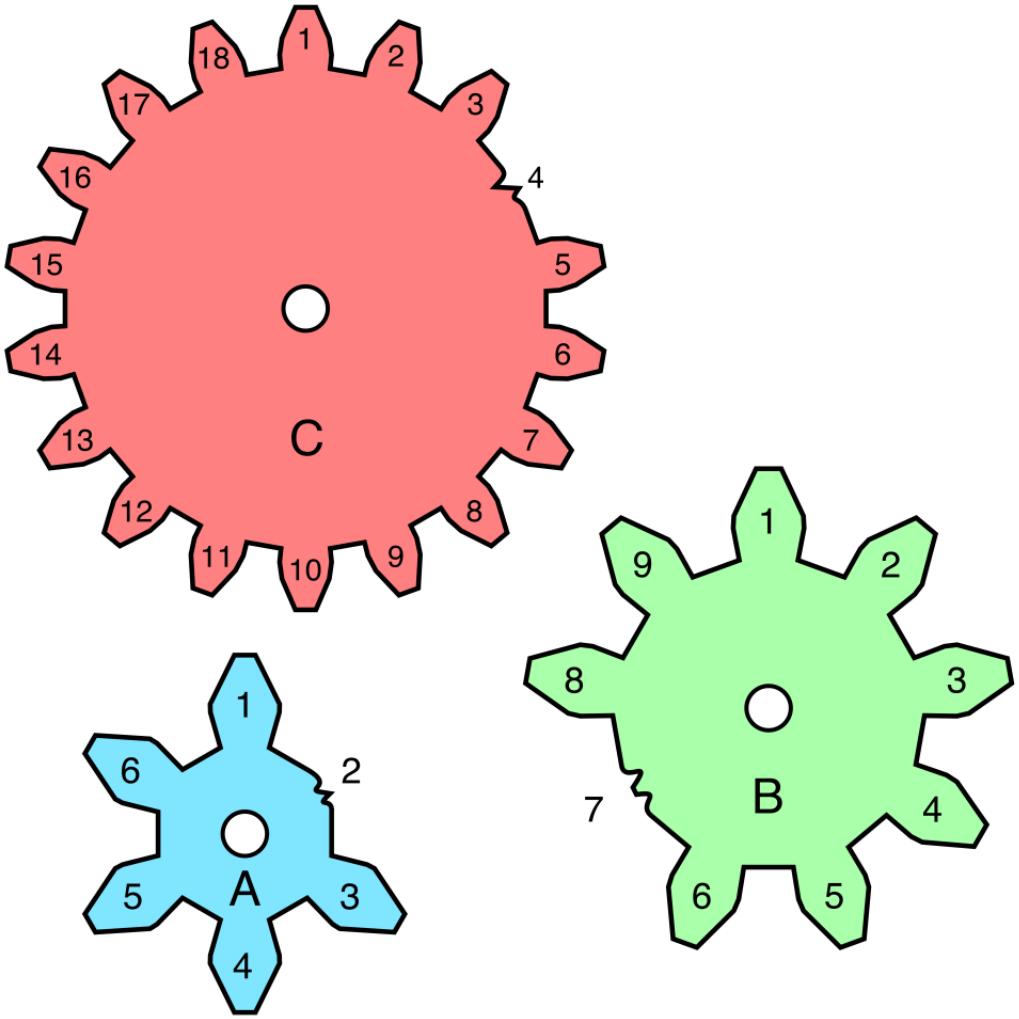
If you were troubleshooting this machine and being observant, you might notice this predictable ratio of 5:1 successes to failures, along with their reliable sequence. This may be an intermittent problem, but it's also a very reliable one. Interestingly enough, missing teeth on gears are utilized *on purpose* in the design of analog counters. For that particular purpose, intermittent behavior is desired: on a counter you want the 10s position to increment only once for every 10

revolutions of the 1s counter. Take a look at the beginning of the video "Building the mechanical counter" (0:30-0:45) to see the use of missing teeth in action for the construction of a counter.

Missing teeth on a gear can be a design feature or cause a breakdown, depending on the context. What lies behind a failure can sometimes be used elsewhere for good. This is a unique perspective on engineering that you will develop as a troubleshooter.

The real world will present you with intermittent failure scenarios far more complicated than the above example of the single missing tooth in a gear. Let's expand on this and imagine a machine with several intermittently failing "circular" components. Their combined behavior will result in very complicated intermittent failure pattern that will, at first glance, *appear* to be random.

Let's start with 3 gears this time and add some broken teeth into the mix:



Three gears (A, B, and C), each with a broken tooth.
(image: © Jason Maxham)

Numbering the teeth from the top clockwise, you can see that:

- the blue 6-toothed gear (A) is missing tooth #2
- the green 9-toothed gear (B) is missing tooth #7
- the red 18-toothed gear (C) is missing tooth #4

We'll set up our scenario with some assumptions:

- The broken gears are located in different places in a machine, independently supporting different functions.

- Gear speed, measured in teeth per minute, is the same for all gears. This means that the gear with 6 teeth will make 3 revolutions in the same time it takes the gear with 18 teeth to make one.
- If any of the gears is currently turning through a broken tooth, the machine will temporarily cease to do its work.

The result of the missing teeth in the separate gears, although acting independently, will together produce a very erratic and *intermittent* failure scenario:

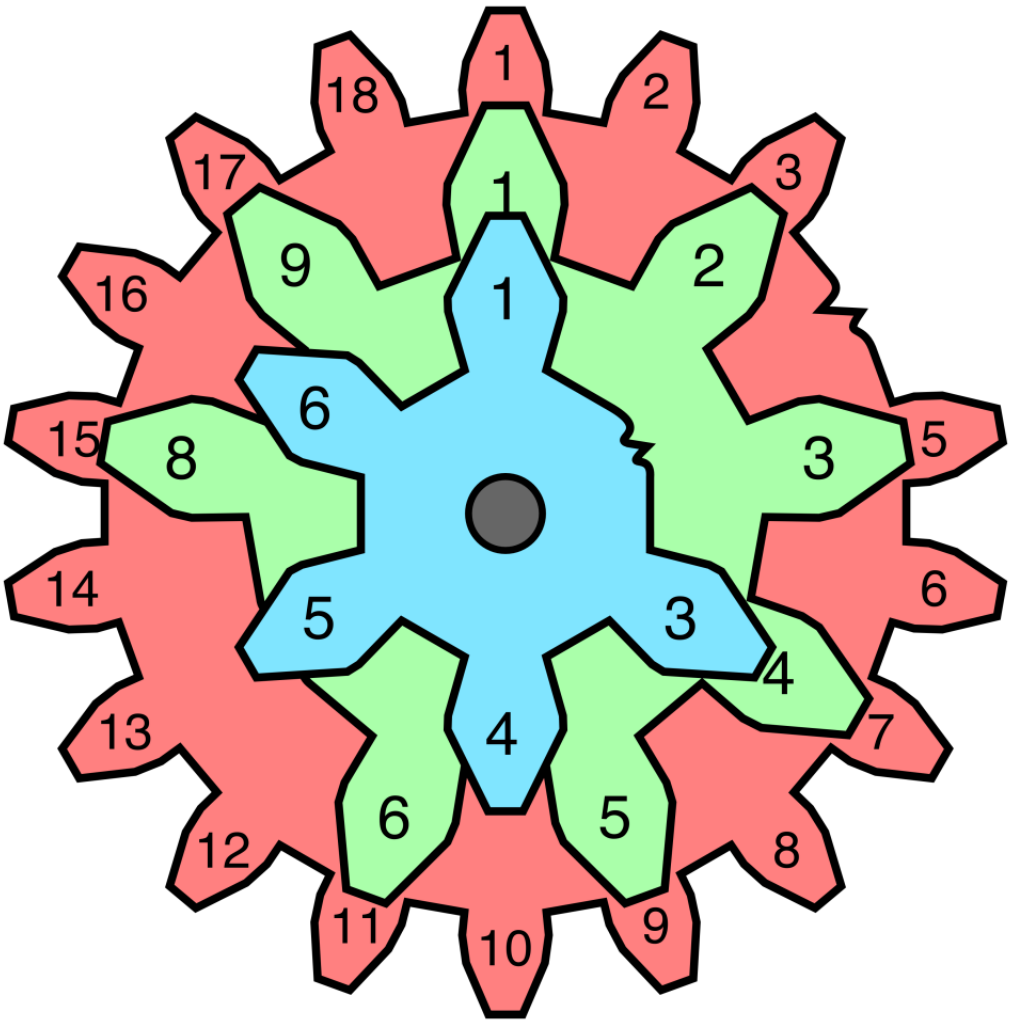
Sequence #	Machine Status	Tooth Meshing		
		Gear A	Gear B	Gear C
1	OK	1	1	1
2	FAULT	2	2	2
3	OK	3	3	3
4	FAULT	4	4	4
5	OK	5	5	5
6	OK	6	6	6
7	FAULT	1	7	7
8	FAULT	2	8	8
9	OK	3	9	9
10	OK	4	1	10
11	OK	5	2	11
12	OK	6	3	12
13	OK	1	4	13
14	FAULT	2	5	14
15	OK	3	6	15
16	FAULT	4	7	16
17	OK	5	8	17
18	OK	6	9	18
19	OK	1	1	1
20	FAULT	2	2	2
21	OK	3	3	3
22	FAULT	4	4	4
23	OK	5	5	5
24	OK	6	6	6
25	FAULT	1	7	7
26	FAULT	2	8	8
27	OK	3	9	9
28	OK	4	1	10
29	OK	5	2	11
30	OK	6	3	12
31	OK	1	4	13
32	FAULT	2	5	14
33	OK	3	6	15

34	FAULT	4	7	16
35	OK	5	8	17
36	OK	6	9	18

If you could only see the external result, this machine's behavior would seem very bizarre. Sometimes it fails once and sometimes twice in a row, with the sequence of successes and failures being: 1,1,1,1,2,2,5,1,1,1,2. Also, it takes 18 iterations for the complete failure pattern to emerge! Of course, you wouldn't even begin to recognize it as a discrete pattern until you had observed two full cycles (that's why I included 2 × 18 = 36 iterations in the table above). Would you have the patience to observe and systematically record all this until the sequence emerged? I doubt I would.

A Cover-up: The Masking Effect Of Time-Aligned Failures

Now, we'll examine how concurrent failures can hide each other. Let's take the gears from the previous example and mount them on a common driveshaft:



Gears (A,B,C) installed on a common driveshaft. Again, each gear is missing a tooth:
A (6-teeth, blue): #2
B (9-teeth, green): #7
C (18-teeth, red): #4
(image: © Jason Maxham)

For this example, we'll say that these aligned gears (A,B,C) are meshed at the 12 o'clock position with 3 separate gears (those connecting gears not shown in the diagram). As before, we'll assume that if any gear is meshing on a broken tooth, the machine will temporarily stop doing its work. Also note: because the gears are being spun by the same driveshaft, the time it takes for the small gear to turn through 6 teeth is equal to the time it takes the larger gear to turn

through 18 teeth. Here's what the failure pattern will look like:

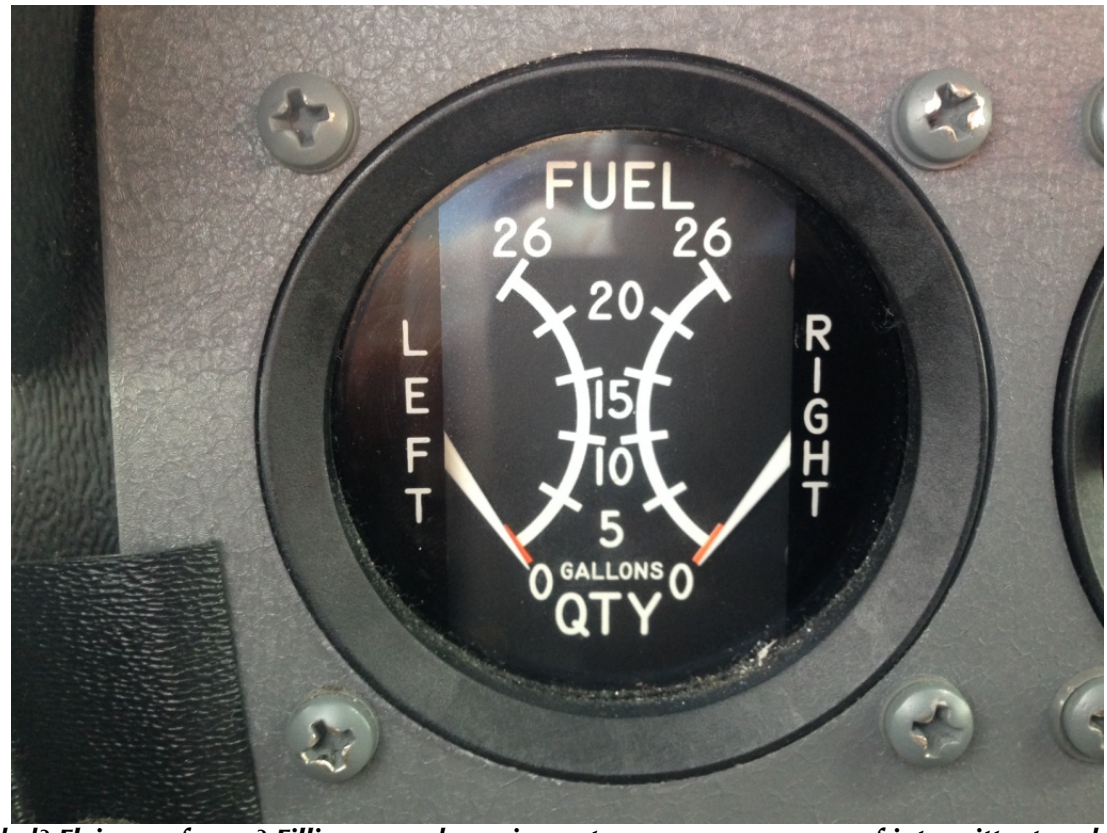
Sequence #	Machine Status	Tooth Meshing		
		Gear A	Gear B	Gear C
1/18	OK	1	1	1
2/18	OK	1	1	2
3/18	OK	1	2	3
4/18	FAULT	2	2	4
5/18	FAULT	2	3	5
6/18	FAULT	2	3	6
7/18	OK	3	4	7
8/18	OK	3	4	8
9/18	OK	3	5	9
10/18	OK	4	5	10
11/18	OK	4	6	11
12/18	OK	4	6	12
13/18	FAULT	5	7	13
14/18	FAULT	5	7	14
15/18	OK	5	8	15
16/18	OK	6	8	16
17/18	OK	6	9	17
18/18	OK	6	9	18

Again, a more complicated pattern emerges with multiple failings gears, versus just a single failing gear. An interesting aspect of this example is that the missing teeth on Gear A (#2) and Gear C (#4) mesh at the **same time.** You can see from the table that these time-aligned failures mask each other. Even if you found and fixed the missing tooth on Gear C, the machine would still fail the same way until you also fixed Gear A.

It Only Gets Weirder

Hopefully, by now you've grasped the concept: malfunctioning parts, each with their own patterns, can act together to produce much more complicated and *intermittent* failures. When viewed individually, the broken gears I've used as examples at least have a reliable error sequence. While gears are a great model for understanding intermittent failures, in the wild be prepared for malfunctioning components with no discernible failure pattern. Combine several of these together and you'll have a recipe for intermittency that looks perfectly chaotic.

When I see completely random behavior like this, I like to take my investigation "up a level" and look for environmental or system-wide explanations. What comes to mind is the time when the cooling system in our data center went down. As our servers were pushed out of their recommended temperature ranges, *very* strange things began to happen. Random reboots, crashes, slowness. Everything bad, and in large quantities. Even under normal conditions, the usage patterns of individual servers in a computing cluster are highly variable and dependent upon the work being performed. Add to that a cooling system loss, which brings the uneven effects of hot air collecting and dispersing in a disorderly fashion, and you have a recipe for failures that will be impossible to duplicate.



Flooded? Flying on fumes? Filling up and running out are common causes of intermittent problems.
(image: RobbieMcConnel / CC BY-SA 3.0)

Running On Empty

A related concept to "circular" components are things that might be "filling up" or "running out." Depending on what is being emptied or filled, these conditions may cause a machine to stop operating intermittently. On the "filling up" side, this could be something like a reservoir that holds waste fluids or the output tray on a copier. When it comes to things "running out," examples might include fuel being fed to a motor or disk space on a computer. The key is that the fill or depletion rate might vary, making the timing of the failure variable as well. Combine several of these buffers or reservoirs that are "filling up" with other resources that are "running out" and you have the recipe for very complicated intermittent failure scenarios (similar to the multiple gears example above).

Probes and gauges to the rescue! If you can be aware of the "filling up" and "running out" of the machine's resources, you can begin to correlate these events to breakdowns. Automated monitoring and alerting is the preferred end game for catching things that have run out or filled up (e.g., a "low oil" light on an automobile dashboard).

What's Going On In There, In The Space Between?

Once you've opened up the Black Box, you can see if the internal components are behaving in a predictable way. If you look closely enough, you may find otherwise. Testing the parts individually, you may find that one (or more) is failing intermittently and therefore causing the intermittent failure of the entire machine.

Take the example of an internal combustion engine as seen from the perspective of your average car owner: they put gas in one end and the car gets propulsion on the other. Another Black Box. Of course, there's a lot going on inside an engine and there's a long list of things that must be right for it to work properly. When it comes to monitoring, you could watch the pressure inside every cylinder, the spark plug voltage during the firing/recharge cycle, the consistency of the air/gas mixture, etc. If you think obtaining this level of detail is fantasy, you probably have never seen a modern analyzer in an auto repair shop. Engine analysis before the era of computers used more primitive tools and therefore more work was required to gather this kind of data. But, antique or modern, the need to dig into and understand the Black Box is the same. Understanding the whole by understanding the parts is a key part of this demystifying process.

While we're probing the goings-on within the Black Box, we also need to include the interfaces *between* components. These points of connection are another common source of intermittent failures. They include examples like: oxidized

contacts between a speaker and an amplifier, a broken clip on a network cable, or a loose screw that terminates a wire in an electrical plug. I think there's two main reasons why these "in-between" parts are so vulnerable to wear and tear:

1. They are often conduits responsible for transferring energy or work. You may have heard the expression "Where the rubber meets the road." This a perfect description of the function of interface parts and the reason why they take such abuse.
2. Within their context, these parts are often externally located and therefore more exposed to hazards or the elements. You can have a beautiful machine encased in titanium, but it likely has a power cord coming out of the side, which can be punctured, squished against a wall, or rubbed until it's bare.

Quantum Troubleshooting

Will always having the same inputs lead to the same outputs when troubleshooting? Chaos theory would say otherwise. Yes, if everything was *exactly* the same, down to the subatomic level, perhaps you could guarantee that the same inputs would result in the same outputs. However, until we're arranging particles at whim, you'll have to realize that there will always be small differences between the original failure scenario and your attempts to recreate it for the purpose of duplication. There's also one big part of the original context that will be impossible to replicate: time. Even if you could magically place every particle in the exact same place, you simply can't roll back the clock to recreate the same time context in which a failure occurred (where's my time machine, already?!).

Troubleshooting is a present and future-oriented exercise. The strategies presented here, especially ones like "don't fix it," are pointed toward a future that is *much better* than the past. Even if you could add a Time Machine to your toolbox, you wouldn't want to! Alright, a time machine that fits in a toolbox *would* be pretty cool. It's just that I'm sure we could find better uses for it than fixing your car. We'd use it to witness important stuff, like the invention of the doughnut.

If your efforts at duplication continue to be thwarted, even after you start monitoring the internals of a system, you might be dealing with a chaotic system. Chaotic systems will take *small* differences in starting input and turn them into very *large* differences in outputs. This will manifest itself in behavior that *appears* to be random. It's not a machine, but a good example of a system that frequently resists duplication is the weather. Even though some of the parts are well understood (how clouds form, seasonal temperature patterns, effects of changes in atmospheric pressure, etc.), they interact in unexpected ways on a massive scale that frustrates reliable prediction. The weather may be the ultimate example of this, but I've worked on systems that are just as frustrating to understand. For the curious, there's a whole field of research devoted to the "control of chaos." The technology that makes lasers possible (the awesome-sounding "OGY Method") was one of the first achievements in the battle against chaos.

Chaotic systems tend to be very complicated (or at least appear so) and have many components, so prepare for some serious troubleshooting. Here are some strategies for bringing stability to a chaotic system:

- **Reduce complexity:** pare back the number of subsystems, variables, and settings, along with opportunities for interactions between them. Especially in networked systems, where every node is connected to every other node, the number of possible connections grows exponentially as you add nodes.
- **Lock down and standardize the flow between parts:** you may need to introduce governors or limiters to dampen the effects of swings in inputs and outputs. Specifying and enforcing limits will make subsystems operate in a narrower band, which can reduce chaotic behavior.
- **Use automated monitoring and correction to restore service:** until you understand what is driving the chaos, a good temporary solution is to monitor your systems and make automated corrections based on that monitoring. As I point out in "Defaults and Reboots", sometimes this path is good enough to work indefinitely if the cost of understanding the root cause is prohibitive. Can you automatically press the "reset" switch whenever a system begins to behave chaotically? Open or close pathways, speed or slow throughput, take subsystems on or off line: these are all options when automated monitoring enters the picture.

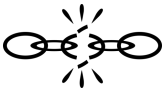
References:

- Header image: Lee, R., photographer. (1939) *Stitching cardboard boxes. Grapefruit canning plant, Weslaco, Texas.* United States, Weslaco, Texas, Hidalgo County, 1939. Feb. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017782081/.

Failing To Fail (Duplicate The Problem, Part 2) was originally published February 14, 2013.

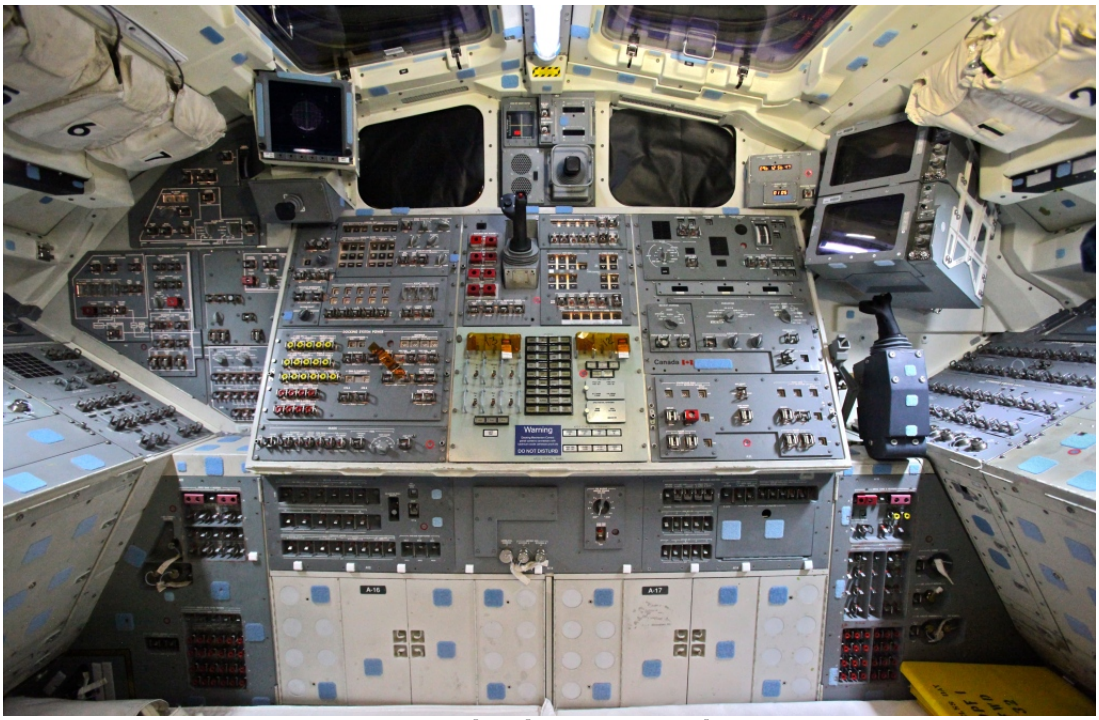
Notes:

Defaults And Reboots



> A lot of times, I'm surprised by how simple the solution is.
>
> **Jamie Karrick**

Any device that is configurable is vulnerable to errors stemming from those same settings being mis-configured. Did a switch or bit inadvertently get flipped, resulting in a malfunction? On a machine with many options, this might be hard to figure out. Therefore, a crude but useful way to determine if the machine is functional is to restore the default settings. On digital devices, this is usually very easy: you select one option and *BAM!*, the device reverts to how it came from the factory on the day you bought it. Mechanical devices also have "default settings", but you might have to look in the manual to find them (i.e., there may be no magic switch that restores them like on a computer).



Um…is this thing set up right?
(image: Steve Jurvetson / CC BY 2.0)

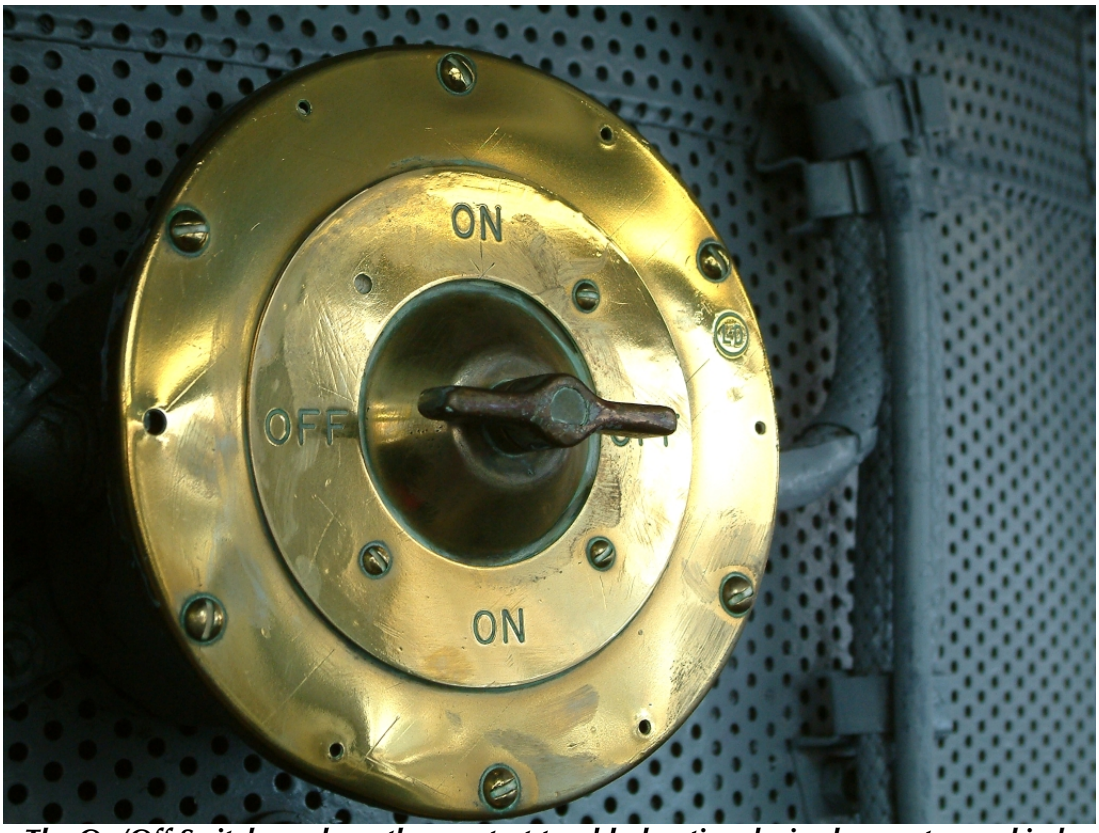
Once you've verified that the machine works with the default settings, you can begin to change the configuration back to your desired settings. If you do this using my "change-just-one-thing-at-a-time" philosophy, you should be able to identify the particular setting that is causing your machine to malfunction.

Alternatively, you may have created your own "default settings": think of a network router that has been configured to work on your network (with your IP addresses, netmasks, routing table entries, etc.). A scheme like this usually means these settings are automatically loaded when the device starts up. Again, these could have been inadvertently changed, so restoring the machine to your desired settings (i.e., your own personal "defaults") will at least eliminate the possibility that a configuration problem is the cause.

Off And On Again

Turn it off. Turn it back on. Whether known as rebooting, power cycling or restarting, this is such a simple yet powerful troubleshooting trick that it deserves its own section. Actually, given how often it works, a shrine would be a more fitting tribute.

Even as my troubleshooting skills have grown over the years, the reboot technique still has its place among my top strategies. Since I'm really into troubleshooting theory, sometimes I want to make things more complicated than they really are. But turning it off and then turning it on again has solved so many problems in so many different contexts that you always have to ask yourself: "Why am I not using this strategy?"



The On/Off Switch: perhaps the greatest troubleshooting device known to mankind.
(image: Markus Tacker / CC BY-ND 2.0)

But Why Does It Work?

How the on/off switch returns a machine back to normal brings together several concepts that we've previously discussed. A restart usually restores a system to a simplified state, along two dimensions we've covered:

1. Just the basics: many machines will start up in their most primitive state, with additional modules or subsystems deactivated. As noted, fewer subsystems in use lessens the chance of unwanted interactions among them.
2. Configuration reset: usually, the longer a machine is on and used, the more its configuration will be changed. This means that, over time, the probability of choosing an error-prone configuration will increase. Since most machines will have a default "startup" configuration (usually designed by the manufacturer to always work), the on/off strategy can be shorthand for removing a bad configuration.

Lastly, restarting can solve another class of problems: corruption that occurs as a result of use. Over time, operators and circumstances will put a machine through its paces. Buffers and reservoirs will fill up or empty out, cruft will accumulate. Because many machines have automated start up procedures that restore a "clean" configuration, the on/off switch may catalyze a simpler state and clear away these issues.

Off Putting

Every troubleshooting strategy has a context in which it's *not* to be used. While we may agree that the on/off switch may have the highest return on investment if you were to rank all available options, there are some times when it shouldn't be used. It may seem obvious, but I must point out that not every machine can be turned off without serious consequences. A respirator keeping a patient alive or the lone engine on an airplane in flight are bad candidates for the restart strategy. Besides the obvious reasons of interrupting someone's breathing or cutting off the only source of thrust for a plane in motion, restarting a machine has another big risk: IT MAY NOT COME BACK TO LIFE AFTER A RESTART.

This is a big factor for machines that have been running for a long time. I've personally observed computers that have been continuously humming away happily for years, only to die on reboot. As long as it remained in motion, things were fine; however, as soon as forward progress was interrupted, it stopped working. Humans may be the same way, our lives require a forward momentum that is sometimes not advisable to interrupt. Think of all the people who have

died shortly after retiring…

So, always consider the context in which a system lives before reaching for that power switch. If the machine has been continuously running for a long time and is critical to your designs, be sure to have a good Plan B before restarting.

Please stand by while I reboot the entire Internet…
(image: Karl Baron / CC BY 2.0 / Cropped from original)

A Long-term Workaround…?

You may encounter problems that can be consistently solved by restarting. Would it be acceptable to use this as a workaround for the long-term? Would this exclude you from considering yourself a Master Troubleshooter? Doesn't the Master Troubleshooter always want to know *why* something doesn't work?

The answer? No. Remember that all troubleshooting decisions have an economic component. The cost of figuring out the *why* behind a failure may be prohibitive. If a simple reboot will right things and can be incorporated into your workflow, go for it! I've seen people automate restarts, rebooting a system at the beginning of every workday, before every shift, etc. Of course, make sure you take into account the full cost of unexpected interruptions before you make restarting a permanent part of your routine.

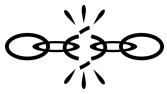
References:

- Header image: Historic American Engineering Record, C., *36. INTERIOR VIEW, BERK SWITCH TOWER, SOUTH NORWALK, SHOWING SWITCHING LEVERS FROM OPERATOR'S POSITION – New York, New Haven & Hartford Railroad, Automatic Signalization System, Long Island Sound shoreline between Stamford & New Haven.* [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/ct0380/.

Defaults And Reboots was originally published December 22, 2011.

Notes:

Change Just One Thing



> I take a scientific method approach: only change one variable at a time.
>
> **Mike McCormick**

Most troubleshooting exercises will be a narrowing to a single failed component (and hopefully, it's just *one* broken part!). Paring down a list of possibilities leads to isolation, which in the context of troubleshooting means that you can confidently point to a particular part of a system and say, **"The problem is somewhere in here."** After you've made your best guess as to the cause and you're ready to test a particular fix, only change **one** variable at a time. Most importantly, you should **re-test for the failed condition after each change**. If you don't, you may solve the problem, but you won't know what you did to solve it.

You should view each troubleshooting session as an opportunity to learn; changing multiple things simultaneously will destroy the knowledge that could be gained from a failure. If you're the kind of person that likes to continually improve the world around you (or, close enough, you're responsible for making a system work), you should crave getting to the root of a problem. Since if often takes multiple failures to have that "Aha!" moment of realization, treat every breakdown as a precious lucky break to start this learning process.

If you want to understand cause and effect, only change one thing at a time. However, the 5-year-old in me just wants to twist all these knobs and slide all these sliders.
(image: eyeliam / CC BY 2.0)

Economy Of Action

I'll often be talking with someone who got something professionally repaired and they'll relate to me how pleased they are:

Me: "What did they end up doing to your car?"
He Who Just Paid A Large Repair Bill: "Let's see, they replaced the fuel pump, battery, hoses, fan belts, and adjusted the valves."
Me:"Wow, but…which of those repairs fixed the problem?"
He Who Just Paid A Large Repair Bill: "I don't know, but it's definitely gone now!"

The problem better be solved, because every component even remotely related to it was replaced! The economics of troubleshooting often favors swapping over repair, especially if the cost of parts is low and the price of labor is high. But, keep in mind the counterbalancing "cost" of shotgun-style repairs: they do little to advance your knowledge of what went wrong.

Of course, I'm not the first person to note the necessity of only changing one thing at a time when determining cause and effect. This is a bedrock principle of the scientific method, which involves modifying a single "independent variable" and observing the results. The people in white coats do this to ensure that they are conducting a "fair test" and that their results will be repeatable for other scientists. A good experiment has static conditions for every round of data collection, save for that lone variable being adjusted, to ensure that any changes stem from it alone.

The scientific method has much to teach the troubleshooter: think of the broken system's output as the **dependent variable** in your fix-it experiment. You form your hypothesis for the cause of the failure, and your proposed fix is the **independent variable** that will be changed. If you find that your hypothesis was wrong (and therefore you'd like to test another theory that involves a different fix), **be sure to put the previous independent variable you changed back to its original condition.** This means putting the system back exactly like you found it: settings, parts you may have taken out, etc.

Before I knew better, there were so many times where I would attempt a different fix, but not put a machine back to the

way it was before trying the previous one. Not only does this scenario violate the “change just one thing” principle, but the change you’ve failed to revert may now be a new, additional cause of non-operation. Prepare for frustration as now you’re troubleshooting two problems instead of one!

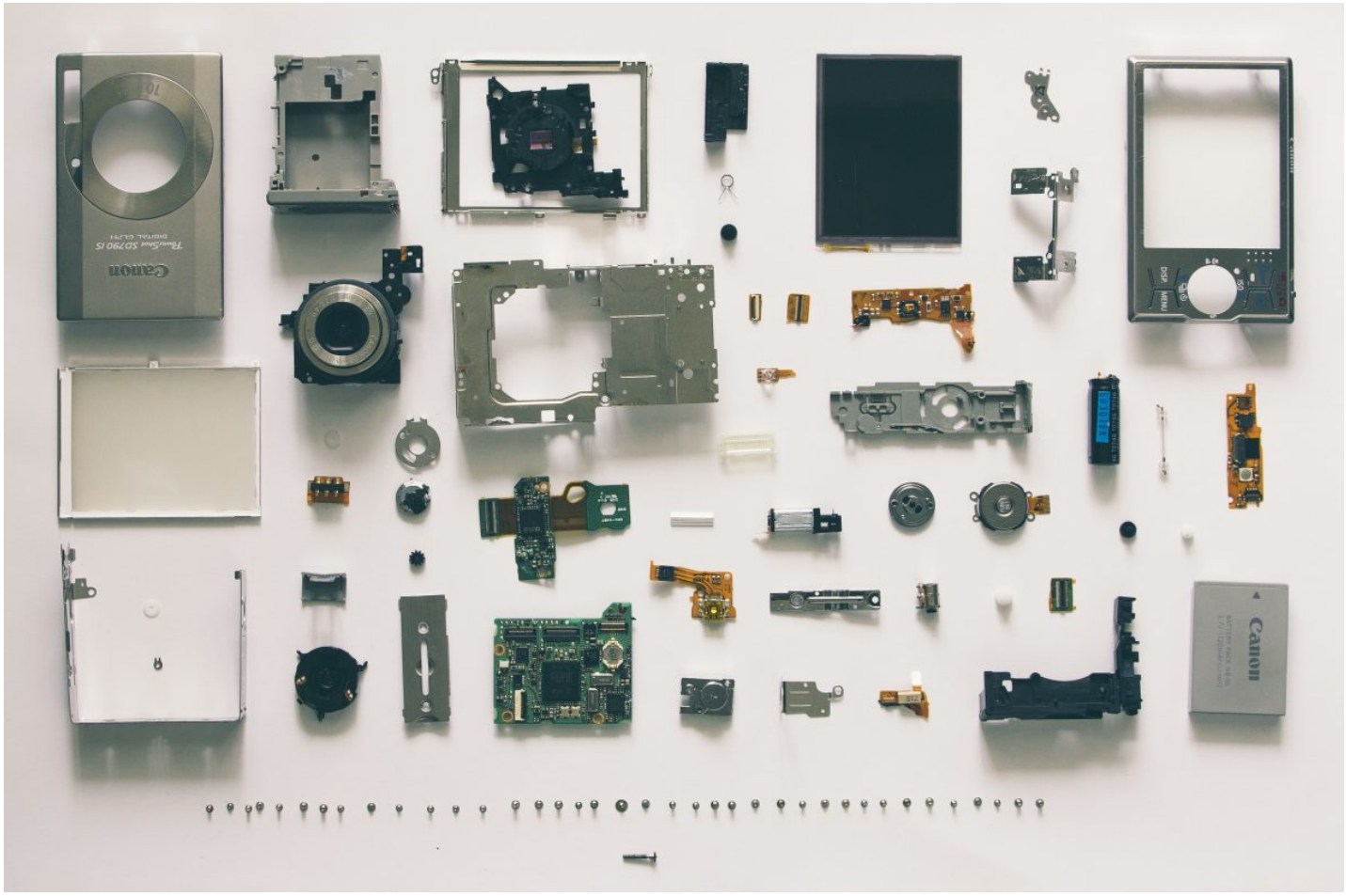
References:

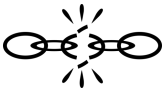
- Header image: *Agrl. Dept., Biological Survey laboratory*. ca. 1920. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016852276/.

Change Just One Thing was originally published January 11, 2012.

Notes:

The Way It Is And The Way It Was





> Write everything down. Once you've started a problem, write down what you did. Each step. Because, you will not remember it. You'll remember 5 or 10 minutes from now, but you may not solve it in 5 to 10 minutes. Three hours later, you won't know what you tried! Especially if you don't know how to get to reproducibility, your memory will lie to you. You won't remember the exact details of how you did something...
>
> **Karl Kuehn**

Before you dive in and start tearing something apart, take a moment, close your eyes, and imagine yourself putting it back in working order. Through the course of your little thought experiment, you may realize that you don't have a clue how it was put together. Which leads us to the following troubleshooting commandment, sent down from on high:

> Lo! Thou shalt keep track of things as you take them apart.

Some very simple things will help you follow this decree: grab a pencil and paper and take some notes, make a sketch, or take a photo. They say a picture is worth a 1,000 words: one can save you countless man-hours when it's time to reassemble. Please, grab your camera and take some snapshots before everything is in pieces. On even the simplest of machines, there's just too much minutiae to ever contemplate memorizing everything. Even if you have a general sense of where things go, there are a lot of details that could potentially be important: positions of dials, gears, levers, fluid levels, etc. Also, keep in mind that sometimes a fix may take a long time to materialize. You might be able to

remember what the machine looked like *yesterday*, but you might not be able to complete your repair in such a short time frame. If you have to order parts that take weeks to arrive or the repair is far down on your priority list, there's the possibility it may be a long time before reassembly. Notes and photos will ensure you can easily pick up where you left off.

If you're working on a digital device, the equivalent to a camera is backup software. Use these tools to take a "snapshot" of a system before you start messing with it: should your efforts go awry, you can later return it exactly to its previous state by restoring the backup. There's really no equivalent in the mechanical world, so take full advantage of this difference!

On a related note, if you're contemplating taking something *completely* apart, you might want to stop and ask yourself: "Why am I doing this?" It's a gross violation of the "change just one thing" principle to start a troubleshooting exercise by creating a large pile of pieces on the workshop floor. If you're vigilant about making only one modification at a time, rigorous documentation won't need to be a priority (like it would be in a complete teardown). Adherence to the "one thing at a time" rule means you're always just a single step away from how you found things.

Paper and pencil: low-tech, but important, tools for troubleshooting.
(image: Brendan DeBrincat / CC BY 2.0)

The "Way It Was" May Be WRONG!

Let's say you're troubleshooting a problem, painstakingly keeping track of how the components fit together: their order, relative position, etc. You do this because you know it's immensely easier to put something back together if you remember what it looked like before you took it apart. You deftly wield your weapons: a camera, pen, and paper. You take photos, write notes, draw sketches, and perhaps even talk into a voice recorder where you describe what you're doing as you dismantle a complicated piece of machinery. Well done, Troubleshooter.

However, be prepared for the following: you think you've discovered and isolated the problem, replaced the failing part, and put everything back exactly the way it was. It appears to work for a while but then, the problem reoccurs! You scratch your head, replace the suspect component(s) again, and it breaks down again. Reading the manual, you discover that the previous person who serviced the machine put it back together wrong. Consequently, all your meticulous work to document the machine's state had the effect of preserving the errors. Remember: **working or not, how you find a machine is not the same as its ideal state.** In some cases, it can be a long way off from "ideal." Who knows how many well-meaning but uninformed people touched it before you? Their repairs or modifications may or may not have been up to snuff. They may have put the machine back together in such a way that it will "work," but not in all circumstances or in a degraded manner that lessens the life of its component parts.

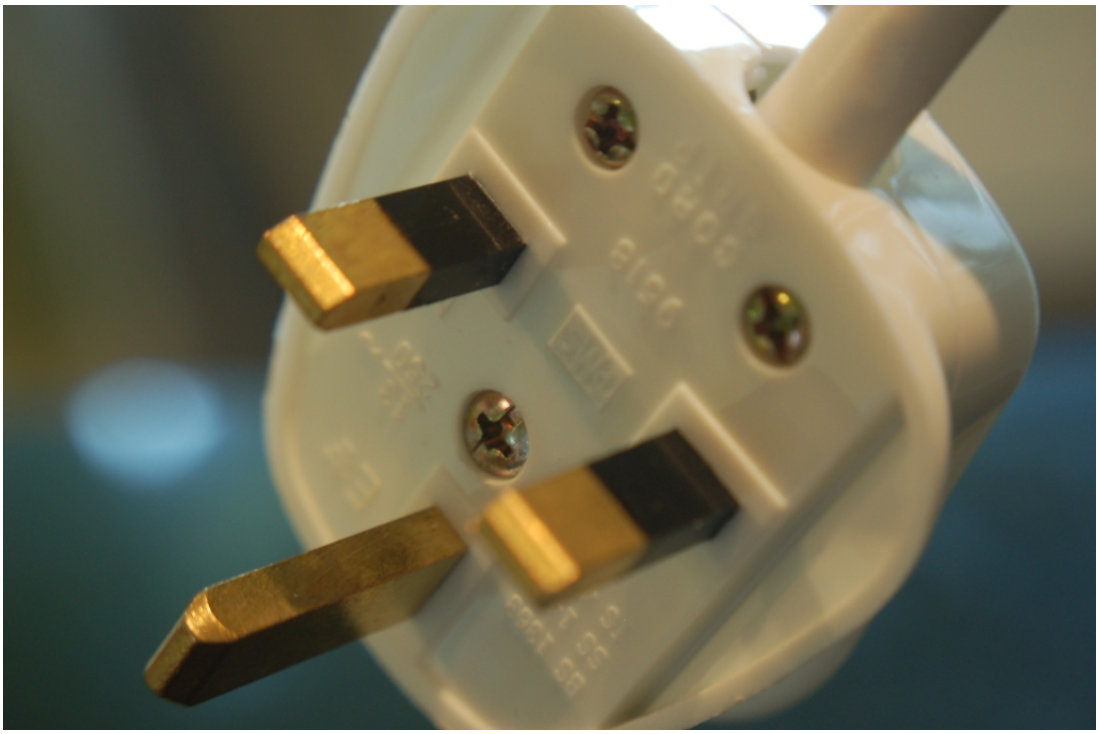
Automated diagnostics are a great way of catching the Some-Idiot-Put-It-Back-Together-Wrong type of problem (but remember, sometimes that idiot might be *you*!). Maintenance logs can tell you who touched a machine last. If those aren't available, I would recommend simply being aware of the environment: do a visual scan of the machine and the general area around a system before starting. Does anything look out of place or different from what you're used to seeing? Even better, ask the operator/on-site guru about its repair history. If you see something that doesn't make sense, let the possibility enter your mind that the machine has not been put back together correctly.

When working on one of these, improvising is discouraged (and probably illegal).
(image: John Murphy / CC BY-ND 2.0)

Some industries have adopted practices to prevent sub-standard repairs and deviations from the ideal system specification. In aviation, the Type Certificate regime is very strict about what kind of parts and procedures can be used to repair a particular airplane. In fact, certain repairs may be mandatory! Imagine getting a notice saying that you must repair your refrigerator by a certain date and in a certain way or it can no longer legally be used to chill food. That would be overkill for a refrigerator, but in the high-stakes world of aviation some repairs can't wait and must be done a specific way. Mechanics who work on Type Certified aircraft are required to maintain the ideal "type" and are not allowed to deviate from it. The idea being that clever "hacks" might end up making an airplane unsafe and kill you.

Of course, adhering to the Type specifications of an aircraft doesn't necessarily mean it's safer in all circumstances, only that it's been scrutinized and documented. This strictness has a cost: there may be only *one* legal way to fix to a particular problem. Novel, cost-saving repairs might take a long time to be approved because of the rigorous review process. Even if you work under a heavily regulated maintenance regime like the aviation industry, you still need to be on the lookout for "the way it is, is wrong." Regulations in place or not, reality always has the last word on whether or not a previous repair was completed correctly.

Lastly, parts and connectors that will only fit one way or safeguards that will only allow manufacturer approved parts are yet more ways to prevent a machine from deviating from its ideal state over a lifetime of repair and regular maintenance (these are sometimes a revenue generating ploy too, see: printer ink).



Parts that only fit one way help prevent repair errors. Alas, not everything will be as well-designed as this plug. Components installed in the wrong place, or in the wrong way, are just another way a previous repair may have been botched.

(image: jkfid / CC BY 2.0)

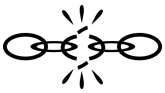
References:

- Header image: *"Camera components"*. Vadim Sherbakov, photographer. Retrieved from Unsplash, https://unsplash.com/photos/osSryggkso4.

The Way It Is And The Way It Was was originally published January 31, 2012.

Notes:

Is It Plugged In?



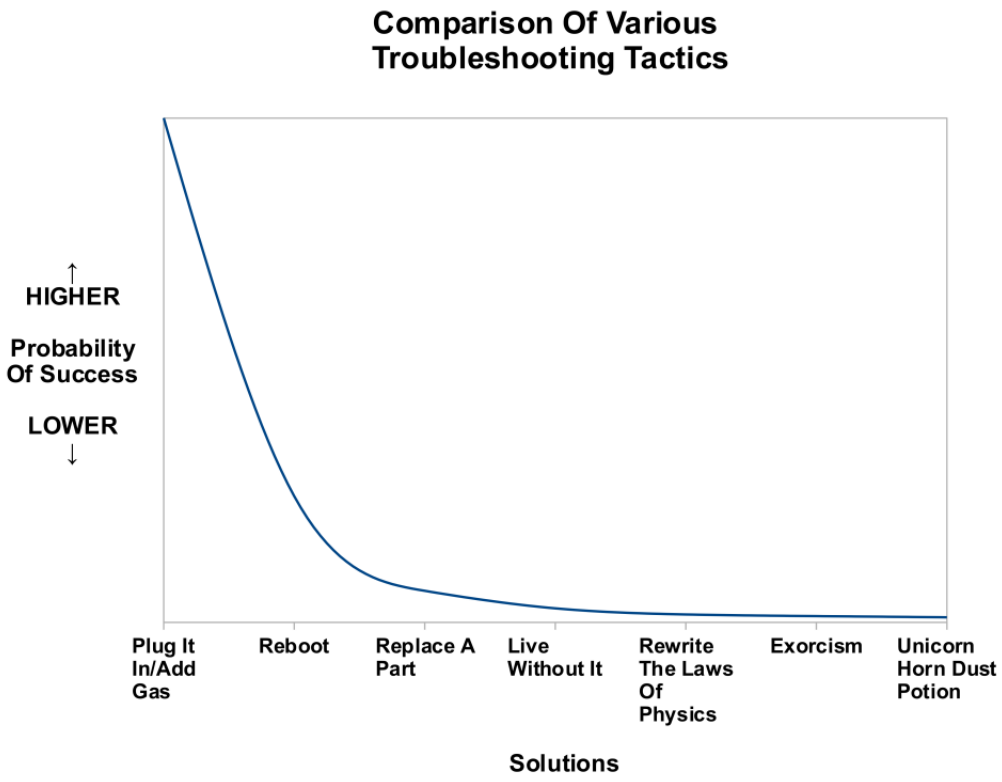
> I check the obvious things first, then progressively go deeper and deeper.
>
> **Rich Kral**

More often than not, when I would mention to someone that I was working on a book about troubleshooting, their eyes would light up and they'd blurt out:

> Oh, I know about that! "Is it plugged in?" Right?!

This kept happening. Soon, I realized that I was writing about a topic with universal appeal. Machines are everywhere in modern life and we rely on them for so many things. Who hasn't had a "I forgot to plug it in" moment in their life?

Humans are natural troubleshooters; anyone who is more than a few years old is experienced in the art. Think of all the things, both large and small, that you've fixed in your lifetime. Another aspect I liked about people shouting out "Is it plugged in?" was the implication that the cause of most failures was something simple. Experience says they're right! If we were to scientifically graph how most problems are fixed, I think we would find:



Graph: a rigorous, scientific comparison of a spectrum of troubleshooting tactics by probability of success.
(image: © Jason Maxham)

One of the goals of my writing is to acquaint you with the full range of possibilities for failures and solutions. Who knows, maybe some day you will encounter a problem that requires (metaphorical) Unicorn Horn Dust. I've definitely seen some weird things in the troubleshooting trenches. However, experience says you'll probably be spending most of your time on the left hand side of the graph, in the midst of solutions like plugging things in, adding gas, and rebooting. The seasoned professional troubleshooter will benefit from a periodic reminder of how often the solution is something *simple*. That's me: my mind loves to entertain fantastic and novel possibilities that fit the data. On the other hand, those new to the game will benefit from having their horizons opened to the possibilities of subtle and complicated failure scenarios.

Prerequisites For Operation

Given its popularity, I was bound to address troubleshooting's most famous question: **"Is it plugged in?"** The question, and brilliant associated solution: "Plug it in!," is an example of a broader principle called **"prerequisites for operation."** Every system has a context in which it functions. This includes the conditions required for it to work: the list of everything that has to be "right" and present (as well as absent conditions, as you'll see later). This list can be very long and we usually aren't aware of just how many things are needed for a system to operate. Frequently, it's only when things break down that we first become aware of the dependencies implicit in the use of a given machine.

Take a gas-powered motorcycle and consider for a moment some of these hidden prerequisites. One dependency you may never have considered is: oxygen. That's right, your typical petroleum-powered engine relies on having this atmospheric gas present in abundance to mix with fuel for combustion. If you shipped your Ducati to the Moon, it's not going to work. That's okay, you can still lean up against it, smoking a cigarette, trying to look cool. Oh no, cigarettes won't work there either! Also, imagine if there were no gas stations. That would severely limit your commuting and road trip options, right? You've probably never thought about it until now, but an abundance of oxygen and gas stations allow a motorcycle to operate and be useful.

Beautiful, but it will not run on the Moon.
(image: Tinou Bao / CC-BY-2.0)

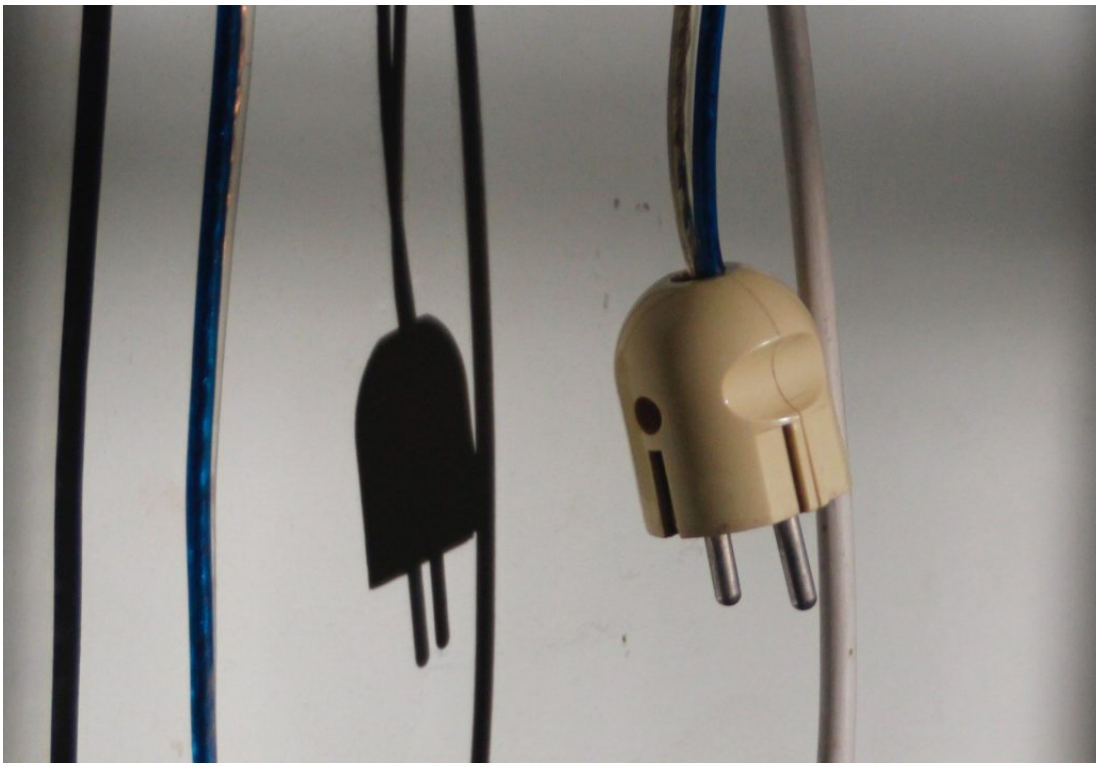
Oxygen and gas stations are ubiquitous over most of the world (at least the part where you'd typically drive a motorcycle), so you're probably not going to find them listed as "needed" in the owner's manual. My point is that the missing dependency that is causing a failure may not be documented. This is especially true for any custom-made system, where you can create an entirely unique web of dependencies from the parts you cobble together (e.g., a server farm or an oil refinery). In "The Order of Things" I outlined a strategy for reducing complexity by disabling subsystems. This is the other side of the coin: there may be a *minimum* level of complexity needed for the system to function!

Do I Need Any Others When I've Got This One?

At first glance, the strategy of fulfilling prerequisites seems like a promising candidate for The One. I'm talking about a singular strategy that you can use to solve *any* troubleshooting problem. Indeed, you can imagine having a List of Everything That Must Be Right, Present and Absent for a particular system. You could then methodically go through this list, line-by-line, to make the machine conform to the specification. It would be beautiful: there would no longer be any mystery to the troubleshooting process and fixing things would always be a certainty. Great! I'll just put my feet up and take a nap while you write up that list…

Done yet? Unfortunately, such a document can't exist as it would have to cover an infinite number of possibilities that couldn't possibly be anticipated. Ducatis on the Moon are just the tip of the iceberg for how machines will get deployed in scenarios that designers, engineers, and technical writers can't anticipate.

The "must be absent" portion of the list is where the problem of describing operating conditions becomes truly infinite. You may have seen an example of **negative prerequisites** in an operator's manual when it says something like: "not to be immersed in water." That's right, in addition to things that are *required*, many devices need things to be *absent* to operate properly. Will a particular machine work when submerged in water or covered in peanut butter or placed in a sealed room filled with helium or launched into outer space or heated to 1000° C…? Each of these operational scenarios could be individually tested and documented (some at great expense) but you can begin to see that it would be impossible to cover them all in our ideal troubleshooting guide.



Sometimes, this isn't where it should be…
(image: Fajrina Adella / Unsplash)

Back To Reality

Beyond the theoretical problems described above, where does the "prerequisites for operation" strategy rank in the real world? My experience is that the effectiveness of this strategy has a big front-loaded payoff (when it works). Up front, merely asking the question "What is needed for this machine to operate?" will uncover things like unplugged power plugs and empty gas tanks. Also, the manufacturer may have given you a list of prerequisites in their product documentation. Given their much deeper experience with a machine through the phases of design, testing, and support, a manufacturer's guidance on matters of prerequisites are usually solid gold.

After that, the efficient discovery of information about failures seems best left to other strategies, versus trying to dream up Everything That Must Be Right, Present and Absent. Life's too short to go that route!

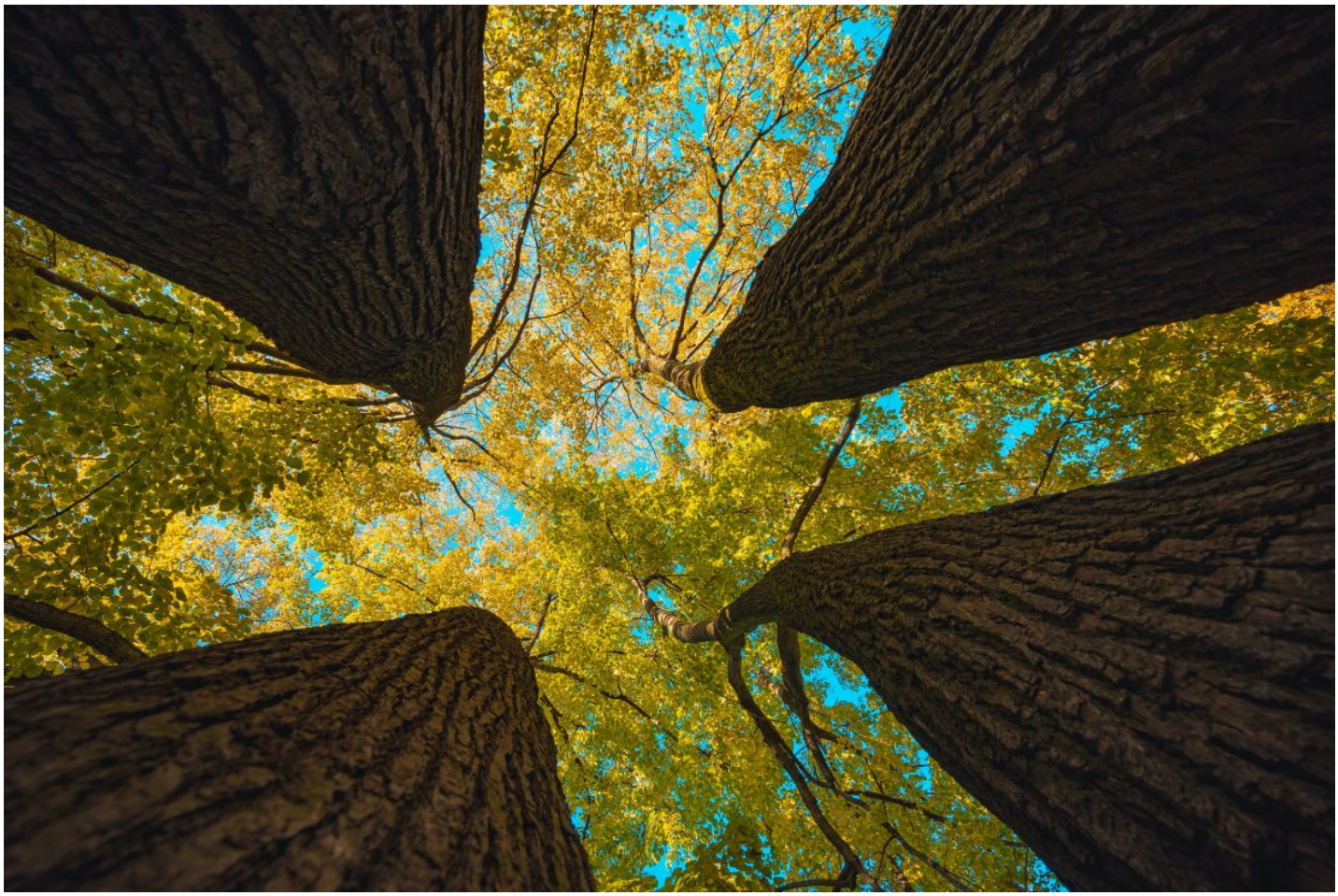
References:

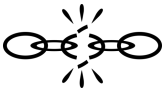
- Header image: *"Guitar wire in black and white"*. Juan Di Nella, photographer. Retrieved from Unsplash, https://unsplash.com/photos/OUgLA2unwtg.

Is It Plugged In? was originally published February 14, 2012.

Notes:

A Different Point Of View





> A fresh set of eyes. That's all it took.
>
> **Jamie Karrick**

If you're buried deep in a troubleshooting problem, it's easy to get stuck in a rut. Since we can sometimes talk ourselves into confusing pretzels, my guide to skillful questioning will help cut through the muddy words often used to describe problems. If it's a limiting belief expressed through language that's holding your investigation back, those techniques will help you detect and overcome it. While the language patterns are powerful and can clear certain types of roadblocks, they won't always be enough to get you past a stuck point. Sometimes, the description of a problem will be clear and yet a solution will continue to be elusive.

To get over the hump, consider soliciting an outside perspective. Even better, multiple outside perspectives. Here "outside" simply means "not you": that's right, anyone that isn't you (or on your team, if you're troubleshooting with a group) can be considered. Asking other people can snap you out of the trance you're in and help you see things you may be missing.

When you're stuck, seek a new perspective.
(image: Ravi Patel / Unsplash)

When considering the benefits of other perspectives, Eric Raymond's "Linus' Law" comes to mind:

> Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone. Or, less formally, "Given enough eyeballs, all bugs are shallow."
>
> **Eric Raymond, The Cathedral and the Bazaar** [1]

Raymond is talking about the world of software development, but I've found the principle to be universal. My own formulation of the "many eyeballs" phenomenon is:

> For every problem, there is someone who will make solving it look (relatively) easy.
>
> **Jason Maxham, The Art Of Troubleshooting**

So, that's what it feels like to quote yourself. It wasn't as good as what I had built up in my mind… Anyway, the "relatively easy" clause is important: just because you put your dilemma in front of lots of people doesn't mean someone will magically come up with a brilliant quick fix. Think of all the thousands of scientists who have battled cancer, and yet a cure still eludes humanity. The benefit of "many eyeballs" is extracting the most from the experiences of others and generating new possibilities for solutions. That process may not yield a definitive answer, but hopefully you'll be pointed in the right direction.

There are five types of relationships, each with different benefits, which will generate useful feedback:

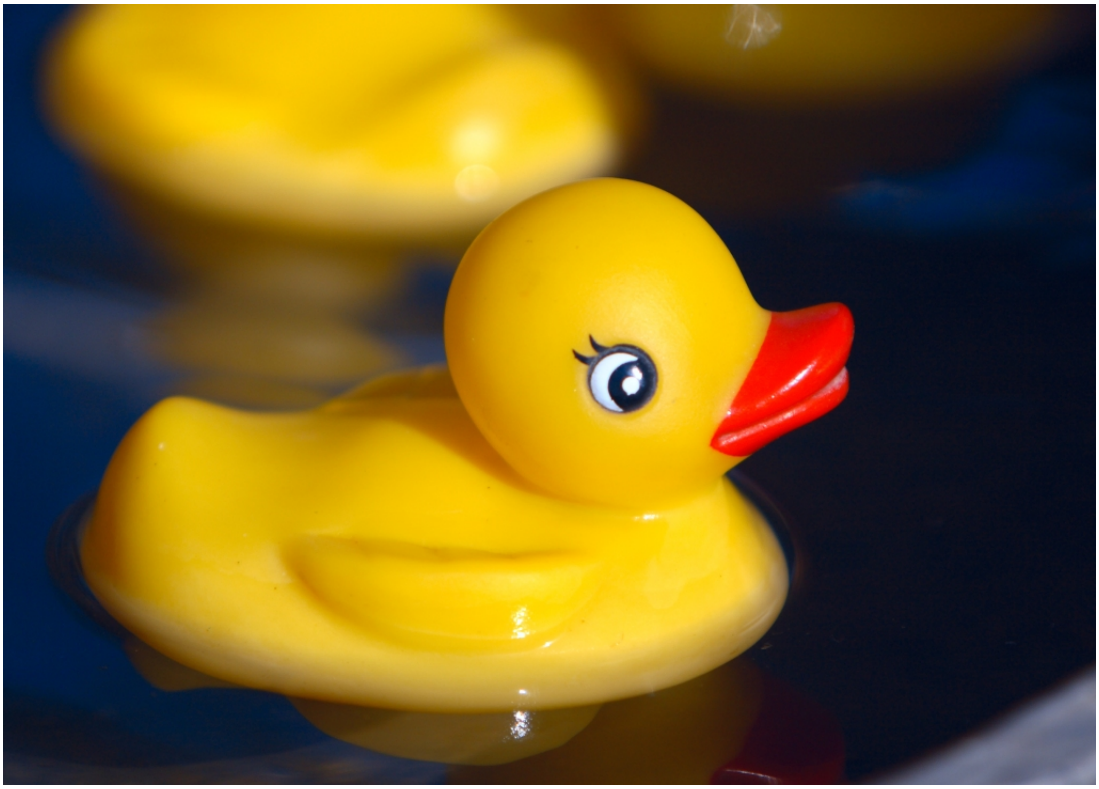
Graphic: tradeoffs when soliciting different points of view.
(image: © Jason Maxham)

1. You

That's right, you're at the center of the circle—it's likely that no one knows more about the problem you're working on than you. Have you gotten the most out what's in your head? This section is about getting a new perspective, but you might not have considered that a different point of view can come from—within you! Have you ever talked to one of these?

It's okay, your secret is safe with me and so let me introduce the "Rubber Duck" method. This is where you explain your troubleshooting problem to an inanimate object: a rubber duck, pencil, the ceiling, etc. Programmers and IT professionals have long used this technique to resolve issues:

> Another effective technique is to explain your code to someone else. This will often cause you to explain the bug to yourself. Sometimes it takes no more than a few sentences, followed by an embarrassed "Never mind, I see what's wrong. Sorry to bother you." This works remarkably well; you can even use non-programmers as listeners. One university computer center kept a teddy bear near the help desk. Students with mysterious bugs were required to explain them to the bear before they could speak to a human counselor.
>
> **Brian W. Kernighan, The Practice of Programming** [2]



A great conversation partner.
(image: longhorndave / CC BY 2.0)

Something special happens when you're forced to verbalize and explain your problem to someone else. "Rubber Ducking" proves that you don't even need another person to get a different perspective: often a quick conversation with *yourself* is all that's required.

2. Your Peers, Subordinates, And Managers

Besides yourself, these people are most likely to understand the details of and be sympathetic to the troubleshooting you're doing. It might even be part of their job description to help you out! What you're hoping is for them to say "That failed on me last week too. I fixed it by doing this…" Even if they haven't encountered your problem and they don't have a quick fix, plow on and ask them: "How would you solve this problem?" Since they are in your organization, they will be familiar with the resources (people, tools, etc.) available to solve the problem. For that reason, the questions they ask and suggestions they give might actually be useful.

Ironically, it's even better if they're a little bit surly about the whole thing, because it might nudge you to reconsider things you missed. Here, we can take a lesson from Kevin Dunbar's research into how scientific breakthroughs are made (taken from an article in *Wired* on the benefits of failure):

> Dunbar found that most new scientific ideas emerged from lab meetings, those weekly sessions in which people publicly present their data. Interestingly, the most important element of the lab meeting wasn't the presentation — it was the debate that followed. Dunbar observed that the skeptical (and sometimes heated) questions asked during a group session frequently triggered breakthroughs, as the scientists were forced to reconsider data they'd previously ignored. The new theory was a product of spontaneous conversation, not solitude; a single bracing query was enough to turn scientists into temporary outsiders, able to look anew at their own work.
>
> **Jonah Lehrer, "Accept Defeat"** [3]

I've personally observed the power of well-intentioned pushback while investigating. So, go forth and solicit questions and comments, "stupid" or otherwise, from your colleagues!

3. People In Your Field/Industry/Occupation

One stage removed from people in your organization are people in your industry or who share your occupation. They might be working at a competitor or have retired long ago. Included in this category are former co-workers, bosses, mentors, teachers, professors, etc. These contacts can be really valuable for troubleshooting because they will be used to solving similar problems using similar tools. However, because they have been at a different organization (or worked in a different era), they might do things…differently. That's great, because seeking out what's "different" is what this section is all about. If they're working at a competitor, they might not share their secret formula with you, but you'd be surprised at the amount of help available to you in a crisis, if you just ask.

Included in this level are technical contacts at your vendors, who typically will have worked with others in your industry and will therefore have a similar breadth of knowledge. Also, because they "work for you," they'll be motivated to help you out and may be less guarded about sharing information than a direct competitor.

It's important to set up these contacts in advance and maintain these relationships, so think about opportunities to network and build contacts through industry associations, trade groups, conferences, trainings, etc.

4. People In Related Fields/Industries/Occupations And Those Who Use The Same Tools

Still useful, but further out, are people who work in a related field. They will be familiar with the problems you are trying to solve, but at a more abstract level. For example, an auto mechanic and an airplane mechanic will have this type of relationship. People in related fields are great to bounce ideas off because they will likely have *very* different solutions. Given the economics involved they may have fixes that simply aren't feasible for your situation, but there's no downside in asking for their perspective.

People who use your same tools, machines or processes (but work in a completely unrelated field) can also be included in this tier. If you're a toymaker that uses the same type of industrial robots as the automobile manufacturer next door, you've got something very important in common, especially when those robots break down.

5. Everyone Else

Consider presenting your problem to: a doctor, a physicist, a biologist, a psychologist, a mechanic, a programmer, an artist, a musician, a carpenter, an electrician, etc. How would they solve your issue with the resources and strategies with which they are familiar? The answers from this tier will be the most hit-or-miss. However, the payoff can be huge: there's a long and storied history of cross-pollination of ideas from seemingly unrelated fields. For instance, Eugene Stoner took his experience using lightweight materials like aluminum for building aircraft and applied them to firearms to create the iconic M-16 assault rifle. Builder François Hennebique saw how gardener Joseph Monier was using steel reinforced concrete for his planters and it led to the amazing skyscrapers, bridges and dams we see in the world today. Especially for very difficult or chronic troubleshooting problems, you may need the insights of someone in a completely different universe.

You might be saying "There's no way I'm going to ask a musician how to troubleshoot a fuel injection system." Thanks for the great segue:

Just Grab Someone, Anyone!

You can involve someone who knows "nothing" about your problem with a technique I call **"get someone to ask stupid questions."**

If you've hit a wall, go grab someone who you're sure can't help (like someone who works in "HR," "marketing," or "management"). Briefly explain the problem, tell them you're stuck and then say, "Go ahead, ask me some stupid questions to help me figure this out." They'll probably start with something like "Is it plugged in?" (is it?) and so you're probably thinking "How is this even remotely useful?"

Here's what I've learned: it doesn't actually matter what they ask you, but it frequently works to get your mind pointed in a new direction. I've personally come up with breakthroughs in the middle of an "ask stupid questions" session with a co-worker. You've gone over the same facts and solutions in your mind a million times; this exercise breaks through the trance you're in and the mental loops you may be running. Having to answer stupid questions gets you out of your head, and forces you to consider the problem from a different point of view.

This section is all about new perspectives, and having to explain something complicated to someone who knows

nothing about your work will force your mind to consider a new point of view. Even if the perspective your colleague is offering isn't very useful, your subconscious will get the metaphor: there's probably another way to look at the problem. Again, I've found that just a few minutes of "stupid questions" will be enough to get me going in a better direction. Try it!

References:

- Header image: "Four trees from below." Adam Nieścioruk. Retreived from Unsplash, https://unsplash.com/photos/53KPnkKjkAY.
- [1] Eric Raymond, *The Cathedral and the Bazaar*, "Release Early, Release Often."
- [2] Brian W. Kernighan, *The Practice of Programming* (New York: Addison-Wesley Professional, 1999), pg. 123.
- [3] Jonah Lehrer, "Accept Defeat," *Wired*, January, 2010.

A Different Point Of View was originally published February 20, 2012.

Notes:

Same Symptom, Different Causes

> If you fixed one car, you fixed all of them. Still, you don't want it to come back and bite you, because you can have the same symptom and two different problems.
>
> **Dan McCormick**

One of the more confusing things to explain is that a machine is broken "just like last time," but this time the *cause* is entirely new. Trust me, this is equally confusing to the troubleshooter who had to figure it out. I think most people would prefer a simple "*only* A causes B" explanation that is consistent over time. But, just like in life, you can end up in the same place again, even though you took a different path to get there.

One Or More

If you troubleshoot long enough, you'll eventually experience systems with *multiple* internal failures. However, even though there may be several things that need to be remedied, this kind of situation often manifests itself in just a *single* external symptom. A car with a dead battery *and* an empty fuel tank will not work just like a car with a dead battery *or* an empty fuel tank. This gets back to our typical experience of failures, where surface-level symptoms and what we are prevented from accomplishing looms the largest in our minds.

Let's work through an example and think deeper about situations involving multiple failures. You have a string of 3 red lights, which you have deployed in hopes of creating a festive atmosphere for a party. The string of lights is wired such that all of the lights must individually work for the string to be lit (i.e., if one goes out, the whole string goes out).

You've had them up in your dorm room, but after your all-weekend rager, they no longer work.

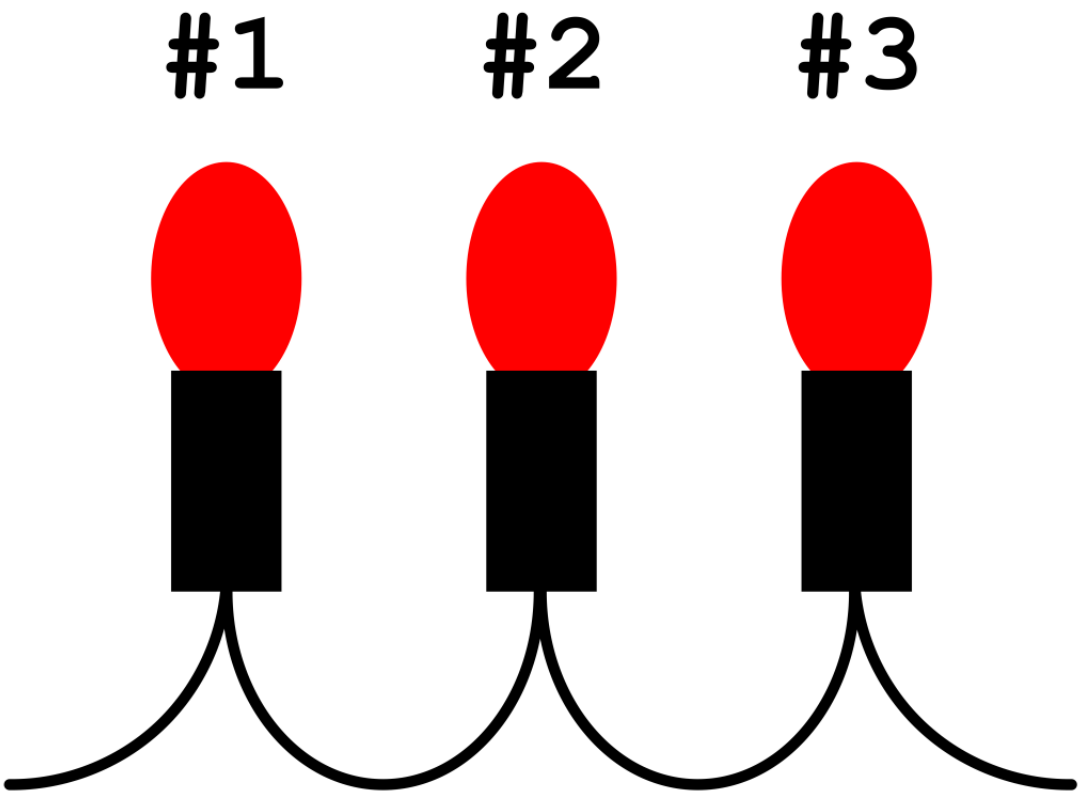


Diagram: a string of 3 lights, trusty enablers of a party ambiance.
(image: © Jason Maxham)

After you've nursed your hangover, you want to show off your troubleshooting skills to your roommate. Consequently, you decide to use a strategy of serially replacing each of the lights with a spare. You'll swap each lightbulb, one at a time, retesting to see if the string works after each swap. If swapping a specific lightbulb doesn't work, you'll put the string back the way you found it and move on to the next light.

As your omniscient narrator, I'll tell you that the state of the broken system is thus:

- **BULB #1:** FAILED
- **BULB #2:** FAILED
- **BULB #3:** WORKING

As you can see, a serial swap, test, and reset strategy won't bring this system back to a working state because there are *multiple* lightbulb failures. Of course, you don't know that and so you grab a spare bulb and begin the swapping process:

1. You swap out BULB #1 with the spare. Test. The lights don't work (#2 is also failed), so you put the original BULB #1 back in.
2. You swap out BULB #2 with the spare. Test. The lights don't work (#1 is also failed), so you put the original BULB #2 back in.
3. You swap out BULB #3 with the spare. Test. The lights don't work (#1 and #2 are failed), so you put the original BULB #3 back in.

You would be scratching your head at this point, but go where logic leads you: the problem is not likely a *single* failed bulb. After your swapping exercise, the remaining possibilities are:

- There are multiple failed lightbulbs on the string.
- The spare bulb is faulty. By the way, this is the reason you should always use "known working parts" as replacements: swapping a working part with a broken one increases the number of failures to be discovered and remedied. Unless you like making more work for yourself!

- The problem with the string of lights lies elsewhere, like maybe the outlet you're plugging into doesn't have electricity.

One way to get clarity on the state of the bulbs is to reverse the process and take them out of the *failed* string and place them into another *working* string. Putting BULB #1 into a working string will cause that string to fail, showing you that BULB #1 is defective. You can test all of the bulbs (including the spare) this way: it's a play right out of "Copy One That Works."

Running The Numbers

Given that troubleshooting is about playing the odds, what can we learn about scenarios involving multiple failures? Are they likely and is it worth looking for them? Let's try to get a sense of the probabilities involved. We'll stick with our example of that string of stylish red lights, starting by listing all of the possible states of these 3 bulbs:

Scenario #	Bulb Status			Total Failures	Overall Status
	Bulb #1	Bulb #2	Bulb #3		
1	OK	OK	OK	0	OK
2	OK	OK	FAULT	1	FAULT
3	OK	FAULT	OK	1	FAULT
5	FAULT	OK	OK	1	FAULT
4	OK	FAULT	FAULT	2	FAULT
6	FAULT	OK	FAULT	2	FAULT
7	FAULT	FAULT	OK	2	FAULT
8	FAULT	FAULT	FAULT	3	FAULT

You can see there are 8 unique possibilities for how these 3 bulbs can be functioning or broken. We said this particular system requires all 3 bulbs to be working for the system as a whole to operate. Given that, note that **only one** of these 8 possibilities will result in a working string of lights! Wow. One way to be right and 7 ways to be wrong.

This is a great illustration of a fascinating troubleshooting principle: while there are typically an infinite number of ways for a machine to be screwed up, there's often only a few ways for it to be right. This is yet another buttress to the "change just one thing at a time" principle and a reminder of why a bias for minimalism should be guiding your repairs. The more mucking around you do, the greater the chances you will be adding to that infinite realm of possible defects.

Next, let's tally up how many of the above scenarios have 0, 1, 2, or 3 total failures:

Fault Count	# Scenarios	% of All Scenarios	Overall Status
0	1	12.5%	OK
1	3	37.5%	FAULT
2	3	37.5%	FAULT
3	1	12.5%	FAULT

The tally line with a fault count of "0" is the working state, which as stated above is only 1/8th (12.50%) of the total number of possibilities. The next line, with a fault count of "1," are our singular failures (37.50% of all possibilities). The remaining two entries (marked "2" and "3") are our *multiple* failure scenarios: you can see that they represent 50% (37.50% + 12.50%) of the possibilities. If each of these failure scenarios was equally likely, we wouldn't be having parties very often. Can you imagine buying a machine that only worked 12.5% of the time?!

Don't get discouraged, most machines will seemingly favor non-operation when going by a raw count like in the table above. However, after weighting the scenarios by their actual likelihood of failure, we find that they are not all equally

likely. Continuing with our example, if we assume that each of these bulbs has a 1 in 10,000 chance of failing (per time unit in use), the likelihood of failures involving various combinations of bulbs is much different:

Scenario #	State %			Combined Probability	Overall Status
	Bulb #1	Bulb #2	Bulb #3		
1	99.99%	99.99%	99.99%	99.9700029999%	OK
2	99.99%	99.99%	0.01%	0.0099980001%	FAULT
3	99.99%	0.01%	99.99%	0.0099980001%	FAULT
5	0.01%	99.99%	99.99%	0.0099980001%	FAULT
4	99.99%	0.01%	0.01%	0.0000009999%	FAULT
6	0.01%	99.99%	0.01%	0.0000009999%	FAULT
7	0.01%	0.01%	99.99%	0.0000009999%	FAULT
8	0.01%	0.01%	0.01%	0.0000000001%	FAULT

Just like before, we can combine the probabilities to get a sense of the relative chance of single versus multiple failures:

Fault Count	# Scenarios	Combined Probability	Overall Status
0	1	99.9700029999%	OK
1	3	0.0299940003%	FAULT
2	3	0.0000029997%	FAULT
3	1	0.0000000001%	FAULT

If a bulb failure was the only thing that could be wrong with our fabulous string of lights, we can see it will be operational most of the time (99.97%). Also, note that the single failure scenario (fault count = 1) is by far the most likely among the various faults listed. By how much? Well, let's calculate the ratio between the single and multiple failure scenarios to find out. We take the probability of a single failure (0.0299940003%) and divide it by the sum of the "2" and "3" tally lines (0.0000029997% + 0.0000000001% = 0.0000029998%):

0.0299940003% ÷ 0.0000029998% = **9998.66**

This result shows the chance of encountering a *single* failure is nearly 10,000 times more likely than all of the *multiple* failure scenarios combined! Now that is actionable intelligence. Given the huge drop-off in probability from one to two failures (and then again from two to three failures) you can see it wouldn't make sense to test all the bulbs if the test was time-consuming or expensive. The odds favor stopping and retesting after identifying that *first* failed bulb.

Back to our example, the failure of the serial replacement strategy opened the door to the possibility of two or more bad bulbs. However, as soon as you had identified a second failed bulb, it would be wise to stop and retest. Given the truly unlikely scenario of 3 burnt-out bulbs, it doesn't make sense to pursue it without additional evidence.

A Lack Of Independence

We've made one key assumption in the calculation of the statistics above: that a lightbulb burning out is an independent event. That is, if one light bulb burns out it doesn't affect the probability of another doing the same. When this is true, it leads to the statistics above favoring the single failure. Troubleshooting in the real world, I caution you against making this assumption without evidence. The machines that fill our world are deeply interconnected systems, full of dependencies and linkages, both within themselves and to the larger context in which they are used.

Whether you arrived from the right or the left, you still ended up in the same place. Multiple paths, one destination, just like some troubleshooting problems you'll encounter.
(image: Lance Fuller)

Even for the simple case of a string of lights, it's easy to think of instances where multiple, coincident lightbulb outages are not independent events: a tripped breaker, an electrical surge blowing the bulbs, a frayed wire in the plug leading to an open circuit, a manufacturing defect in this particular batch of bulbs, etc. I keep stressing the importance of context when troubleshooting, because it allows you to correctly classify a problem and choose the most efficient strategy. Encountering highly improbable multiple failures is a signal that you may have missed an important shared connection: these are opportune moments to step back and consider systemic causes (and solutions).

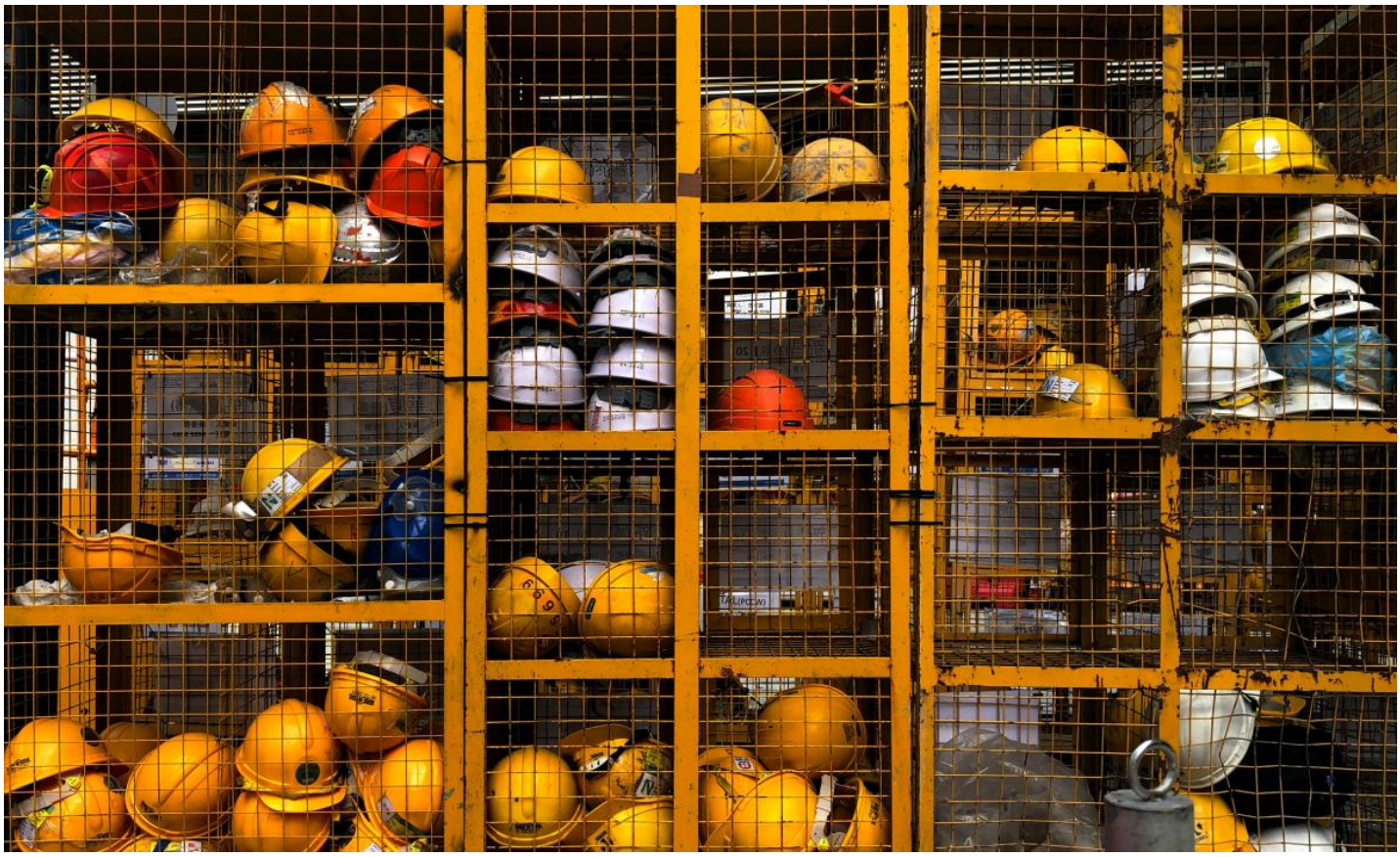
References:

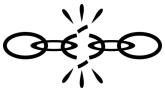
- Header image: Hush Naidoo, photographer. Retrieved from Unsplash, https://unsplash.com/photos/yo01Z-9HQAw.

Same Symptom, Different Causes was originally published March 13, 2012.

Notes:

Improving the Environment





> Frustration is actually a productive emotion. It's there because there's something is in your way... Being uncomfortable can help with that.
>
> **Alex Chaffee**

You probably thought this was going to be about old-growth forests, endangered species, global warming or water rights. However, in the context of troubleshooting, when I say "improve the environment," I mean *your* environment!

File under "obvious," but upgrading the conditions under which you troubleshoot can significantly improve your chances of finding a timely solution. Vice versa, an inhospitable or distracting environment can work against you and significantly delay finding a fix. First, let's observe that many machines are in locations not among the most comfortable on earth. For example, during my time in the IT world, I spent my fair share of time in colocation facilities (aka, "colos"). These are places where companies buy or rent space to house their computers. Colos are spartan and austerely beautiful places that are designed for one purpose: to keep computers running smoothly. Along that front, they have abundant resources: backup power generators, redundant Internet connections and enough air conditioning to recreate Arctic conditions. I guess you could say that everything has its ideal setting: a rattlesnake is probably happiest while sunning itself on a rock in the desert, a polar bear prefers an iceberg, and a frog on a lily pad in a swampy pond feels quite content. Likewise, I always imagined my computers smiling from ear-to-ear when I placed them in a well-equipped colo. Uninterruptible power, ample cooling, and a fast network: what else could a server want?

You probably don't want to linger here. Many industrial locations are built for machines, not humans.
(image: Alfred T. Palmer / Library of Congress)

What a server wants is probably different from your own desires. What are colos and other industrial environments typically **not** designed for? That's right, humans. When you're in a giant room filled wall-to-wall with servers, your ears will be violated by the deafening whoosh of thousands of cooling fans. The low-frequency electrical hum from all that power being transformed into computing cycles will start to lull you to sleep. If you're working anywhere near the A/C vents, cold air will blast you and remind you to put your jacket back on. Alternatively, if you're standing behind a rack of servers expelling the heat from hundreds of CPUs and disk drives, you will feel like you're on a tropical island and wish you had worn shorts. Oh, and there's no comfortable place to sit either. Personally, in a setting like this, I feel my problem-solving skills are degraded because my mind is distracted by these environmental factors. For these reasons, I strongly disliked troubleshooting at the colo and my subconscious agenda was always to get in and get out as soon as possible.

Here are some environmental conditions that will impact your troubleshooting abilities:

- **Light:** of course, you're better off if you can clearly see what you're working on. If you can't use artificial light and it's night, sometimes it's best to wait for day.
- **Sound:** noisy environments are fatiguing and will make communication difficult if you're troubleshooting with a team (or taking instructions over the phone).
- **Space:** the area around a broken system must be large enough to accommodate your team and equipment.
- **Temperature:** extremes of hot or cold will negatively affect your physiology, making repairs more challenging. These type of conditions can also be hard on your tools.
- **The Elements:** fixing things while exposed to the wind, rain, or sun will require Nature's cooperation.
- **Safety:** troubleshooting amidst hazards, man-made or natural, can be very dangerous. Examples; working on a car while it's stuck in the middle lane of a busy highway, or perched near the edge of a cliff.
- **Business Requirements:** the need of a firm to continue operating normally may constrain when or how you work. When repairs will be a hindrance, because of the noise, mess, or interruption, be prepared to work after the close of business.
- **Scrutiny:** the need to maintain a professional demeanor may clash with your ability to efficiently get the job done. This tension is most often felt when you're forced to work in close contact with a customer. You might encounter a personality conflict, perhaps with a client who likes to micromanage. Likewise, a repair might require something experimental or messy that, to the untrained eye, may cause alarm. People may not understand that

troubleshooting requires improvisation, thoughtful pauses to reflect, and frequent course corrections. Put another way, there's a good reason that sausage is made behind closed doors.

- **Time:** an important dimension that will affect all of the conditions listed above. Daylight, temperature, tides, weather patterns, sleepiness, fatigue, crowds, etc. will all ebb and flow with the passage of time. You will need to deal with environmental factors the right way **and** at the right time.

Must we troubleshoot out in this?
(image: Donna B. Cooper / CC BY-ND 2.0)

Given that the environment will either help or hinder troubleshooting, you'll always want to be on the lookout for opportunities to:

- Make the current environment more hospitable.
- Change venues and continue working elsewhere.

Please Turn On The Lights

If you're troubleshooting at the site of a failure, your ability to modify the environment to suit your needs may be limited. Even so, you can typically make some improvements. I can't tell you the number of times I have found a group of engineers squinting at a problem in a darkened room. Oh, the look of wonder and awe in their eyes when I would triumphantly flip the light switch on, exclaiming "Let there be light!" Sometimes, I would say it in Latin ("Fiat Lux!") and the look they'd give me was, "Who's this crazy guy turning on lights and shouting things in Latin?"

In Latin or English, turning on the lights so you can actually see what you're doing might have the highest improvement-to-effort ratio when it comes to upgrading your conditions. Others will typically involve a tradeoff (usually time or money). Sometimes, it might be very difficult to make even trivial changes to the place where you're troubleshooting. What if turning on the lights involves something complicated like finding someone who can operate the automated building management system (like in a stadium)? For those cases, you'd probably wish you had brought along your flashlight!

Some equipment isn't very portable.
(image: Haakman / CC BY 3.0)

Where The Grass Is Greener

When circumstances favor relocating to a place where repairs will be easier, faster, cheaper, etc., the factors will typically involve:

- The need for specialized or cumbersome equipment that can't be brought on-site.
- Repairing at the scene would either be unsafe, uncomfortable, or disrupt business with your presence.
- Downtime is so costly that repairs can't be done *in situ*. The faulty piece of equipment must be removed immediately so that a spare can be deployed.
- Better scheduling: repairing on-site may constrain you to the operating hours of the business.
- Stepping out of the spotlight: it's always better to be able to do your work (and make those inevitable mistakes) without having a customer breathe down your neck.

Again, the decision to relocate will involve tradeoffs and have an economic component. Clearly, if you knew in advance that a fix could be completed in just a few minutes, then bringing a machine back to your workshop an hour away would be a waste of time. On the other hand, if it took a 3 weeks to find the answer, you'd probably be wishing on the third day that you had changed venues sooner. For this reason, you can't say "Never relocate!" or "Always relocate!" I am only here to remind you of the possible *benefits* of changing locations to your workshop.

For me, the benefit of a venue change was both physiological and psychological. It was cold, loud, and lonely in the colo. I couldn't throw Toto on the stereo, make some tea to help me think, or easily bounce an idea off a fellow co-worker. However, I never let my desire for creature comforts get the best of me. Remember, when you retreat with a broken machine to your deluxe, climate-controlled workshop you must eventually return to the same place with a fix. Don't use a venue change as a stalling tactic. Furthermore, while moving a machine may make you more comfortable, it also might be a colossal waste. Time should be among your most precious resources, and conserving it will sometimes mean standing and fighting. On-site.

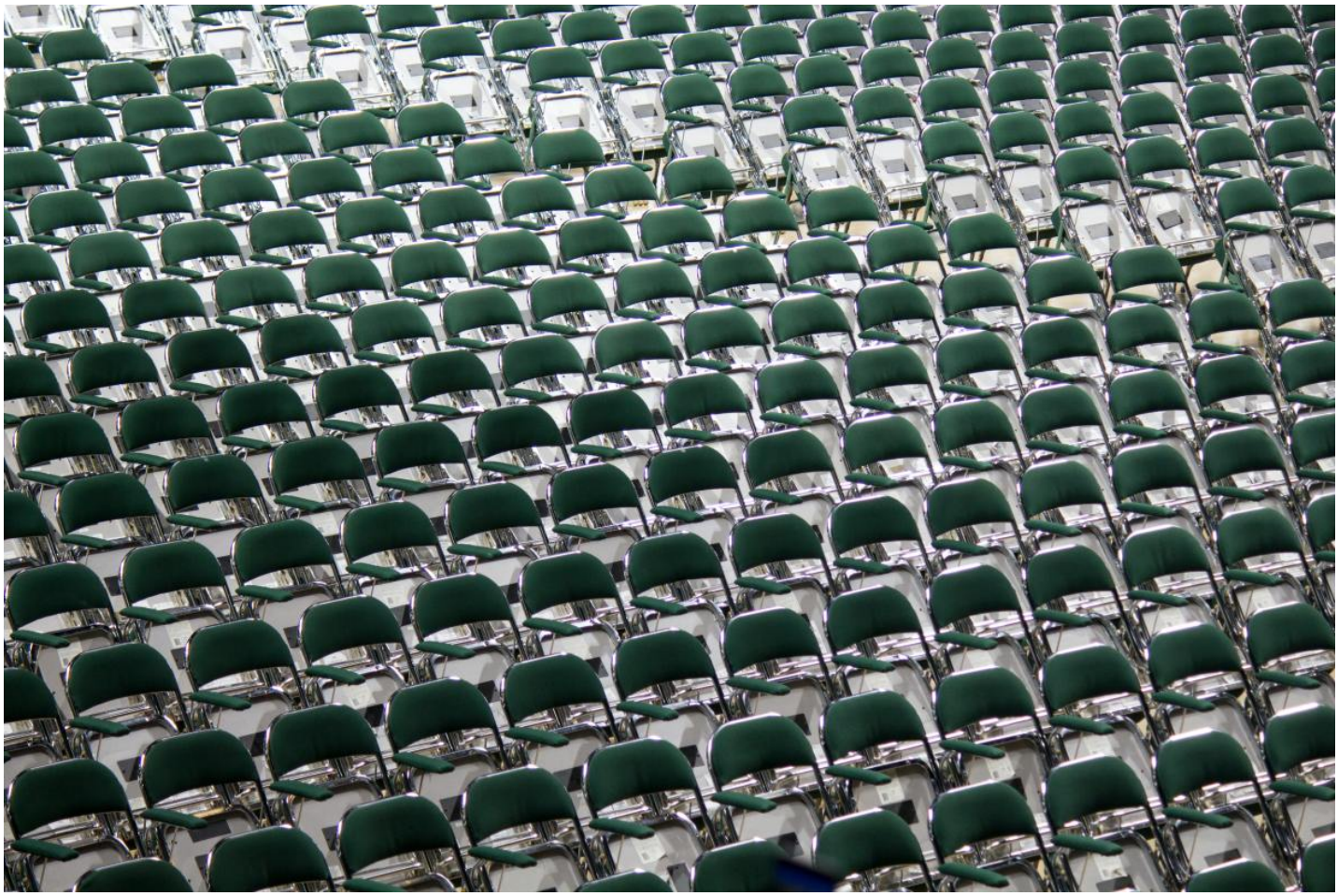
References:

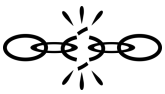
- Header image: *"Construction helmets"*. Pop & Zebra, photographer. Retrieved from Unsplash, https://unsplash.com/photos/wp81DxKUd1E.

Improving the Environment was originally published March 20, 2012.

Notes:

Copy One That Works





> Copy, paste, and tweak: that's a technique as old as time... You don't have to understand it. It's expedient and it feels dirty, but it works. Up to a point.
>
> **Alex Chaffee**

When I first began to repair computers, I quickly discovered the value of having a working computer next to one that was broken. Even better if the machine that was working happened to be the **exact same make and model** as the broken one. For starters, a working computer could access the Internet so I could look at manuals and search forums for other people experiencing the same problem. However, the advantages went way beyond that.

Having "one that works," a functioning copy of something broken, gives you the ability to swap parts between the two machines in an attempt to isolate a failing component. If you think a particular part is the culprit, you can easily test your hypothesis by taking it from the working machine and putting it in the broken one (and vice versa).

A working version is also useful as a *model* to copy. In our lives, we employ the principle of modeling all the time. If you meet someone who is successful at something you aspire to, you want to ask them: "What steps did you take to get where you are?" The idea being, if you took a similar path, you could recreate what they've done. If someone serves you a bacon-topped donut (what a friend!), you may feel compelled to inquire about the recipe, with the hopes you can pull it off in your own kitchen (a recipe is a model, too). Working machines can't answer questions about why they're "successful," but you can observe them and infer things about how they function. A working model is the basis from which to take measurements, see where parts are located, watch normal operation, and copy configurations and

settings. What's different between the working one and the broken one?

Twinsies! Having an identical one around might threaten your sense of originality, but actually it's great for troubleshooting.
(image: Matthew Rutledge / CC BY 2.0)

Again, Just One Thing

Be sure to only change one thing at a time when you're swapping anything between a machine that works and a machine that's broken. Always retest for the failed condition after you make a change. If you replace two or more components simultaneously, and it suddenly works, how will you know which change actually solved the problem? That's right, you won't.

Proceed cautiously with your working copy, especially if it's your only one on hand. Make sure you reverse any changes you make to your model before proceeding to test a new theory. Tearing apart your functional clone is like messing around with your life raft while adrift at sea. Having *two* broken machines is much worse than one, especially if you intended to use the working one for actual work while you fixed the broken one.

Warning: Corruption Can Occur

While swapping components is a great strategy for isolating and testing suspect parts, be aware that sometimes transferring a working component to a failed system will destroy the part. For instance, if you have a dodgy power supply that has burned out a motherboard, and you swap a fresh motherboard from a working computer…well, there goes another motherboard! As noted above, try to think a few steps ahead with regards to your working replica: if accidentally destroying "the one that works" will leave you in a desperate situation, you should avoid swapping parts. Since this decision is highly contextual, you'll have to use your best judgement about the risks and rewards of trading components.

On the outside, they may look identical. But under the hood, is everything really the same?
(image: Hugo90 / CC BY 2.0)

Weird Stuff

You would think that having a pair of identical, *working* machines would allow you to move parts between them with no change in the working status of either system. That's what you'd think, but you'd often be wrong! When it comes to swapping parts among identical machines, a down-to-the-atom level of similarity is an ideal that you'll never reach (although the nanotech industry is working on it). You never really have two "identical" systems to swap parts between. Sure, they may be the same make and model year. Heck, they might even have rolled off the assembly line one right after the other! But, they're not really the same by the standard of an atom-by-atom comparison. On top of that, when you're repairing machines that have actually been doing *work*, even machines that started out the same at the factory will have diverged over time.

Slight, but meaningful, differences are behind the similar-but-incompatible phenomenon:

- **Model year:** this year's model and last year's model may *look* the same on the outside, but manufacturers often make internal changes that may be difficult to see (or understand!).
- **Manufacturer's revisions:** apart from the model year, manufacturers will frequently decide to change—pretty much anything—and still keep the same model number. Components can be added or eliminated, miniaturized, rearranged, etc. The manufacturer may do this because they've found a cheaper way to make the product or to eliminate defects found in earlier production runs. On top of that, different facilities may be manufacturing the "same" product, but doing so with slight variations.
- **Firmware/software version:** for any system that has digital components, there will be software. Those crafty programmers, having nothing better to do, can churn out revision after revision—as if that's what they were paid to do. Depending on what version of the software was current at the time it rolled off the assembly line, or what was current at the time it was last serviced/updated, you can have multiple "identical" machines each running slightly different code.
- **Usage:** over time, machines will begin to reflect how they are used. Differences in workload, accidents, location of installation, and time in use will begin to impact tolerances between components and leave unique wear patterns. Eventually, these differences may prevent a part from being swapped between two supposedly identical machines.
- **Maintenance history:** regular maintenance (or lack thereof), rebuilds, use of different brands of replacement parts, and customizations can subtly change a machine. If two machines are not maintained exactly the same way, they will diverge over time.

Note that some of these differences might not be visible to the naked eye (especially minute differences in tolerances created by the wear and tear of moving parts). Keeping this subtlety in mind will improve your chances of success when employing the "copy one that works" strategy.

Don't Just Copy Machines, People Can Be Modeled Too

I'm not talking about strutting your stuff on the catwalk. We've discussed copying a working machine, but don't stop there. If you know someone who's really good at fixing a particular type of system, consider imitating them as well. How do they make their diagnoses? Which tools do they use? What kind of training have they received? While writing *The Art Of Troubleshooting,* I interviewed many great troubleshooters to learn their methods. You can do the same: ask if you can observe them on the job, or offer to buy them lunch and pick their brains. What do they know that can help you become a better problem-solver? Ask and you shall receive.

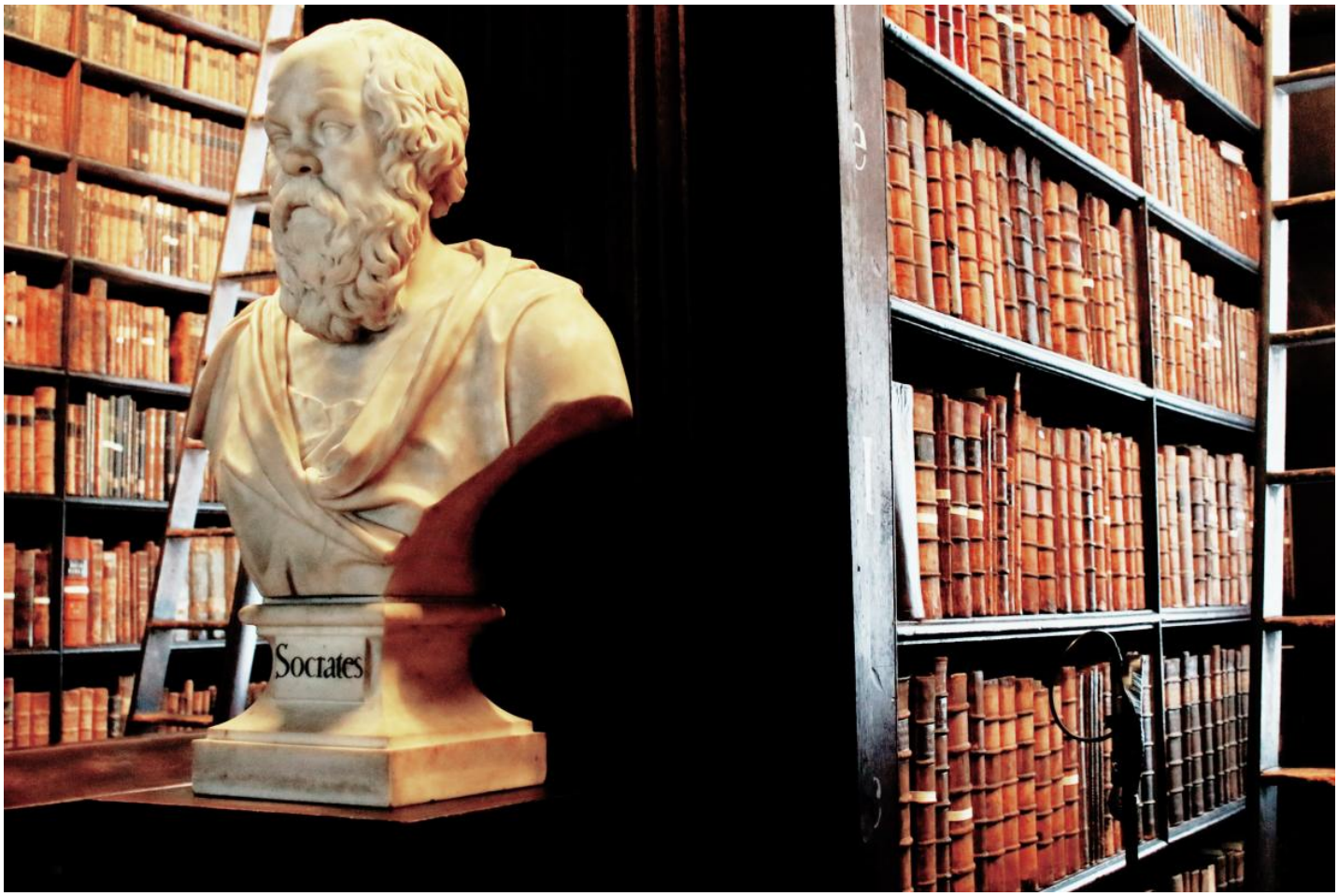
References:

- Header image: Katie Montgomery, photographer. Retrieved from Unsplash, https://unsplash.com/photos/j5dD6jNLhHk.

Copy One That Works was originally published April 17, 2012.

Notes:

Let's Be Reasonable



> Real world projects are not logical proofs. So you can use these techniques, but they can lead you astray.
>
> **Alex Chaffee**

Troubleshooting requires the use of reason, all problem-solving does. As reality always has the last word, make sure you're in tune with what it's saying. So, let's fire up your left brain and learn about the principles of logic relevant to troubleshooting, while trying to avoid cutting ourselves with Occam's Razor.

Induction

Induction is a type of reasoning that makes generalizations from specific facts. A classic example from the ancient philosophical texts:

1. A Pontiac Trans Am is wicked cool.
2. A Pontiac Trans Am is a car.
3. Therefore, all cars are wicked cool.

Based on this fine specimen, can we make the leap and conclude that all cars are awesome? Welcome to the problem of induction…

(image: jackxface / CC BY-ND 2.0)

Okay, okay, that example was just an excuse to include a picture of a sweet 'Merican muscle car. Also, that particular leap from specific to general is ill-conceived. Here's the actual example from antiquity:

1. Socrates is mortal.
2. Plato is mortal.
3. Aristotle is mortal.
4. Therefore, all men are mortal.

Induction is a powerful method of organizing your discoveries about the world. Going from the specific to the general allows us to know things that would otherwise be costly to discover without this power of abstraction. The payoff comes later when you take what was learned via induction and make judgements about specific things using induction's opposite: deduction.

Specific And General Problems

Induction's associated pitfalls have been discussed since ancient times and could fill a library. How to proceed from a set of specifics to a generalization can be a complicated matter. This is supposed to be a slim volume of very practical knowledge, so if you're viscerally excited by the "Problem of Induction," you'll have to go and study the topic on your own (preferably in a black beret and turtleneck in a coffee shop somewhere). However, I do want to acquaint you with some common induction problems that are frequently encountered by troubleshooters.

Sample size: going from specific to general based on a single example or without knowledge of the entire universe of possibilities can lead to generalizations that aren't true. You can see the problem in the two examples above: we've yet to encounter a man that isn't mortal. However, looking at one Trans Am and leaping to the conclusion that all cars are cool is easily falsifiable. Good sir or madam, have you never laid eyes on a Geo Metro?!

Induction is a chain: your conclusion is only as good as your initial premises. "Garbage in, garbage out" nicely expresses this potential problem with induction. If we had misjudged the true nature of Socrates, and he really was immortal, then our subsequent reasoning is really going to be screwed up! Put another way, if your initial specific observation isn't right, errors will be amplified when you make a generalization based upon it.

Loose conceptual connections: the conclusion that man is mortal flows easily from man's nature, perhaps even from

the very definition of man as a rational *animal* whose biology as a living organism necessitates the possibility of death. In contrast, "all cars are wicked cool," seems like a conclusion with a looser connection to the concept of a car. Cars were designed for the purpose transportation and their essential characteristics include things like doors, wheels, and engines. "Coolness" seems further afield. Of course, a generalization about all cars being cool could be true, but its disconnectedness should be a red flag for further investigation.

Deduction

Deduction reverses the direction, going from the general to the specific. Back to our friend Socrates for an example:

1. All men are mortal.
2. Socrates is a man.
3. Therefore, Socrates is mortal.

Deduction is a very useful time-saving shortcut: if you know something is true in general, it must also be true on the smaller scale of a specific example. If "all men are mortal," you don't have to ask a gunshot victim if they need help. If they're human, they are vulnerable to death, so of course they do!

Deduction is a powerful tool, but it too has its pitfalls. One problem involves improper classification: if you perceive something to be a member of a broader class when it's really not, you will be making a big error when you apply your generalizations. For example, if you happen to be a comic book villain and you mistake Superman for a "man" (one like you and me), you may find yourself really confused when he won't die.

Also, any errors you made during the induction phase will come back to haunt your deductions. If you made a generalization that simply wasn't true (perhaps because you hadn't discovered all the different possible cases) that mistake will carry through when you apply it to a specific case.

Induction And Deduction In Action While Troubleshooting

Now, let's make the connection with troubleshooting. Understanding and applying both induction and deduction properly is important because these logical principles show up all the time when you're on the hunt for a solution to a breakdown.

When using deduction, your prior experience with a machine will act like a generalization. "All men are mortal" is analogous to "when my car fails to start the cause is a dead battery." However, you can probably spot the difference between philosophical truths and those encountered while troubleshooting: those turtlenecked deep thinkers sitting in coffee shops may make conclusions based on generalizations that are universally true for all time and for all cases. By contrast, the matching of symptoms with failures while troubleshooting will often be *statistically* true (e.g., "83% of the time, this type of problem is caused by X…"). Most systems can fail in a mind-boggling number of ways, so there's no guarantee that the cause of a particular failure will be the same as last time, even if the symptoms are *exactly* the same (see "Same Symptom, Different Causes"). Your experience can be a very powerful time-saver, quickly pointing the way to a solution. However, be aware of the problem of deduction and think before applying the generalizations of your experience to all scenarios. Troubleshooting is so often about the exceptions!

Speaking of exceptions, when you find counterexamples to your troubleshooting generalizations, you need to incorporate them to make your conclusions tighter. Take a generalization, formed from your experience, like "when my car fails to start the cause is a dead battery." One day, you will inevitably find yourself hooking up jumper cables to your car, only to have it *not* start! That is, the car won't start, but something other than a dead battery will be the cause. This is exciting, because you are on the verge of discovering a *new* cause of your problem. Perhaps this time you discover the car is out of gas. Now, your original generalization can be narrowed to: "If the car is otherwise functional and there is gas in the tank, a dead battery will prevent it from starting." When there's more than a few exceptions, stating them like this can become unwieldy, so this type of information is better presented in a troubleshooting tree.

Induction will come into play during the "Cleaning Up" phase, where you try to learn from and prevent a particular breakdown from happening again. You'll have some specific facts, the circumstances surrounding a failure, from which you will attempt to form generalizations. Those generalizations may be procedures for how to conduct future repairs (e.g., "Next time, we should attack the problem like this…") or recommendations for routine maintenance (e.g., "We should replace this part every month…"). The challenge is making these leaps with incomplete information,

keeping in mind that sometimes you may not have enough data to make a recommendation with certainty. As I've said before, you should always make the best guess you can with the information you do have and then correct your course as you go along by collecting and analyzing data.

Hypothesis Testing

When troubleshooting, you should look for opportunities to create testable hypotheses regarding the failure of a system. Remember the generic problem-solving formula I introduced in "One-size-doesn't-fit-all":

> **Step 1.** Defining the problem
> **Step 2.** Gathering facts
> **Step 3.** Analyzing information
> **Step 4.** Eliminating possibilities
> **Step 5.** Proposing a hypothesis
> **Step 6.** Testing the hypothesis
> **Step 7.** Solving the problem
>
> **Amir Ranjbar, Troubleshooting and Maintaining Cisco IP Networks** [1]

Steps 5-6 form a loop that can be iterated over and over until a solution is found. Form a hypothesis, test it and, if the problem persists, incorporate what you've learned into a new one. Wash, rinse, repeat.

Asserting something about the cause of the failure (and when I say "something," I mean *anything*), can be a good way to start theorizing. Psychologically, I think this is because it's often useful to have something to resist against. To facilitate the formation of hypotheses, I suggest making statements of the form: "If X is being caused by Y, then Z must be true." Say it out loud.

Examples:

"If the car won't start because the battery is dead, then replacing the battery will allow the car to start."

This is a very clear hypothesis statement that includes two avenues for action: 1) testing the battery to see if it's dead and, if so, 2) replacing it to see if the car will start.

"If the low pressure readings are being caused by a faulty pressure gauge, then replacing the gauge with one that is known to work will restore normal readings."

Again, a good hypothesis makes the course of action to be taken obvious.

Everything Has To Fit

The goal is to find a hypothesis that integrates all of the known facts; the goal of testing the hypothesis is to expose unknown facts that you haven't accounted for. As shown in my ode to data collection, there are an infinite number of things that can be observed and recorded for any given machine. So, you'll never get to know everything about a particular problem. For the purposes of troubleshooting, all you really care about are the subset of the facts that are preventing the machine from working. The hypothesis testing loop above is a great way to discover these relevant items.

Keep It Simple

A bias towards simplicity should drive your evaluation of competing hypotheses. You may have heard of Occam's Razor:

> [Occam's Razor] implies that—other things being equal—it is rational to prefer theories which commit us to smaller ontologies.

> **The Stanford Encyclopedia of Philosophy** [2]

"Smaller ontologies" means theories with fewer entities, a parsimony of thought. Occam's Razor is sometimes misunderstood to mean that you should always favor simpler explanations over more complicated ones. Simple or complex, the hypothesis must always fit the data. If you're trading simplicity for an equal amount of explanatory power, Occam's Razor doesn't say that one hypothesis is necessarily better than the other.

While troubleshooting, it's easy to get carried away with complicated explanations for a failure. Are you beginning to entertain theories involving Leprechauns and Unicorns, prancing around on epicycles? That's your wake up call that you might be on the wrong track. Most likely, you've subconsciously blocked off an entire realm of possibilities or haven't followed the evidence to its logical conclusion.

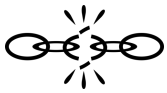
References:

- Header image: Fleur, photographer. Retrieved from Unsplash, https://unsplash.com/photos/QreQvdSr-Wc.
- [1] Amir Ranjbar, *Troubleshooting and Maintaining Cisco IP Networks* (Indianapolis: Cisco Press, 2010), pg. 41.
- [2] Simplicity, *The Stanford Encyclopedia of Philosophy*.
- Ben Bayer. *Introductory Practical Logic.*

Let's Be Reasonable was originally published September 1, 2012.

Notes:

Know Your Limits



I loved carburetors. I'd make $90 to clean a 4-barrel carburetor, using a kit that cost $37. I had a guy come in and I quoted him $90. He said, "That's too damn high, I'll get the kit and do it myself." I said, "Okay, go ahead and do it yourself."

He came back later with a pail full of parts and said, "Can you put this back together?" I said, "Yeah, it'll be $90." He said, "$90!" I said, "Did you get the instructions? They should have come with the kit." You have to smile while you're saying this, of course.

Twenty minutes later I had it back together. He said, "Wow, that's fast money." I said, "You can do the same thing, just learn what I've learned and start up your own business."

Gerald Quade

This one is for the amateurs. And, we're all amateurs. Even if you're a professional that's paid to fix things, you're surely a novice in some context. The auto mechanic might not be good at fixing computers. A nuclear engineer won't

necessarily be up on the latest-and-greatest in the world of riding lawn mowers. Even when operating in their area of expertise, good troubleshooters know when they're in over their heads. When this happens, they know to call for help!

The decision to troubleshoot is a combination of several interrelated factors:

- **Motivation:** the pressure or driving force behind getting something fixed. If it's part of your job, you may not have a choice in this regard.
- **Resources:** time, money, people and tools available to help you troubleshoot.
- **Expertise:** the ability to find the problem and make a particular repair. Even though you could learn how to fix something, the time and effort required might be prohibitive.
- **Risks/safety issues:** knowing what's at stake. Is the company's fortunes riding on making a repair? Could someone die or get injured in the process?

The context in which you troubleshoot affects these factors greatly. Hobby-related troubleshooting projects typically have no deadlines: I've seen classic car and motorcycle restorations that have taken years to complete! In contrast, if you're a technician working for a telecom company, chances are your clients expect problems to be fixed *today*. Any of the above considerations can also be the deciding factor to *not troubleshoot*. These same factors (motivation, resources, expertise, risks/safety) also come into play when deciding whether to call in a professional to help you fix something. Let's weigh the pros and cons for getting involved with an outside professional in each of these categories:

Motivation:

- **Pros:** If it *must* be fixed (and you can't do it), then getting outside help might be the only way.
- **Cons:** In a low-stakes situation, what's the rush? In these cases, I like to see how far I can push myself—these relaxed scenarios are great for improving your troubleshooting skills. If you hit a wall you can always call for backup, so consider giving it a try by yourself first.

Resources:

- **Pros:** In the right context, hiring a professional can actually *save* resources. If the rest of your staff is busy with more important things, hiring a professional troubleshooter will allow them to continue uninterrupted. Sometimes a breakdown will hold up production, idling expensive materials, machines, and manpower. In these situations, a professional's ability to reach a resolution faster can be worth many times their fees. Finally, hired guns often have access to expensive and specialized tools that you'd rather not buy just to make a single repair.
- **Cons:** Hiring and managing a professional consumes its own resources. First off, there's the professional's fee, but this isn't the only cost. You must first search for the right person or firm, selecting them among the various candidates (perhaps by checking references and vetting their previous work). Also, time is required to present the problem to them in a way they can understand and bring them up to speed about your infrastructure. You must coordinate schedules, deliver the broken system to them or make an appointment for them to come on site. Finally, when they say it's fixed, will you need to double-check their work?

Expertise:

- **Pros:** When you lack the ability to find the problem or make a particular repair, the need for a professional becomes obvious. Constraints may not allow you to learn what's necessary in the timeframe required; delegation may not be possible because of a lack of staff.
- **Cons:** The time you spend learning how to fix something yourself can pay dividends later. If a particular type of failure happens often enough, you'll probably want to know how to repair it on your own.

Risks/Safety:

- **Pros:** A competent professional will be more aware of the hazards associated with attempting a given repair. If they are concerned with their reputation, they will typically advocate the safest path that guarantees results. Also, you might make the situation *worse* by botching a fix. I definitely have.
- **Cons:** There's always the chance of hiring someone who's dishonest or incompetent (or, if you're really cursed, both!). Dealing with the fallout of hiring a bad contractor can be *very* time consuming. Letting someone into your facilities risks theft and exposure of trade secrets. Also, if the repair is critical to your business, can you trust handing it over to someone else?

Losing the fight? Time to call in the cavalry.
(image: Édouard Detaille / Wikimedia Commons)

Let The Circumstances Decide

One of my goals in writing *The Art Of Troubleshooting* is to lower the perceived barriers to troubleshooting through education. Even though the world of machines continues to increase in complexity, there's still so much you can do for yourself. Knowing the basics and having a language to speak about troubleshooting, you can go very far. That being said, I have no bias towards doing it yourself or hiring someone to help. Expanding your fix-it skills can be very gratifying, but ultimately the troubleshooting process must be guided by the factors listed above. Go with what the situation favors.

Second Opinions

I've had many ridiculous, super-sized estimates for repair work cross my desk over the years. You may "get what you pay for," but that doesn't mean you have to gold-plate somebody's yacht. A professional may know more than you, but they are not infallible. Don't be afraid to ask questions and seek clarification. After all, they work for you! For really expensive or important repair decisions, take a note from medicine and consider a second opinion. Or even a third: the conventional wisdom in business is to always get 3 estimates when considering a large purchase. This approach has saved me thousands in repair bills over the years!

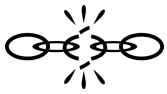
References:

- Header image: Harris & Ewing, photographer. *Repairing government trucks at the Treasury procurement section.* United States, Washington D.C., ca. 1937. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016872242/.

Know Your Limits was originally published February 22, 2013.

Notes:

Where Do I Begin?



> Check the stupid stuff. Before you tear apart the alternator, see if the belt is tight.
>
> **Jeremy Sheetz**

Every troubleshooting project requires an entry point. How do you find a good place to start? Choosing poorly can mean the difference between fruitful problem-solving and a trip down the proverbial rabbit hole. It seems like it should be straightforward, but it's often not. What's usually obvious is the symptom, but of course that's different than the cause. If you're new to a machine, knowing where to begin will be an intuitive process involving trial and error. Before you've achieved expertise with a system, consider these promising possibilities for starting points:

Start By Duplicating The Problem

Always a great place to start, because replicating the problem allows you to start a loop where you:

1. Try something
2. Test to see if the failure is still present

Unfortunately, there are times where attempting to duplicate a problem isn't a very useful beginning, like when:

- The machine has not recently been operational and there are likely to be multiple failures. The duplicability of these situations will be 100%, but that fact won't necessarily aid problem discovery.
- The problem is intermittent.

If only where to start was always this obvious…
(image: Michel Filion / CC BY-ND 2.0)

Start With An Inspection

Giving a failed system a once-over is a great way to initiate the problem discovery phase. A basic inspection may uncover obvious signs like smoke, weird smells, or noises. These attention-grabbing symptoms may highlight the problem area, but remember not to confuse them with the actual *cause* of the problem. Even so, it's a gift when the starting point is so obviously presented to you.

A basic inspection may also turn up less glaring, yet still promising leads. I'm talking about cracks, dirty contacts, frayed wires, or bent parts. You might notice a disconnected hose, a wire about to come off a post, or a loose screw. Machines are supposed to be orderly inside, so even if you aren't an expert in how a system is supposed to work, you can still spot things that are amiss. You will hear yourself saying: "This doesn't look right." When searching for things out of place, include fluids not where they should be (aka, leaks).

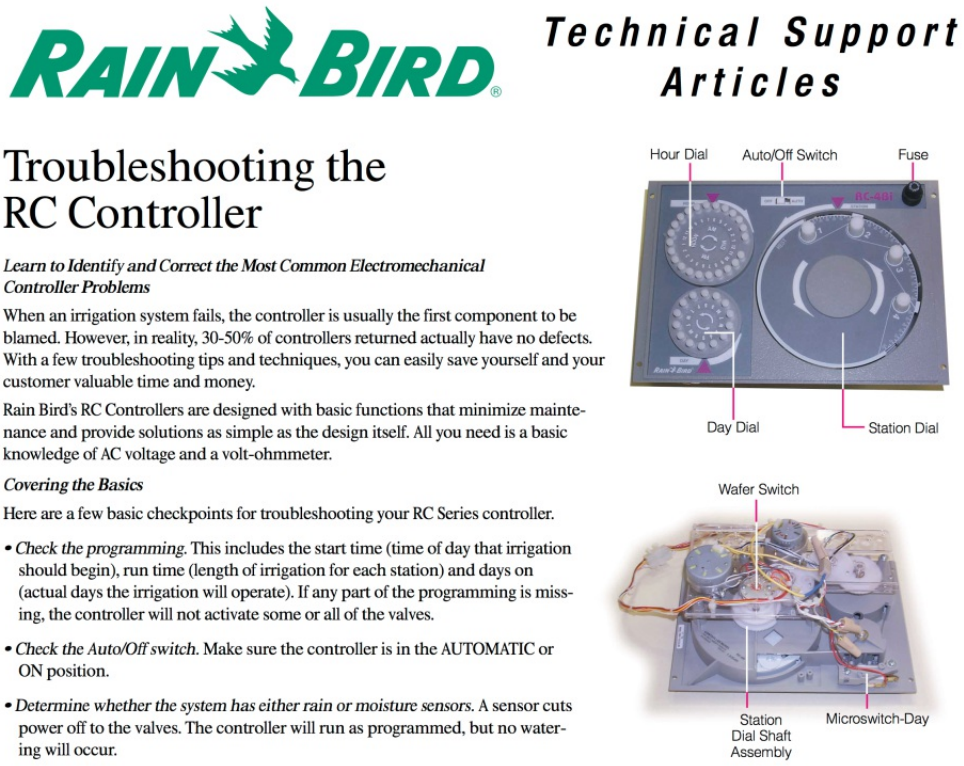
Digital devices have an extra layer, beyond the physical, that requires inspecting. Of course, they also have problems that manifest themselves physically: I've smelled all kinds of wonderful things emanating from computers over the years (like burning power supplies). However, there are a whole host of problems that can lie hidden among the bits, unobservable to the naked eye. For this dimension, viewing the console, running diagnostic programs, and examining logs are the digital equivalent of the walk-around.

Start With Routine Maintenance

Any time you discover a lapse in routine maintenance, you've also found a great starting point. You know that maintenance regimes are designed to prevent the most common problems, so it makes sense that simply doing the recommended upkeep can bring a system back to life.

Part of routine maintenance is keeping current with the latest-and-greatest from the manufacturer and end user community. This means awareness of recalls, applying firmware or software upgrades, and adhering to any new "best

practices" on how to effectively use a machine. Especially when a problem is known and a fix readily available (as with a recall or a software update), this aspect of maintenance is a great place to start a new troubleshooting project.



RAIN BIRD

Technical Support Articles

Troubleshooting the RC Controller

Learn to Identify and Correct the Most Common Electromechanical Controller Problems

When an irrigation system fails, the controller is usually the first component to be blamed. However, in reality, 30-50% of controllers returned actually have no defects. With a few troubleshooting tips and techniques, you can easily save yourself and your customer valuable time and money.

Rain Bird's RC Controllers are designed with basic functions that minimize maintenance and provide solutions as simple as the design itself. All you need is a basic knowledge of AC voltage and a volt-ohmmeter.

Covering the Basics

Here are a few basic checkpoints for troubleshooting your RC Series controller.

- *Check the programming.* This includes the start time (time of day that irrigation should begin), run time (length of irrigation for each station) and days on (actual days the irrigation will operate). If any part of the programming is missing, the controller will not activate some or all of the valves.
- *Check the Auto/Off switch.* Make sure the controller is in the AUTOMATIC or ON position.
- *Determine whether the system has either rain or moisture sensors.* A sensor cuts power off to the valves. The controller will run as programmed, but no watering will occur.

Excerpt: manufacturer troubleshooting guides like this will save you a lot of time and are a great place to start.

(source: Rain Bird RC Troubleshooting Guide)

Start With The Manual Or Technical Support Documents

Many manufacturers kindly list the most common problems and solutions for a particular product in their manuals or support guides. Yes, you should take the time to familiarize yourself with this information. Why reinvent the wheel?

You might usually throw this kind of material away. That's okay, because most manufacturers publish their product manuals online for easy access. If the machine in question is expensive and/or mass-produced, there may be significant third-party resources available to help you troubleshoot, perhaps with step-by-step instructions for solving even the smallest of problems. Automobiles, for example, have a large professional troubleshooting industry that spans the spectrum from do-it-yourself (manuals, parts stores, etc.) to full-service "don't make me lift a wrench" solutions (repair shops).

Manufacturer materials are usually a big win because of the deep experience they have with their products. From design, to testing, to manufacturing, to reports from customer service, they are in a unique position to give good advice. For you, the amount of time and expertise needed to independently discover the knowledge encapsulated in even one line of a troubleshooting guide might be significant. The general principles in *The Art Of Troubleshooting* will help you to address any problem, but there really is no substitute for having the solution to your *exact* problem laid out in black and white. Don't worry, there will be plenty of "undocumented" problems for you to solve. For the rest, feel free to take the easy route! Again, if the solution was there all along, you're going to kick yourself later if you wasted your valuable time trying to be a hero.

Even if there isn't a troubleshooting guide for your particular problem, contacting the manufacturer might still be fruitful: they may not be able to offer a solution, but someone in the service department should be able to steer you in the right direction.

Start By Realizing That You Are Not A Beautiful Or Unique Snowflake

At least not when it comes to troubleshooting. When fixing mass-produced items like consumer electronics or cars, take comfort in the fact that you're not alone. There's a good chance that someone has encountered your problem before and there is a fix or a workaround just waiting to be uncovered, should you bother to lift your fingers and type the magic words into a search engine (or your company's issue tracking database).

I've been totally astonished by breakdowns I would have thought to be completely unique. We're talking about some really obscure and exotic failures. However, after searching online forums, I discover:

1. Hundreds or even thousands of others have encountered my same problem.
2. A fix or workaround that is fully tested, documented, and ready to implement (sometimes years old!).

Welcome to living on a planet with 7+ billion people. Sorry, but you'll need to find some other way to make yourself feel unique beside having the most interesting malfunction (may I suggest a vintage hat worn at a jaunty angle or perhaps a t-shirt with a clever slogan?). The upside is that you can leverage the collective experience of the masses. Save your sweat for the situations that actually require "going it alone." Being a smart troubleshooter means tapping into the social network. It doesn't matter to me if you solve the problem by Googling or through a Herculean triumph of reasoning.

Either way, you can take the credit. After finding a solution off the Internet (which sounds boring), I occasionally add a little drama by telling people that the answer came to me in a dream: a soaring Bald Eagle represented the engine and a man wearing a loin cloth made of bacon represented the bad spark plug…

Start Where The Graphs Get Funky

If you have solid operational data and can answer the question "What is normal?," then a good place to start troubleshooting is with your graphs. Look for:

1. Parameters outside their normal range
2. Variability/volatility
3. Increasing/decreasing periodicity (i.e., cycles taking longer or shorter than normal)
4. Upward or declining trends

Cross-comparing and overlaying data can be a powerful way to start theorizing about the cause of a problem. Let your data point the way!

Start With Recent Changes

Recent changes to a machine or its environment are great starting points for an investigation, as explained in "What's Changed?"

Start With The People On The Front Lines

Have you talked to the people on the front lines who might know the answer? I discovered this, quite accidentally, after doing some solo analyses of downtime incidents. I would diligently collect and analyze data on my own and be kept up late at night pondering the causes of these problems.

Frequently, after I had found a solution and written a report about it (perhaps months later), I would be casually chatting with one my engineers and the topic of one of these incidents would come up. In the middle of telling my triumphant troubleshooting story, they would usually stop me and say something like: "Oh yeah, I've seen that before: I bet it was…"

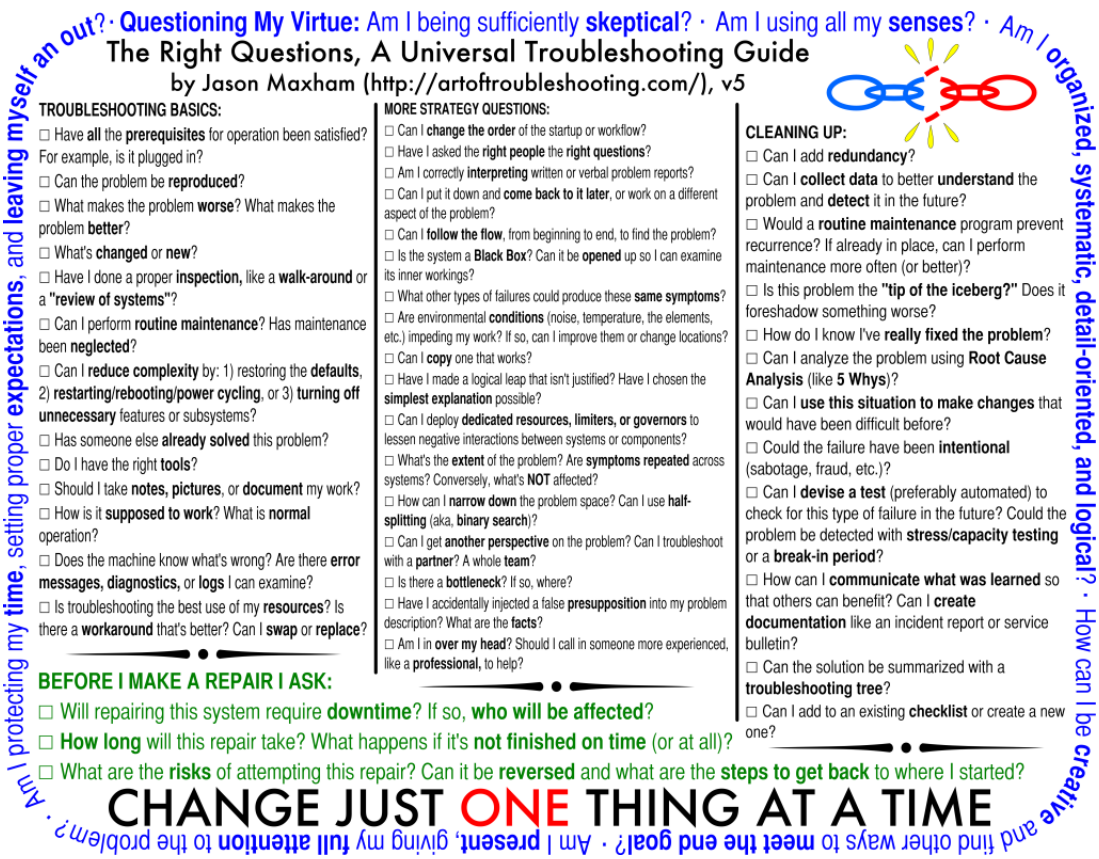
Cut to a deflated look on my face. What had taken me days of painstaking investigative work to identify, they already knew, and were carrying around in their heads. Ugh! I would cry out "Why didn't you say something?!" Frequently, their answer was: "No one asked me."

After this happened a few times, I began to cast a very wide net when troubleshooting. The things I *didn't* know (but were known by others) frequently surprised me, and I had been with the company from day one!

I don't think I was oblivious, but I sure wasn't getting our problems in front of the right people. The takeaway is: are

you ignoring those who might know the answer or can quickly orient you? People on the front lines: operators, maintenance personnel, programmers, systems administrators, etc. will have relevant firsthand experiences and useful knowledge. Ask them!

Begin With This Most Excellent List Of Questions



My free 1-page Universal Troubleshooting Guide distills the strategies described in *The Art Of Troubleshooting* down to a powerful list of questions that you can ask yourself while actually problem-solving. It's the best way to start any troubleshooting exercise!

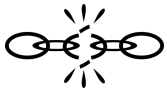
References:

- Header image: Bain News Service, P. *Starting Aeroplane Motor*. ca. 1915-1920. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2014708511/.

Where Do I Begin? was originally published March 1, 2013.

Notes:

What's Changed?



> I look for anything out of place. Is there something burned? Is there something that looks melted? Is it shiny when everything else is dull? Is there something glowing cherry red? It's that saying from Sesame Street: "One of these things is not like the others."
>
> **Mike McCormick**

The buildup to a machine failure can involve normal wear and tear, neglected maintenance, misuse, abuse, and Acts of God. Add one more to this list: the unintended consequences flowing from changes to how a machine is used, configured, or to its environment.

Lao Tzu says:

> What is recent is easy to correct. [1]

The Troubleshooter says:

> What is recently changed is likely to have caused the problem.

Making modifications to a system or its environment can be the unintentional catalyst for a meltdown. Perhaps an increased workload pushes a machine to its breaking point. Maybe a recent software upgrade contains a bug that disables a critical feature. Perchance a machine gets too hot when someone accidentally turns off the air conditioning. Experience says that looking for what's recently changed is a great starting point for any troubleshooting exercise. Look for these type of situations:

Notice anything different?
(image: Honoré Daumier / Wikimedia Commons)

The Floodgates Have Been Opened (Or Closed)

Change type: **Workload**

If a machine does the same job, day in and day out, its behavior will be fairly predictable. However, when you significantly change a system's workload, it can be the setup for a breakdown. Even though a machine may be designed for the additional burden, an increased workload often exposes wear and tear and foregone maintenance. It's a chipper that's been used to mince light brush, but then is fed a small tree and has a meltdown. It's an aging SUV that has never left the pavement, then cracks an axle on its first off-road adventure.

I can think of numerous examples from my own life where increased usage has tipped a machine over the edge. I had a paper shredder that was just fine with a single sheet of paper at a time, but died a sudden death when I tried to do 5 at a time. I owned an old car that was fine for driving around town, but wasn't up for a long road trip. And on and on.

Surprisingly, *decreasing* a system's workload can also have negative consequences. When storing a car long-term, it's recommended that you start the car occasionally to charge the battery and keep key components lubricated. You may never have considered it, but *using* a car is part of its maintenance, just like an oil change! Our bodies are the same way, a minimum amount of activity is required to keep them functioning. Astronauts living in a weightless environment have to battle severe atrophy of their muscles because of their lessened usage. Waterless urinals are another great example of the unintended consequences of decreased throughput: the high concentration of acids, normally diluted by water when flushing a traditional urinal, will eventually corrode copper pipes.

Finally, the work done by a machine doesn't have to be any different than before, because simply more of it can cause

trouble. For example, if your employer is thinking about adding a second or third shift, your machines' usage could double or triple. Increased usage shortens the required maintenance interval: if you run a single 8-hour shift and do scheduled maintenance every 3 months, then expanding to 3 shifts (24-hour usage) will require maintenance on a monthly basis. If you don't recognize this and stick to the old schedule, prepare for some problems!

A Switch (Or Bit) Was Flipped

Change types: **Configuration, Maintenance, Software Version**

How a machine is set up makes the difference between it doing useful work…and it doing nothing. Some systems have multiple work "modes," and perhaps only one is useful for your purposes. This means that a well-meaning, but errant, alteration of a machine's configuration can render it non-operational. Sometimes the erroneous modification will target a specific function, while confusingly other features will continue to work. A classic example of this is accidentally pressing the "mute button" on a TV. While the sound may be off, this doesn't affect your ability to change channels.

You should also be suspicious of breakdowns immediately following routine maintenance. It wouldn't be the first time a technician failed to properly reassemble a machine after a service visit!

Software upgrades should also be on your radar when looking at recent changes. Many a troubleshooting investigation has begun with these words: "So, I just got done running the updater and then it stopped working…" Reverting to a prior version is sometimes the only way to fix an issue while the programmers try to find the cause.

Don't Be A Test Pilot

Change type: **Overloading**

Pushing a machine beyond its design limits will result in failures. The pressure behind these type of incidents usually comes from trying to handle too much business with too few resources. Feeling the pressure of clients and deadlines, people may respond by pushing equipment beyond its breaking point. Whenever we won a large contract, my mind would race ahead, thinking about all of the things likely to be broken as our engineers stretched our infrastructure to its limits.

Given that many machines have a built-in "margin of safety," failures from overloading may take a while to fully manifest themselves. When the line is first crossed, nothing may happen immediately, and so further transgressions are seemingly justified. A truck designed to haul 5,000 pounds might make repeated trips loaded with 6,000 lbs. before something bad happens. Catastrophic failures from overloading can have a delay, because damage from overuse can take a while to fully reveal itself. When the chaos finally rains down, the machine might even be *properly loaded*, causing even more confusion about the origin of the issue.

When I was studying for my pilot's license, my instructor and I were reviewing how to calculate the cross-wind component of various headings and wind speeds. The Cessna 172, the airplane on which I was training, has a "maximum demonstrated crosswind velocity" of 15 knots. "What if you exceed that?," I asked. He said, "Congratulations, you're now a test pilot! But, don't *ever* be a test pilot."

The Environment

Change type: **Context**

A change that causes a failure doesn't necessarily have to originate within a machine itself. Rather, it can come from the *environment* in which the machine is installed.

The Weakest Link: this is where a system in a workflow chain has changed its output, affecting the input fed to a machine downstream. For instance, imagine two machines connected by a conveyor belt: one crushes gravel and the other puts the gravel into bags. The bagging machine requires the stones to be less than a certain size or else it will become clogged. However, the crushing machine can be set to produce gravel of any size. If the crushing machine is accidentally set to produce gravel that is too large, it will operate normally. However, the problem will appear downstream, at the bagging machine.

Whether you use pipes, cables, or conveyor belts, when you connect machines together, a change in one might cause unintended consequences in another.
(image: Bob Duran / CC BY 2.0)

Machines that are connected have implicit expectations about inputs and outputs. If there's no enforcement mechanism that ensures compatibility between machine interfaces, a problem can easily propagate. These kind of failures happen in the software world all the time, where an unintentional API change can render other programs that rely on the API non-operational.

It's Getting Hot In Here: also be on the watch for changes in environmental conditions. Changes to factors like humidity, temperature, exposure to the elements, etc. can degrade performance leading to—you guessed it—a breakdown.

Spot The Difference

You need to identify the information sources in your workplace that can answer the question, "What's different?" A good place to start is with the people closest to the problem, ask them if they can think of anything that has changed recently. Other repositories may include things like change logs, maintenance records, and reports summarizing work that's ongoing or recently finished. Some companies have procedures that mandate recording the time, personnel involved, and nature of modifications made to key systems. For the digital world, there's typically a logging option for most programs, operating systems, or devices. In software development, diff'ing two versions of a program is a great way to figure out what has changed and will turn up candidates for further investigation.

The best case scenario for the "What's changed?" strategy is this: you will discover a difference, use that as a starting point for your investigation, and eventually be led to the underlying cause. However, it doesn't always work out like that. If you're dealing with a Black Box, your understanding might be limited to "this change is causing the failure." That is, you won't know *why* the change is having the effect it does. Knowledge has a price: the cost of fully understanding cause and effect might be prohibitive. The implication of "spot the difference" is that the change can be *put back* to its former state: that might be your only workaround if the *why* is elusive.

Caveats

I've seen troubleshooters obsess over recent changes with the zeal of an Inquisitor. You've heard the phrase "correlation is not causation" and this is a prime example. Recent changes are a great *starting point* for your investigation, but don't get so fixated on them that you miss other relevant factors. Until proven guilty, they should be treated as coincidental, not definitive proof of a particular cause.

Uncovering information in this vein usually requires interviewing people, so be careful to manage the human side of investigating recent changes. Be sure to have a positive "I'm just trying to understand what happened" attitude, lest you be mistaken for an interrogator. I've seen people fixate on a recent change as a cause for a breakdown and then proceed to adopt a blaming tone. After that drama, frequently the investigation found the cause had nothing to do with the recent change. Oops. Playing the blame game is a bad idea if you want people to be forthcoming in the future. The changes described in the above categories usually result in *unforeseen* consequences (i.e., no one maliciously anticipated that changing X would cause Y to fail). The person who made the decision to make a change was probably acting with the best of intentions. Most people will gladly change their behavior going forward, if they are *calmly* made aware of how their actions were responsible for a breakdown.

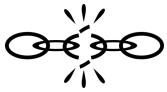
References:

- Header image: Palmer, A. T., photographer. *Salinas Valley, California. A truck being loaded with guayule.* United States, Salinas Valley, California, 1942. Nov. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017860204/.
- [1] Lao Tzu, translated by Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 64.
- Joshua Davis, "Pissing Match: Is the World Ready for the Waterless Urinal?," *Wired*, July, 2010.
- Ian Fletcher, "Space flight can cause astronauts' muscle tissue to waste away by nearly half," *MailOnline.com*, August 18, 2010.
- "'Green' Experiment at City Hall Stinks" *NBC Chicago*. Sunday, Feb 7, 2010.
- Ronald Montoya, "How To Prep Your Car for Long-Term Storage," *Edmunds.com,* July 19, 2011.

What's Changed? was originally published March 5, 2013.

Notes:

Dedicated And Shared Resources



> No program is an island.
>
> **Alex Chaffee**

You used to live by yourself, but then you got some roommates to help with the rent. Now, when you need to use the bathroom at 6 a.m. to get ready for work, they're in there and you're left waiting in the hallway wearing just your towel (not a pretty sight at that early hour, I might add). This is the problem of shared resources, which you will encounter with both roommates and machines.

When machines share resources, sometimes there aren't enough to go around. The ensuing competition is the basic setup for many a troubleshooting problem. As an example within this category, let's look at one of the most common shared resource situations that produce intermittent failures: electrical circuits.

To prevent electrical wiring from carrying a dangerous amount of current, most distribution networks have amperage-limiting breakers (aka, fuses) in place. For the same reason, many machines have fuses within them to prevent drawing a damaging amount of power.

A single machine on a dedicated circuit has the ability to trip a breaker, but that's where its impact ends as far as the electrical system is concerned. However, when you start hooking up *multiple* machines to a common power supply, and thereby create a shared resource situation, the fun begins:

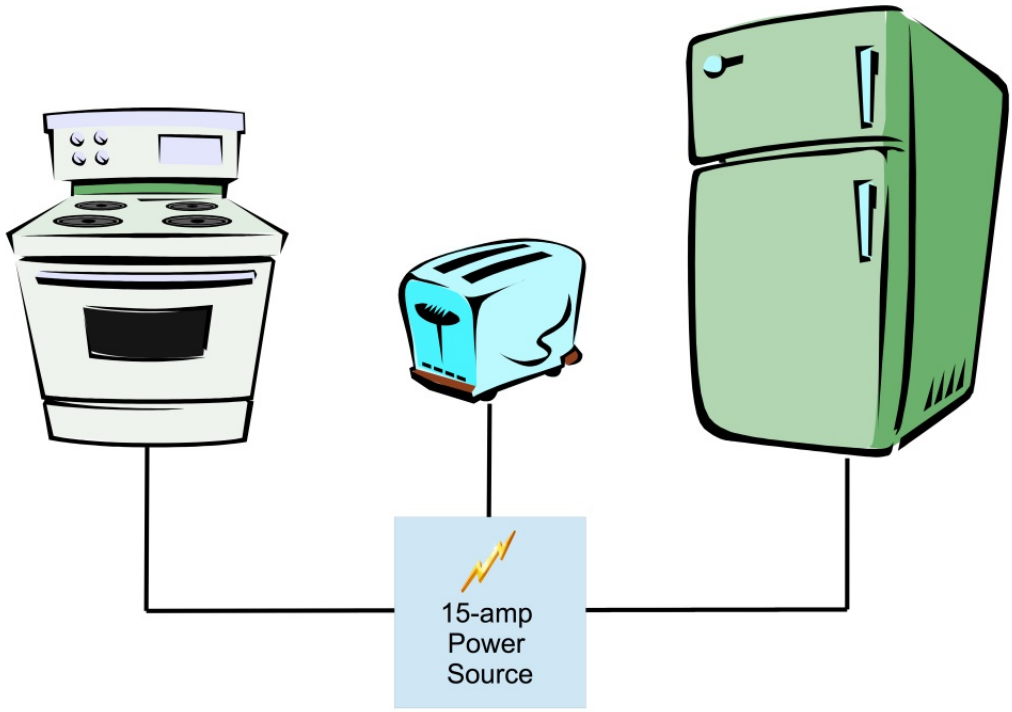


Diagram: kitchen appliances with a common power source.
(image: © Jason Maxham)

Here we've got a refrigerator, toaster, and oven on a shared circuit. Let's say they have the following power consumption characteristics:

Appliance	Average Power Consumption (amps)	Maximum Power Consumption (amps)
Refrigerator	5	8
Toaster	4	9
Oven	6	12

Most electrical devices will have an average and maximum power consumption, depending on the kind of work being performed. These statistics can be misleading though, because your usage of a device might not be "average." For these simple kitchen appliances, it's easy to imagine a variety of circumstances that would vary the power consumption by large margins. For example, if the refrigerator was placed in an unventilated and un-air conditioned room in the middle of a sweltering summer, its power draw would be much higher than in the dead of winter. The oven and toaster will also have large swings in their power consumption based on what kind of work they are doing: the energy required to keep bread warm at 100°F/38°C is much less than cooking a pizza at 500°F/260°C.

You can see from the table that, when operating alone, each of these devices would be fine on a 15-amp circuit: their maximum current numbers are safely below 15 amps. However, when they are all turned on simultaneously, the chance for overloading the circuit becomes a possibility.

Shared resource situations like this can produce intermittent failures. There will be times when these 3 kitchen appliances can be operated simultaneously without incident, and other times when using them together will trip the circuit breaker. If you aren't aware of the amount of power being drawn and the maximum capacity of the electrical circuit being used, the whole situation will seem like voodoo! Of course, the failure condition isn't really intermittent once you know what's going on: if the amount of power being consumed is over 15 amps, the breaker will trip 100% of the time. That's reliable and predictable, the exact opposite of an intermittent problem.

Dedicated To The Job

You probably saw it coming, but a strategy we can use to alleviate an overconsumption situation is to deploy

dedicated resources. Let's stick with our kitchen appliance example: you can see there's no combination of appliances (attached to a single 15-amp circuit) that won't result in an overload, if the maximum current draw is reached simultaneously.

⇤	Maximum Power Consumption (amps) ⇥		Total Consumption (amps)	Potential Overage
Refrigerator	**Toaster**	**Oven**		
8	9	12	29	+14
8	9	—	17	+2
8	—	12	20	+5
—	9	12	21	+6

In the table above you can see the combined effect of these appliances drawing their maximums in various groupings. Depending on which appliances are turned on, the overages range from 2-14 amps beyond what our 15-amp circuit can provide. Someone's dinner is going to get ruined!

If you wanted to completely eliminate the possibility of intermittent power failures in this clip art kitchen, you will need to add a dedicated electrical circuit for each appliance:

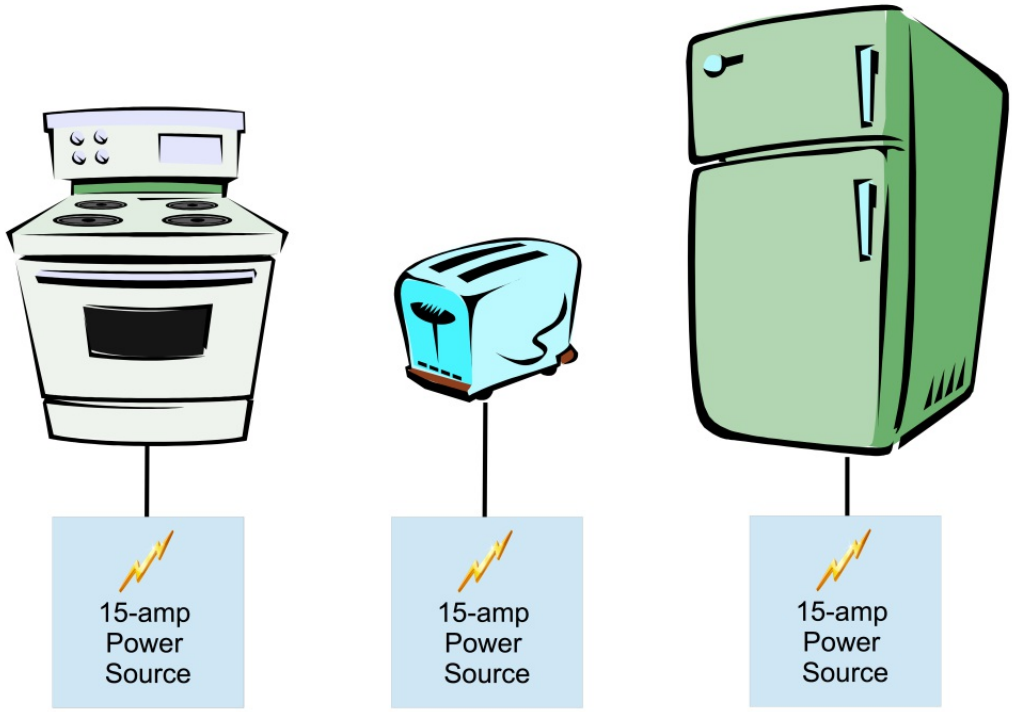


Diagram: kitchen appliances with dedicated power sources.
(image: © Jason Maxham)

Now, each appliance is isolated from the others with its own dedicated power supply. Even if each one simultaneously draws its maximum rated power, there will be no resource conflict. As a bonus, if one of the appliances malfunctions and temporarily tries to draw as much power as it can, beyond its rated maximum, the other two will continue to function.

Getting Your Fair Share, And Priority When Needed

Another option for preventing resource conflicts is to install a governor (also called a limiter) that enforces a certain level of consumption and makes sure that all machines accessing a shared pool of resources gets their "fair share." Once an allocation enforcement mechanism is in place, you can develop more sophisticated schemes to adapt to

situations where a particular situation requires an uneven partitioning of resources.

What has developed in the field of networking is a classic example of this strategy. Think about a busy cafe that offers free Internet access: during peak times there might be a lot of devices (smartphones, laptops, tablets, etc.) sharing this finite resource. Other times, when it's just you and the tattooed barista, you'll have no competition for access to the network. The dilemma is that a typical Internet connection has a fixed amount of bandwidth, but the number of people (and how they use it) will vary widely over the course of a day. I think most cafe owners would prefer that their network automatically adapt to both busy and quiet periods, without having to look over people's shoulders and yell at them to stop watching videos on YouTube.

A group of technologies called Quality of Service (QoS) has emerged that addresses the type of problem faced by a busy Internet cafe. The basic idea is that every device accessing the network gets its "fair share" of the bandwidth. On top of that, some QoS schemes employ a system of prioritization to ensure that people using the network for certain purposes (like making phone calls) get priority access.

For "fair share" use, the basic idea is to take the finite resource and divide it evenly by the number of consumers. If you had 20 people with laptops sharing a 1000 kilobits/second Internet connection, that would be:

1000 kilobits/second ÷ 20 users = **50 kilobits/second per user**

Playing Favorites

You can also favor certain types of usage with a prioritization system. Continuing with our cafe example, let's say we've determined that people use the cafe network primarily for watching videos, backing up files, surfing the web, checking email, and making phone calls. We absolutely never want anyone to have their phone call dropped. Also, we'd like people to be able to quickly check their email when they're in a rush. Web surfing and watching videos are important, but not as important as phone calls or email. Then there's backups, where the files being transferred are large and can take days to transfer over the network. We'll let people do that, but it should never interfere with any of the other uses mentioned previously.

Now that we've determined what's important, we can make the following traffic priority list:

1. Voice calls
2. Checking email
3. Web surfing
4. Watching videos
5. Everything else: backing up files, file sharing, etc.

Within a traffic category, we'll give everyone an equal allocation. Whatever is left gets distributed over the next category in the same way, and so on until all the bandwidth is used up. We'll also add the stipulation that each category is limited to 50% of the remaining bandwidth: this is to preclude a particular traffic category from completely preventing all other uses.

Let's say we have 3 people on phone calls (requiring 120 kbps per call), 2 checking email, 10 web surfers, 4 watching videos, and 5 backup programs running. From our rules, the bandwidth would be allocated as follows:

Remaining Bandwith (kbps)	Traffic Category	Allocation (kbps)	Traffic Category Users	Share Per User (kbps)
1000	—	—	—	—
640	Voice	360	3	120
320	Email	320	2	160
160	Web surfing	160	10	16
80	Video	80	4	20
0	Everything else (backups, etc.)	80	5	16

Starting with our total bandwidth of 1000 kilobits/second, we deduct 3 × 120 kbs per user (360 kbps) for the phone calls. Of the remainder (640 kbps), we've decreed that up to 50% (320 kbps) can be used for checking email. From there, half of the leftover (50% × 320 kbps = 160 kbps) goes to web surfing and so on through the remaining categories, until all the bandwidth is allocated. This scheme definitely slows things down, but it ensures that everyone can get their work done (if you can call sitting in a cafe, sipping joe, "work"). Brownouts associated with one user hogging all the network resources are also prevented. Furthermore, it's *automatic* and *flexible*: the network will continue to function in a variety of conditions (busy and quiet) without any intervention or policing on the part of the cafe owner.

Prioritization and "fair share" allocation are universal tactics that can be applied to any resource that is being exhausted and causing you headaches. Back to our kitchen appliances, let's say that installing additional electrical circuits was not an option. If these appliances had to live on the same circuit indefinitely, we could concoct several schemes to share the electricity. Here are two possibilities (among many) on how to make these machines play nice together:

1. Install a 3-way switch that diverts electricity to only one appliance at a time. This will implicitly prioritize usage: the machine getting electricity will obviously be the one most needed at the moment!
2. Install a current limiting system so that each appliance is restricted to its "fair share" of the available current. If there's one machine turned on, it can use the full 15 amps. If there's two on at the same time, each one can draw a maximum of 15 amps ÷ 2 = 7.5 amps. If all three are turned on, then each is only allowed a maximum of 5 amps (15 amps ÷ 3). Of course the downside is that the ovens may not be able to reach their full temperatures this way. Likewise, the refrigerator may not be able to keep its contents cool if underpowered in certain conditions. However, this scheme will prevent the breaker from tripping and allow the appliances to simultaneously function at a basic level without interruption. Tradeoffs…

"Umm…what do you mean there's only one bathroom?"
(image: James Cridland / CC BY 2.0)

"You're Gonna Need A Bigger Boat"

To show you that I will courageously state the obvious, let me say that another way you can respond to resource shortages is by…adding more resources. *Poof!* Was that your head exploding? Returning one last time to our kitchen appliances example, another possible solution would be to upgrade the capacity of the shared circuit from 15 to 30 amps. With a 30-amp circuit in place, even with each appliance topping out at their maximum consumption, you could operate all 3 simultaneously without a failure:

8 (refrigerator) + 9 (toaster) + 12 (oven) = **29 amps** < 30-amp circuit

As a practical matter, the strategy of increasing the size of a resource pool has its own complications. For example, in North America, most household electrical circuits are sized in very specific capacities. You can't just decide you'd like a 17.9-amp breaker because all the mass-produced parts and building codes are based on 15 and 20-amp circuits. Larger capacities exist beyond that, but the outlets and plug types are different so you'd also need to replace your appliances. In networking, the same thing can happen when you want to go beyond a given capacity boundary. At a certain point, increasing your bandwidth may require you to replace copper cables with fiber optics, consumer-grade switches with professional-quality gear, etc.

Most resource pools are like this: there will be constraints that prevent you from adding an arbitrary amount of capacity that will perfectly suit your needs. Instead, you'll have to add capacity in amounts that are standard to the manufacturer or industry. Often times, stepping up to that next "chunk size" will require costly upgrades that exceed the cost of duplicating what already exists (i.e., adding dedicated resources).

Economics

Cost will usually dictate which resource strategy you pursue. The typical tradeoffs are as follows:

- **Adding dedicated resources:** usually the most expensive option, but it's also the one that guarantees the highest level of stability and reliability. Isolating a machine with its own resources eliminates the possibility of interactions with other machines, so it's ideal for those "mission critical" systems that always need to be functional. Besides the up-front cost, the other downside is the required expansion of your infrastructure. This in turn increases the overhead of your operations. Whatever you're adding (another electrical circuit, another Internet connection, etc.) will require routine maintenance and monitoring on an on-going basis.
- **Installing governors or limiters:** usually the cheapest option, but at the expense of throughput, speed, etc. In our cafe example, a rate-limiting system ensured that everyone had equal access to the network, but at a much slower rate than if everyone had a dedicated Internet connection. For this reason, governors aren't a catch-all solution: the slower rate imposed by them can take you under the minimum threshold required to make your systems or business work.
- **Increasing the pool of shared resources:** depending on what you're adding capacity to, the cost can vary widely with this option. When it's a good deal, the cost will be incremental and proportional to the amount of capacity being added (e.g., doubling the capacity will double the cost). However, as previously noted, there are plenty of examples where capacity can only be added in portions that may be much larger than what you need. Upgrading to the next "chunk size" can mean changes in your infrastructure that may exceed the cost of simply adding dedicated resources. The upside to shared resources is better economics and reduced overhead versus managing many small dedicated pools. That's because it's easier to worry about *one* thing versus many things!

References:

- Header image: Carl Nenzen Loven, photographer. Retrieved from Unsplash, https://unsplash.com/photos/N8GdKC4Rcvs.

Dedicated And Shared Resources was originally published March 22, 2013.

Notes:

A Common Problem

> I try to isolate the symptoms of whatever the problem is. Okay, email isn't flowing. But, is it just to certain addresses? Just outbound? Just inbound? Understanding what's being affected and what's NOT being affected is really important.
>
> **Austin Quade**

Maybe you've had that common recurring dream where you forget to wear pants: to school, to work, or anywhere outside the privacy of your home. If you're the *only one* not wearing pants and stalking around in public in just your underwear, it's pretty embarrassing. However, if *everyone* isn't wearing pants—now that's a party! (Why can't I have *that* dream?)

When it comes to wearing pants, *context matters:* dressing appropriately requires knowing the situation. Reading the environment also benefits troubleshooting and is crucial to choosing the appropriate strategy. You'll need to understand how the parts relate to the whole, especially when a machine is installed within an interconnected web of supporting systems. As we'll see, knowing whether machines share symptoms will give you a wealth of additional information that can help pinpoint the source of a problem.

Properly assessing the surroundings ensures that you're troubleshooting at the correct level: system-wide problems need to be solved system-wide. Lifting your head up and realizing an issue spans multiple systems will ensure that you don't waste time trying to fix a problem on an individual basis.

Things We Have In Common

Symptoms that are shared across machines can help you to quickly isolate the cause of a problem. Let's look at an example from the field of networking which will illustrate the power of tuning into shared symptoms. Consider a public library that wants to provide free Internet access to its patrons. Two banks of computers are set up for this purpose:

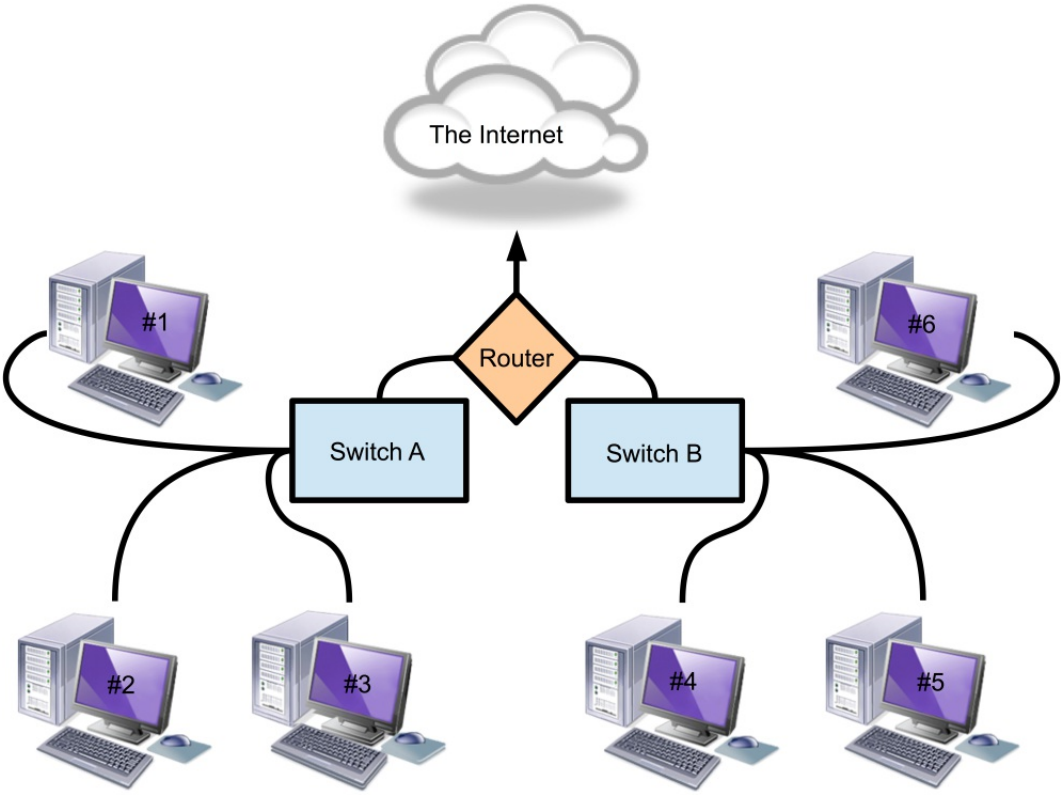


Diagram: computers connected to the Internet via two switches (A,B) and a router.
(image: © Jason Maxham)

At first, everything works fine, but one day people using computers #1-3 complain that they can't access the Internet. You verify these claims while also noticing that the patrons sitting at computers #4-6 continue to happily surf away:

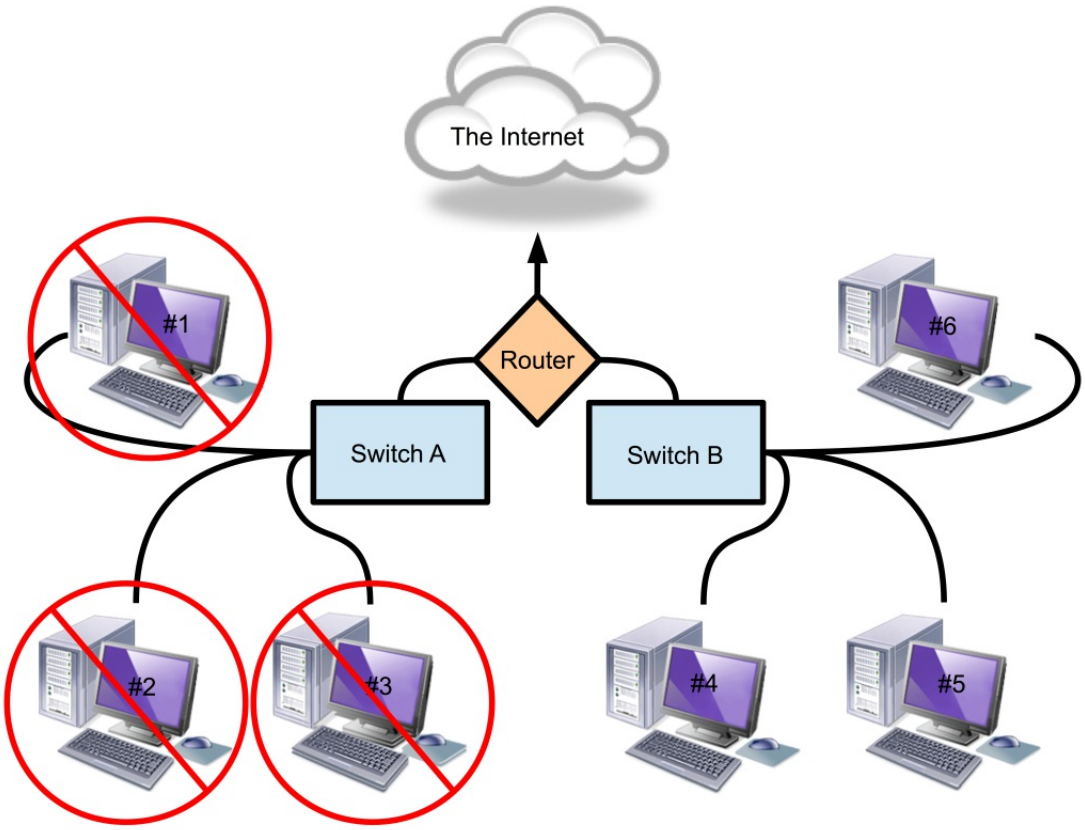


Diagram: partial network outage.
(image: © Jason Maxham)

Hmm…what do the computers experiencing the outage (#1-3) have in common? Likewise, how are they different from the computers that can still access the Internet (#4-6)?

Looking at the diagram, the answer to the first question is: Switch A and the cable that connects it to the router. That path to the Internet is shared by all the computers experiencing the problem (#1-3). Examining Switch A, you see that the power cord came loose and the switch is off. You plug it back in and computers #1-3 are able to access the Internet once again!

In our library computer room, different groupings of shared symptoms will point to various components as the source of a problem. For instance, if *all* of the computers (#1-6) couldn't access the Internet, then the router and the Internet connection itself would be the best candidates to investigate: these are the only things that are shared by all the computers. Conversely, if only one computer couldn't access the Internet, there's a good chance the problem is isolated and internal to that specific machine. By the way, I use the word "candidate" deliberately: noticing shared symptoms is a way to accelerate learning about a failure, but it's not definitive proof of anything. If *all* the computers can't access the Internet, there are a variety of possibilities that are logically consistent with this scenario:

- The Internet connection and/or router is down.
- Switches A+B are **both** down. As all of the computers are attached to one of these two switches, if they were to malfunction at the same time, it would have the effect of knocking out access to all of the computers.
- Each of the computers (#1-6) is independently misconfigured: perhaps with a virus or errant network settings.
- Any simultaneous combination of the above factors.

Each of the above scenarios could be true, but the power of the "shared symptoms" strategy is that, most of the time, the source of the problem will be the thing held in common. If all 6 computers can't access the Internet, the law of simplicity says it will be a *single* cause the majority of the time. To understand why, let's think about the problem statistically and count the number of things that need to be simultaneously wrong for the various scenarios to be true (again, we're considering the case where all 6 computers are unable to access the Internet):

# Failures	Router	Switches A+B	All 6 Computers Properly Configured

0	OK	OK	OK
1	FAIL (1)	OK	OK
2	OK	FAIL (2)	OK
3	FAIL (1)	FAIL (2)	OK
6	OK	OK	FAIL (6)
7	FAIL (1)	OK	FAIL (6)
8	OK	FAIL (2)	FAIL (6)
9	FAIL (1)	FAIL (2)	FAIL (6)

Here, I've listed the various possibilities and added up the number of simultaneous failures associated with them. Look at the story described in the bottom row, which tallies **9** simultaneous things wrong: the router (1), both switches (2), and all the computers (6). This is a *possible* failure condition, but think about how *unlikely* it would be: the probability of it happening is the product of all 9 scenarios multiplied together! The message rings out loud and clear: in situations with symptoms shared across systems, the odds favor the cause of Just One Thing. Other possibilities require increasingly unlikely coincidences of simultaneous failures.

For bed or lying on the couch? Yes. For a job interview? Not so much.
(image: Mark Baylor / CC BY 2.0)

Wearing A Belt And Suspenders

Failures that involve shared symptoms will expose **dependencies** that may not have been obvious when you first deployed a system. Sticking with our library example, you might not have realized that plugging computers into a common network switch would mean an outage for all the attached systems when that same switch malfunctions. Now you know! Within that setup, the computers are *dependent* on the switch for a path to the Internet. Mitigating the negative consequences of dependencies are necessary if you want to create systems that are resilient to failures.

Let's say that our library is expecting a visit from a very important donor and celebrity. The library's public relations director has even scheduled a photo op with members of the press that will include our VIP using the Internet. If the Internet were to be inaccessible during the photo op, it would be a huge embarrassment. With so many people milling

about, the chance of a power cord being kicked loose is high, so we want to eliminate the chance of a switch failure knocking out access. To make our system more robust, we decide to connect each computer to both switches:

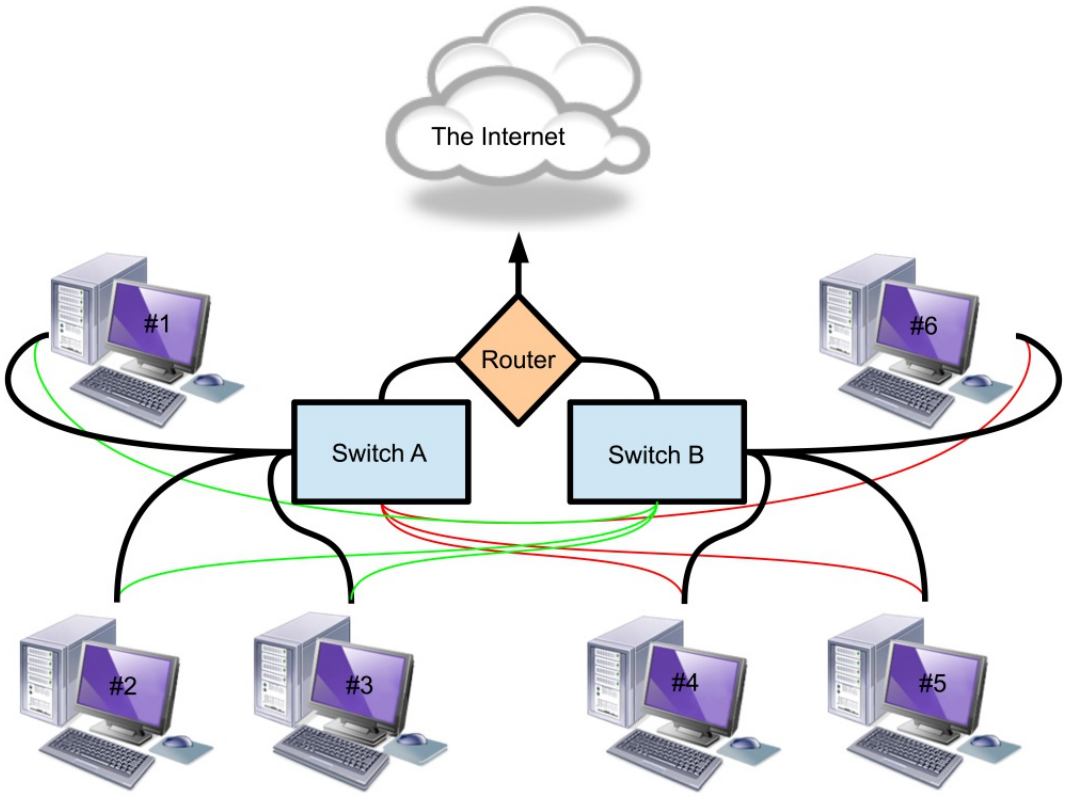


Diagram: computers with redundant network connections (green and red lines).
(image: © Jason Maxham)

Now, each computer has two network connections: one to Switch A and one to Switch B. When properly configured to use the extra connection, each computer will have two separate paths to the Internet: one that passes through Switch A and another that passes through Switch B. We have begun the process of eliminating what are known as **single points of failure**: parts of the system that, when they stop working, cause the system as a whole to fail. You can see that, even after this upgrade, two other single points of failure remain: the router and the Internet connection. How far you go to make your systems highly available is an economic decision. Perhaps the redundant network connections only require an extra set of cables (a small, one-time cost), but another Internet connection will mean an additional router and monthly service charges. How much money we choose to spend on this reliability project all depends on how bad it would be to let down our VIP.

Finally, you might have noticed that the "shared symptom" strategy is closely related to the "shared resources" problem we previously discussed. The distinction is that here we're not talking about competition for resources between machines, but rather their availability to the system as a whole. When Switch A failed in our example, it wasn't because one machine was hogging all the network bandwidth. In fact, how the switch failed (from a power loss) was completely independent of how the computers were using the network. The "resource" in jeopardy was a suitable path to the Internet.

Spreading The Sickness And The Cure

There's another angle from which we can view the "shared symptoms" strategy. Up to this point, I've discussed it as information used to pinpoint the location of a failure. However, we can go one step further and *actively* attempt to create the shared symptoms.

Let's return again to our library network, to the point in time where we discovered that computers #1-3 were unable to access the Internet. This time, however, let's say that Switch A is on and appears to be functioning normally, at least based on its indicator lights. What's a quick way to verify that Switch A is the problem? A clever hack would be to take a computer from the group that's working (#4-6), plug it into Switch A, and see what happens. Let's change computer

#4's connection from Switch B to Switch A:

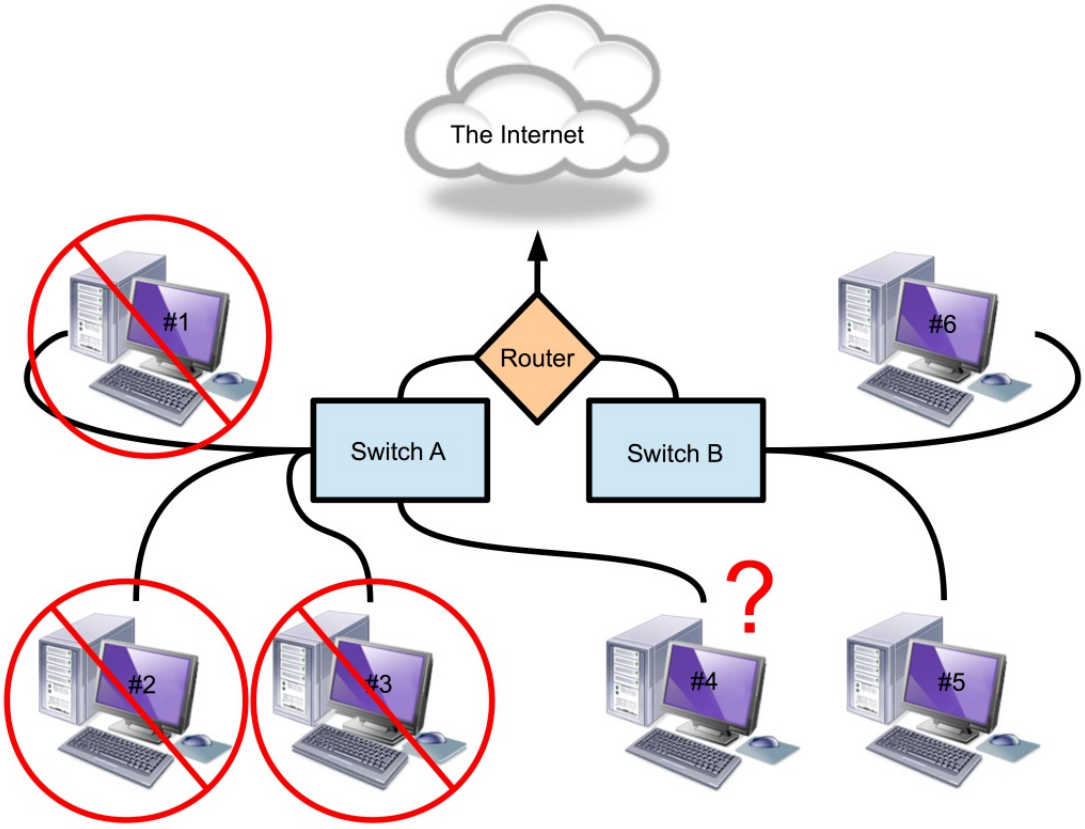


Diagram: we disconnect computer #4 from Switch B and move it to Switch A, then check its status.
(image: © Jason Maxham)

After we connect computer #4 to Switch A, we find that it can't access the Internet. Good job! We've taken something that worked, made only a single change, and now it doesn't work. We've succeeded in moving the symptom and have learned something valuable in the process. Now, the evidence that Switch A is the culprit is very strong indeed! The converse would have worked as well: we could have taken one of the computers that couldn't access the Internet (#1-3), connected it to Switch B, and observed the results. We could call this opposite method "spreading the cure" and, for destructive symptoms, that is the preferred way!

One last possibility for this situation is to use a related technique called **"moving the problem"**: that's where you take a suspicious component, move its position in the system, and see if the problem follows. In this context, that would mean taking Switch A and putting it in Switch B's place. If Switch A is the cause of the failure, then it should create problems wherever it goes.

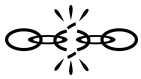
References:

- Header image: *"Funicular"*. Magda B, photographer. Retrieved from Unsplash, https://unsplash.com/photos/xlRFH9KYyyg.

A Common Problem was originally published April 14, 2013.

Notes:

Clear Up To Here



> Can I draw a box around the problem? I use the visual image of drawing a box around it and saying, "My entire problem exists within this box." It's very useful because then you can start drawing smaller boxes, or chopping boxes in half.
>
> **Karl Kuehn**

When troubleshooting, the goal is to efficiently restore the *need* a machine once served. Opportunity costs constrain how much can be learned about the cause of a failure and justify the reasons for *not* fixing something: finding a workaround, replacing instead of repairing, etc. As noted in the section on economics, just figuring out what is wrong can, by itself, weigh heavily on a repair's total cost. Therefore, it's no coincidence that many troubleshooting techniques are geared towards knowing as much about a problem with as little effort as possible.

It's within this context that I bring up the twin concepts of **narrowing** and **isolation**. When combined, they aim to minimize the cost of learning about a failure. After you define what the problem is, the next question invariably will be: "Where is the problem?" The answer, "in a million possible places," is not very reassuring if you're footing the bill for the repairs. By approximating where the malfunction lies, the cost of repair can begin to be estimated.

Different parts of a system will have varying levels of accessibility, complexity, and costs associated with their repair. Many systems are built in a modular way, which can favor swapping over repair. A good illustration is to think about the common scenario of a light that won't turn on. Once you've narrowed the problem down to the light bulb itself, you don't contemplate repairing it. Why did it burn out? "Who cares?," you say as you twist a new one into its place and move on with your life. Alternatively, if the bulb is okay and instead your narrowing places the problem within the electrical system, now you're facing a different cost calculation. The fix may be trivial or major but, if you're not an electrician, you're going to need to hire someone. If you don't have the money to do that, you might decide to set up a portable lamp until you have the wherewithal to hire someone to look at the problem.

As strange as it might seem, troubleshooting is often concluded without ever discovering the precise cause of a failure. However, knowing the general location of a problem speeds the decision-making process; often, this alone is all you need to choose the right course of action.

If you cut the problem in half, it's easier to handle…
(image: chichacha / CC BY 2.0)

I Just Want Half

Continuing with our theme of "Where is the malfunction?," let me introduce an amazing technique that can quickly pinpoint the location of failures in interconnected systems. It's called **"half-splitting"** and its name is well-chosen: by splitting your problem space in half multiple times, you can swiftly isolate a failure.

I'll give you a verbal description first, but don't fret if half-splitting doesn't sink in from these words alone. Shortly, we'll look at an example with visuals that will aid your understanding. To start with, we need a certain kind of system: specifically, one that has a definite middle. If you can't approximate the mid-point between the two boundaries, well there goes the whole "half" concept out the window! You can do half-splitting on just a portion of a chained system, but you need to choose a discrete beginning and end to bound the process. Linear systems that have a "flow" and move stuff from point A to B are well-suited for half-splitting. The "stuff" being carried or transformed can be water, electricity, fuel, information, vehicles, products—when you think about it, you realize these type of systems are everywhere: pipes, wires, assembly lines, roads, computer programs, data networks, canals, etc.

We also need a way to examine the state of the flow as it goes from beginning to end. Starting in the middle (or as close as we can get), we'll inspect for the error. Depending on whether the error is present, we'll either look forward or backwards along the direction of flow. Each time, we'll split the remaining "unknown" part of the chain in half. As we iterate, each inspection halves the remaining possibilities for the location of the error. As you'll see, on average this technique confers a huge advantage over inspecting every chain link one-by-one.

But enough of my yakking! My description of half-splitting is likely not enough to understand how it works in practice, so let's look at an example. Let's consider an automated assembly line which decorates teddy bears. An unfinished teddy bear enters the line and, at various stations along the way, is decorated and dressed:

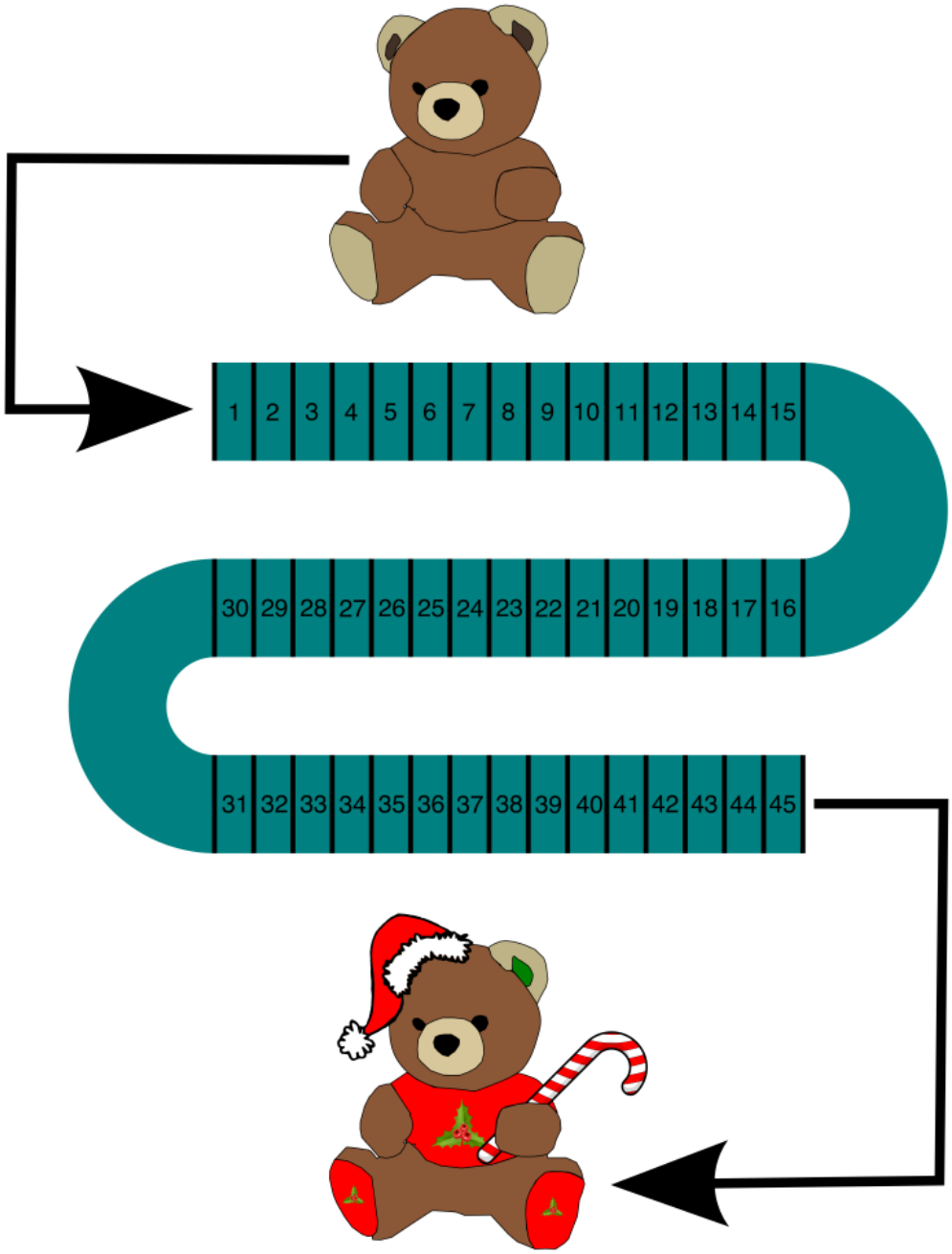


Diagram: an assembly line that decorates teddy bears.
(image: © Jason Maxham)

Today, something has gone terribly wrong: the unfinished bears go in one end, and out comes a pile of junk on the other!

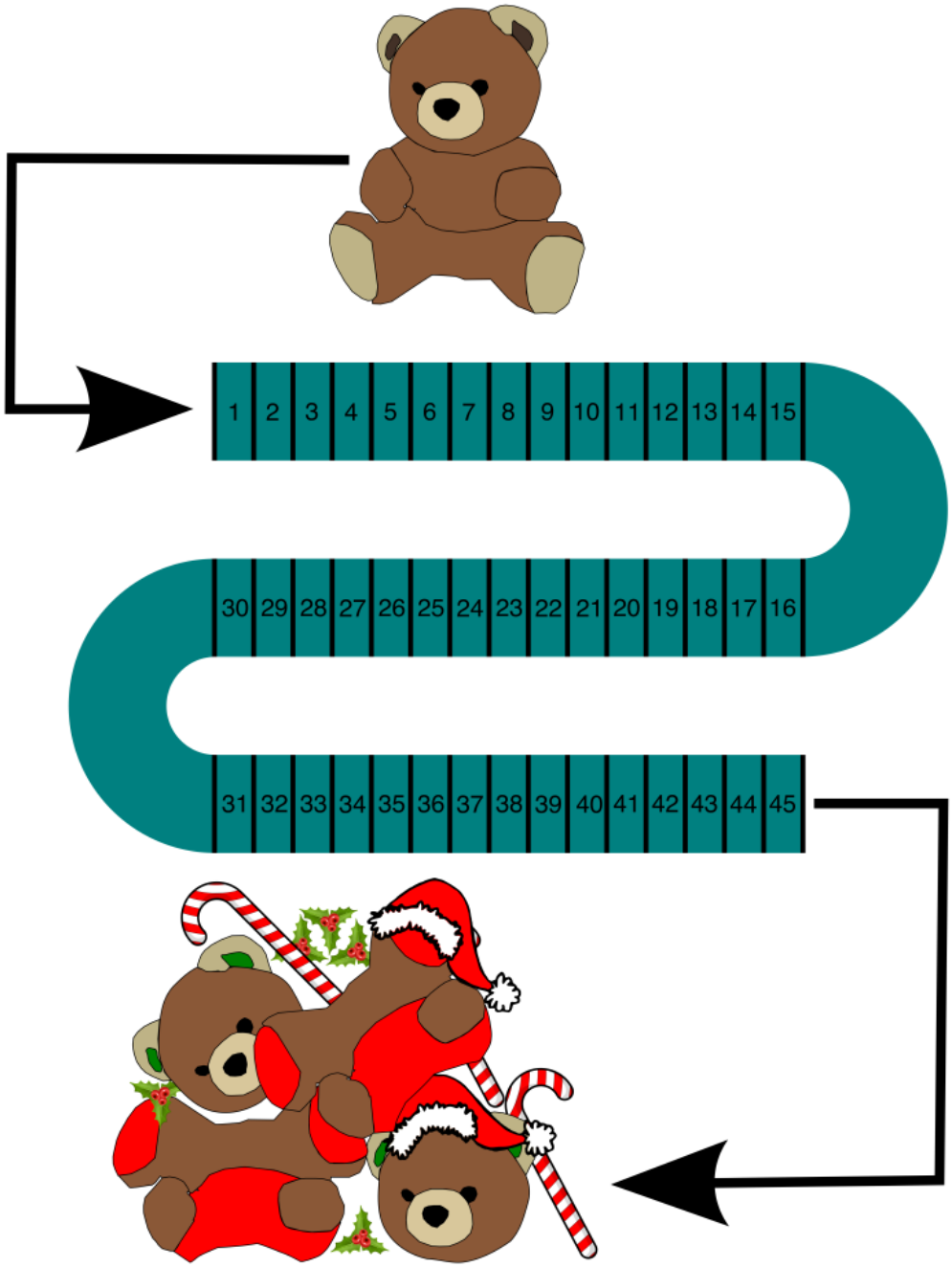


Diagram: the assembly line malfunctions, destroying each bear that enters, ejecting a pile of debris on the other side.
(image: © Jason Maxham)

Clearly, something is severely malfunctioning and destroying our teddy bears. But where is the problem? There are a lot of stations to check on the assembly line: 45 in all. We need the teddy bear assembly line up and running ASAP—after all, fake fur is money. To add to our difficulties, the assembly line is completely covered and therefore we can't see what's going on inside. There are access hatches above each station, but going through the line and checking all of them one-by-one is going to take a long time. Instead, we'll use the half-splitting technique to speed up the process, find the cause, and get the line rolling again.

First off, we want to divide the assembly line into 2 equal parts. There are 45 stations, so the middle station is #23 (there are 22 stations on either side of #23). We open the hatch over station #23 and take a look:

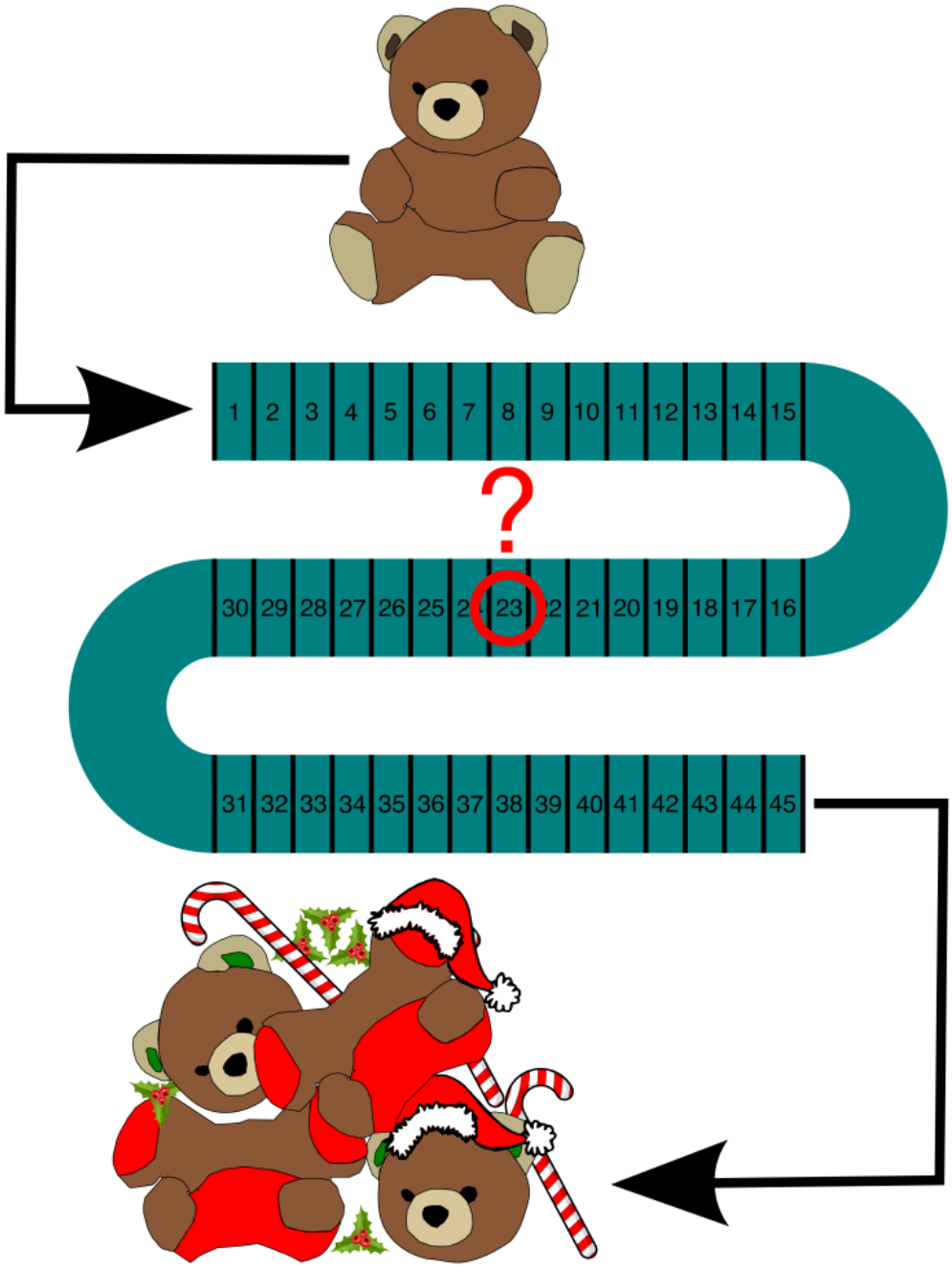


Diagram: half-splitting the problem means we start our investigation in the middle. What's happening at station #23?
(image: © Jason Maxham)

At station #23, the teddy bears are fine. The assembly line flows from station #1 to station #45. By verifying the correct operation of station #23, we know the problem lies further downstream. Think about what we just did: in one fell swoop, we eliminated stations #1-23 as the cause of our bear tragedy. That's a very efficient use of our time!

Now, we know that the problem lies somewhere within stations #24-45. So, let's half-split again: the middle station between #24-45 is #35. Inspecting #35, we find good bears once again. Hooray again for our potent efforts: another 12 stations (#24-#35) were eliminated in a single action! We're on a roll, so we'll continue splitting and inspecting until we find the cause. This table shows our progress as we narrow in on the cause:

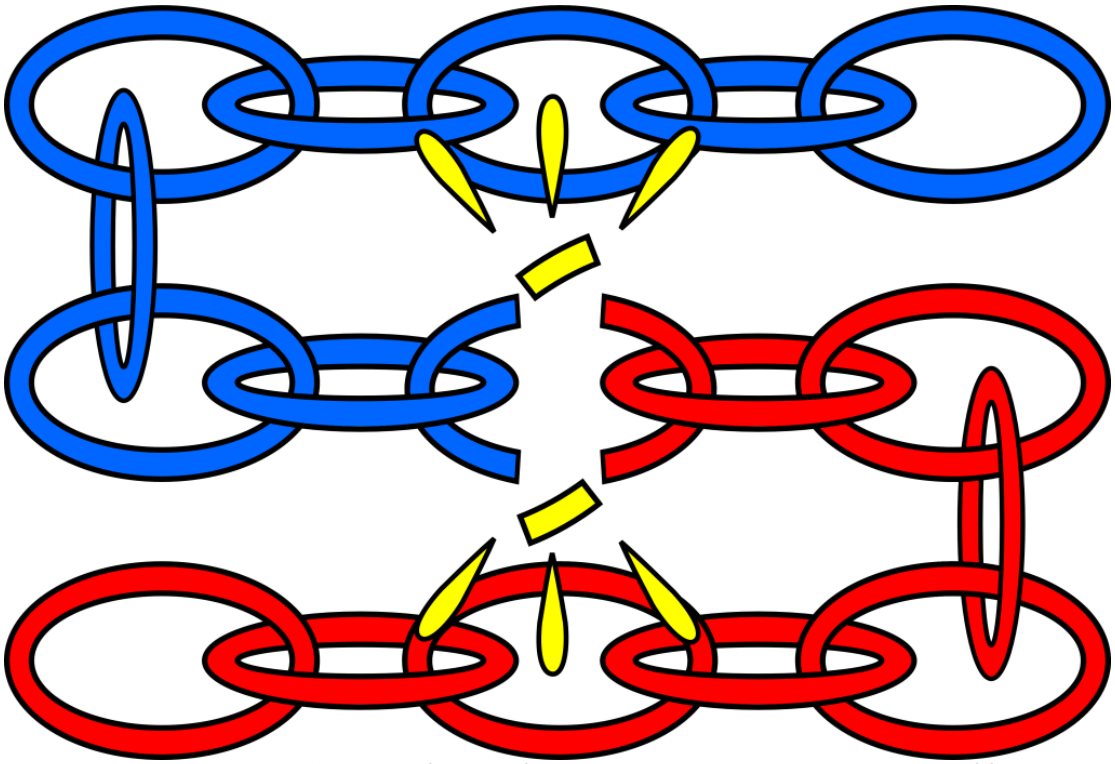
Iteration	Lower Bound	Upper Bound	# of Remaining Possibilities	Middle	Middle Status	Next Split
1	1	45	45	23	OK	→
2	24	45	22	35	OK	→
3	36	45	10	41	FAULT	←
4	36	40	5	38	FAULT	←

5	36	37	2		37	FAULT ←
6	36	36	1		36	OK

A summary of what we did: after our initial two splits, we knew the problem was between #36-45 (the lower and upper bounds listed for iteration #3). We examined the middle station (#41), and found a mangled pile of bear parts! Whenever you find a failure, you know that the problem must lie **at that station or before,** and therefore the next split you make will be backwards. So, that's what we did in iteration #4: we split backwards and examined #38 (which also turned out to be a failing station). This leaves only 2 stations (#36 and #37), and we look at both of them because we want to verify the transition point from working to failed. Station #37 has a pile of things that used to be a teddy bear, while station #36 is fine. We've found **a working station followed by a failing station,** and that means we've found the culprit: station #37 must be the one wreaking havoc!

In any chained system, finding a working node immediately followed by a failing node is the jackpot. Half-splitting accelerates this process, and the table shows the very quick narrowing of the range in which the error was located: 1-45 → 24-45 → 36-45 → 36-40 → 36-37 → 37. As a consequence of this, the number of remaining possibilities dramatically drops by half after each iteration: 45 → 22 → 10 → 5 → 2 → 1. It only took us 6 steps to find the error, whereas if we had gone serially from the beginning until the error was discovered, it would have taken 37 steps!

To clarify the process, when there is a single "weak link in the chain," the goal is to find the **first failing node.** In these type of systems, everything before that first failing node will work, and everything after it will not. Here's a visual representation of how this works:



Many chained systems behave like this: when a link fails, the failure will propagate along the direction of flow. Abstractly, it looks like the image above, with the blue links representing the working part of the system. After the first broken link, the system fails to work: these are the red links.

(image: © Jason Maxham)

Half-splitting vs Serial Search

Half-splitting will quickly identify that first failing node in a chain. The larger your system, the bigger the payoff in terms of time saved. Let's see how half-splitting and serial search measure up in systems of various sizes:

Chain Length	Total Inspections		Average Inspections		Advantage
	Serial	Half-splitting	Serial	Half-splitting	
2	3	4	1.50	2.00	SERIAL
3	6	6	2.00	2.00	EVEN
4	10	10	2.50	2.50	EVEN
5	15	14	3.00	2.80	HALF-SPLITTING
6	21	18	3.50	3.00	HALF-SPLITTING

7	28	21	4.00	3.00	HALF-SPLITTING
8	36	26	4.50	3.25	HALF-SPLITTING
9	45	31	5.00	3.44	HALF-SPLITTING
10	55	36	5.50	3.60	HALF-SPLITTING
15	120	60	8.00	4.00	HALF-SPLITTING
20	210	90	10.50	4.50	HALF-SPLITTING
50	1,275	288	25.50	5.76	HALF-SPLITTING
100	5,050	674	50.50	6.74	HALF-SPLITTING
1,000	500,500	9,978	500.50	9.98	HALF-SPLITTING
10,000	50,005,000	133,618	5,000.50	13.36	HALF-SPLITTING

To create this table, I wrote a computer program to generate all the possible initial error positions for a chain of a given length (if you're interested, the source code is included in the notes at the end). Before we get into that, let's review what we know about chain-like systems. I use the term "chain-like" to mean any system where the output from the first node is fed into the second, the second into the third, and so on until the end. As noted in "Follow The Chain," errors will typically propagate along a chained system in two different ways:

- The output at the end of the chain will be flawed in some way.
- There will be no output.

The predictable, one-way flow of these kinds of systems confers an advantage for troubleshooting: when you find an error, you know that it must have originated somewhere upstream!

In the table, you can see that the serial and half-splitting methods are evenly matched at the start. Serial search even wins the first round (systems with 2 elements), and then it's a tie for 3-4. However, after the systems grow to 5 elements, half-splitting starts to pull away. At 15, half-splitting finds the cause twice as quickly. By the time we get to 1,000 nodes, half-splitting's advantage is overwhelming: 50 times faster! Remember, these aren't just numbers, they're how much time it would take you to find the cause. Interested in working 50 times as long?

The Tipping Point: 5 Elements

To get a better understanding of how the two methods compare, let's examine a chained system with 5 nodes. This is a good example to look at because it's the point where half-splitting gains the advantage over serial search. Even though the average number of inspections are nearly the same (2.8 vs 3) for this number of elements, there are some important differences I want to point out. Let's start by listing all the possibilities for the location of a single error in a system with 5 elements, and see how the error propagates down the chain for each:

Scenario #	Node 1	Node 2	Node 3	Node 4	Node 5	Failing Node
1	FAULT	FAULT	FAULT	FAULT	FAULT	1
2	OK	FAULT	FAULT	FAULT	FAULT	2
3	OK	OK	FAULT	FAULT	FAULT	3
4	OK	OK	OK	FAULT	FAULT	4
5	OK	OK	OK	OK	FAULT	5

For the serial search method, you'd start at node #1 and search up the chain until you find an error. For possibility #1, you can see that you'd find the error right away, with just a single inspection (because the error is at node #1). For possibility #2, it takes 2 inspections, 3 for possibility #3, 4 for possibility #4, and 5 for the last possibility. Using the half-splitting method on a system with 5 nodes, you always start at node #3 (the middle) and then move forwards or backwards, based on what you find. Here's a table comparing the number of inspections taken by half-splitting and serial search for all of the possible error locations:

Fault Position in a 5-node Chain	Serial Inspections	Half-splitting Inspections
1	1	3
2	2	3
3	3	2
4	4	3
5	5	3
Total	**15**	**14**
Average	**3**	**2.8**
Median	**3**	**3**
Variance	**2.5**	**0.2**

Going over the statistics, you can see that the totals and averages are nearly the same, while the median is identical. What's remarkably different is the **variance,** which is a statistical measurement of sprawl for a group of numbers: the more spread out they are, the higher the variance. It's easy to see why the variance is higher for serial search: its range of inspections is 1-5, while the range for half-splitting is 2-3.

Let's imagine a case where the inspection of each element in our 5-element system takes one hour to complete. That would mean the serial search method could take anywhere from 1 to 5 hours to complete. Half-splitting, in contrast, would never take more than 3 hours. That kind of predictability could be highly prized among your customers.

Based on the data I generated, my recommendation is to use serial search when the number of elements is 4 or less. Serial has other advantages at this level: it's easier to execute and keep track of the process. However, if controlling the variance of repair times is a big concern for your particular situation, then only use serial search for systems with 2 elements (and half-splitting for all the rest).

The Math Behind Half-splitting

When using the half-splitting method, the approximate number of times you'll need to divide your system and inspect an element is described by this formula:

$log_2 n$

Where *n* is the number of elements in your system. Logarithmic functions grow slowly (compared to linear functions), and are considered the standard when writing scalable algorithms in the world of computer science. You get a good sense of the power of logarithmic functions when you look at really large data sets. Consider how quickly half-splitting finds the error in a chained system of 1,000,000 components:

$\log_2(1{,}000{,}000) =$ **19.93**

Only about 20 steps: that's efficient! Contrast this with the formula for the average number of steps required by the serial search method:

$(n + 1) \div 2$

Again, *n* is the number of components in your system. A downside for the troubleshooter is that this function grows linearly with the size of the chain. That means that a serial search for an error in a chain of one million components would take:

$(1{,}000{,}000 + 1) \div 2 =$ **500,000.5**

That's right, on average about a half-million steps!

The Right Stuff

Half-splitting requires a system where you can inspect any single element and, based on its state, know where to look next. That's why it's especially suited for transformational or logistical chains: assembly lines, data networks, computer programs, pipelines, electrical circuits, etc. In our teddy bear example, we could examine a particular station on the assembly line and, based on the condition of the bear, know whether the problem was upstream or downstream.

In the field of computer science, the same technique speeds lookups in long lists of sorted information (in computer science, the method is known as "binary search"). Opening a sorted list and inspecting the middle entry, you call tell if the piece of data you're seeking is forwards or backwards in the list. Again, the key is knowing where to look next based on any one element.

The Wrong Stuff

You can't split every problem in half and get the efficiencies described here: as noted, the ability of a single element to tell you the direction of the problem is the key for half-splitting. If you were working on a broken car, you wouldn't gain anything from saying: "Let me split this problem in half, I'm going to see if the failure is on the right-hand side of the car." Inspecting a random windshield wiper or a spark plug on the right-hand side of a car won't tell you in which direction the problem lies. A car, taken as a whole, is a collection of independent subsystems: in that sense, there's really no "middle" to split! Some of the chain-like subsystems on a car (fuel, electrical, etc.) would benefit from half-splitting, but only after other strategies have identified them as candidates for further investigation.

Notes and Computer Code:

PlanetMath's page on the binary search algorithm says the "average-case runtime complexity" is approximately:

$log_2 n - 1$

The context given for the PlanetMath wiki page is dictionary lookups, which is similar to the use of half-splitting when troubleshooting: each iteration halves the number of possibilities leading to big efficiencies as the size of the data set grows. However, the use isn't exactly the same and I wanted to compare actual numbers. I needed a way to generate real data comparing both methods in the context of troubleshooting. The data I generated was very close to the "$log_2 n - 1$" approximation. However, I found half-splitting for the purposes of troubleshooting is closer to just $log_2 n$. It would be cool to have a formula that gave the *exact* answer! I'm not a mathematician, so I'm not sure if this is even possible. Any takers to figure it out?

For the half-splitting program I wrote, it took a while for me to figure out the correct algorithm to detect the position of the initial error. For

my first few tries, my implementation took a lot more steps than what I considered to be the ideal. The breakthrough came when I noticed that, if the last and current "middles" chosen by the algorithm converged, the answer had been found. This insight replaced a tangled mess of "if" statements in my code. I also made the decision to record the *unique* number of times the program examined a particular chain element. This brought the number of inspections in line with what I had experienced doing the algorithm by hand on paper. I also think this better simulates what a person would experience: specifically, in real life, anytime you notice a "WORKING" element followed by an "ERROR," you know you've found the failing node. With the "converging middles" solution, sometimes the computer will take an extra step that a human wouldn't. You could probably code this intelligence into the program, but I liked the simplicity of the "converging middles" code and so I decided to leave it alone. If you have a better solution, feel free to post it!

Anyway, here's the program I wrote (in ruby) to simulate and compare the serial and half-splitting methods of troubleshooting:

```
#serial_vs_half_splitting.rb
#Author: Jason Maxham, https://artoftroubleshooting.com/
#Purpose: to simulate and compare troubleshooting of "chain-like" systems
#(assembly lines, pipelines, networks, etc.) with both the serial and
#half-splitting methods. May the best algorithm win!

def create_chain(chain_length, first_error_position)
  chain = []
  for chain_position in 1..chain_length
    if chain_position < first_error_position
      chain.push("WORKING")
    else
      chain.push("ERROR")
    end
  end
  return chain
end

def find_error_by_half_splitting(chain)
  #we need to set and keep track of these variables:
  lower_bound = 0
  upper_bound = chain.length - 1
  last_chain_middle = ''
  #keep track of the elements we've inspected to compare to the serial method
  inspections = {}

  #do the first split and inspection
  range_length = upper_bound - lower_bound
  chain_middle = (range_length.to_f / 2.to_f).ceil
  inspections[chain_middle] = 1

  while 1 do
    if (chain[chain_middle] == "WORKING")
      #error is forwards in the chain: split forwards, set the
      #lower bound to the current middle element plus 1
      lower_bound = chain_middle + 1
    else
      #error is backwards in the chain: split backwards, set the
      #upper bound to the current middle element minus 1
      upper_bound = chain_middle - 1
    end
    inspections[chain_middle] = 1

    #set the last_chain middle to the current one
    last_chain_middle = chain_middle
    #calculate new middle
    chain_middle = ((lower_bound + upper_bound).to_f / 2.to_f).ceil

    #It's all over when: the current and previous middles converge
    if (last_chain_middle == chain_middle)
      return inspections.length, last_chain_middle + 1
    end
  end
end

def find_error_serially(chain)
  inspections = 0
  first_error_position = 0
  chain.each_with_index do |element, index|
    inspections += 1
    if element == "ERROR"
```

```
      first_error_position = index + 1
      #stop, we've found it!
      break
    end
  end
  return inspections, first_error_position
end

def generate_chains(chain_length)
  #we'll keep track of the number of times that the serial and half-splitting
  #methods find a different initial error. NOTE: this should never happen,
  #but it was useful when I was debugging!
  num_errors = 0

  #also keep track of the number of times we looked at an element in the chain.
  #the point of this exercise was to see how much more efficient
  #half-splitting is versus serially searching through the elements
  #of a real system when troubleshooting.
  total_serial_inspections = 0
  total_hs_inspections = 0

  #create chains, with the initial error in every possible position
  for initial_error_position in 1..chain_length
      chain = create_chain(chain_length, initial_error_position)
      (serial_inspections, serial_error_position) = find_error_serially(chain)
      (hs_inspections, hs_error_position) = find_error_by_half_splitting(chain)
      unless (serial_error_position == hs_error_position)
        num_errors += 1
      end
      total_serial_inspections += serial_inspections
      total_hs_inspections += hs_inspections
  end
  return num_errors, total_serial_inspections, total_hs_inspections
end

def serial_vs_half_splitting
  total_errors = 0
  total_serial_inspections = 0
  total_hs_inspections = 0

  calculate_these = (2..20).to_a
  calculate_these.push(50,100,1000,10000)

  calculate_these.each { |chain_size|
      (errors, s_inspections, hs_inspections) = generate_chains(chain_size)
      puts "#{chain_size}\t#{s_inspections}\t#{hs_inspections}\t#{(s_inspections.to_f / chain_size.to_
      total_errors += errors
      total_serial_inspections += s_inspections
      total_hs_inspections += hs_inspections
  }

  puts "SERIAL AND HALF-SPLITTING DIDN'T FIND THE SAME ANSWER: #{total_errors} TIMES"
  puts "SERIAL SEARCHING REQUIRED #{total_serial_inspections} INSPECTIONS OF CHAIN ELEMENTS"
  puts "HALF-SPLITTING REQUIRED #{total_hs_inspections} INSPECTIONS OF CHAIN ELEMENTS"
end
serial_vs_half_splitting
```

References:

- Header image: Mason Kimbarovsky, photographer. Retrieved from Unsplash, https://unsplash.com/photos/qEwJFHU3uOE.
- PlanetMath, Binary Search.

Clear Up To Here was originally published April 26, 2013.

Notes:

Team Spirit

> I form my opinion, but I also want to tap into other people's past experiences. Because I've seen people work a long time, only to have someone come up and say, "I had that problem last week, do this!"
>
> **Dan McCormick**

Troubleshooting is often a solitary exercise: the cost of labor favors the lone wolf problem-solver. The cable guy is usually—just one guy. I've never called a plumbing service and had a whole team of plumbers show up. Especially when repair work is an unreimbursed cost for a company (e.g., on-site warranty service), it'll likely be just one person on the call.

I've seen "collaborative environments," where troubleshooters work in close proximity and can rely on each other for help. Think about an auto repair shop where the mechanics have their own repair bays, but colleagues are just a shout away to provide assistance. Still, this is only a small tweak on the solitary problem-solving routine: you're expected to work by yourself and others are consulted only when you're stuck.

Of course, there's another way to troubleshoot: with a team. I've worked on large-scale problems that were too complicated (and too important) to leave to a single person. When the company's future is on the line, expect to be given all the resources you need to resolve an issue. That will inevitably include some extra personnel.

However, you might not be used to problem-solving in a group setting. If you are lucky enough to command an entire troubleshooting team, you'll want to make the most of this precious human resource. Always start with a brainstorming

and organizing session where you discuss strategies and assign roles. Since you'll have everyone's attention, this is the time to really open things up and consider all of your options. During this meeting, you'll also want to address these issues:

Communication

Especially if you're troubleshooting in the midst of a crisis, there will be lots of people interested in the outcome: customers, management, other teams, etc. You don't want these requests for updates to be a constant source of interruptions for the people actually solving the problem. Therefore, be proactive and assign someone to take on the role of the communicator. They will relay the team's status to interested parties and free the rest of your team to actually get some work done. Another tip: have this person set clear expectations as to the *frequency* of communication. You will receive fewer interruptions from interested parties if they know they will be receiving updates on a regular basis.

Alternatives And A "Plan B"

Of course, you should put most of your personnel on the "high-percentage play" (i.e., the strategy you feel is most likely to pay off). But, if your team is large enough, consider breaking into multiple teams, each of which will pursue alternate theories regarding the cause. One such team can even be assigned to "Plan B," preparing for the eventuality that you will not find the cause or fix the problem. This team could be building out a backup system or investigating workarounds.

Setting Thresholds

Once you've decided on a strategy, put limits on how long you'll work without finding a solution (by the way, this is a good idea for the solo troubleshooter as well). The threshold can be the achievement of a specific goal, but always include a **time limit** as well. For example:

"We'll spend the next 2 hours trying to determine if replacing the valve will bring the pressure back to normal levels. If the pressure rises above 100 lbs./sq. in., then we'll stop and shut off the boiler to avoid an explosion."

The thresholds set in this example are:

1. **Timed goal:** work through a specific fix for the next 2 hours, then break and reassess.
2. **Pressure:** stop and mitigate if the pressure rises above 100 lbs./sq. in.

Time limits are a crucial backstop to arrest the momentum of a plan that is going nowhere. Tunnel vision can be difficult to overcome unless there's a prearranged means to stop the madness.

Keep Someone "Up Above It"

One person needs to be managing and monitoring everything listed above and making corrections as needed. Ideally, they're not involved with the actual troubleshooting and therefore won't get swept up in the details. If you've set thresholds, this person is watching the timer and the other parameters you've agreed upon. If you're pursuing multiple alternatives, they're checking in with the various teams on a regular basis, synthesizing this information and deciding to either maintain course or change direction.

Teamwork, An Every Day Thing

If you have the latitude to change how you problem-solve, consider introducing more teamwork. Specifically, try it in pairs. I've done "pair troubleshooting" on numerous occasions and have found working with a competent partner has many benefits:

- **Improved quality of work:** problem-solving with someone else "keeps you honest." Knowing your work is being scrutinized makes you less likely to take the "easy road"—this means fewer shoddy repairs. Fixes will typically conform to the person with higher standards.
- **Faster resolution:** more collective experience means it's likely that someone has "seen this one before." Four eyes will view the problem more clearly and four hands will speed the work along.
- **Less chance of pursuing dead-ends:** if you're on the "road to nowhere," someone will get bored or frustrated and

demand a reassessment of the situation.

- **Better ideas:** two different points of view, and you both can ask "stupid" questions to get the team unstuck.
- **Keeps you externally focused:** the interactive and social aspects of working with a partner is another hook to remaining present while problem-solving.

Attention lone wolves: you'll hunt bigger game in a pack.
(image:Tambako The Jaguar / CC BY-ND 2.0)

Of course, putting two workers on a troubleshooting project has an opportunity cost: they could be pursuing their own repair projects individually. If two employees are able to solve a problem in the same amount of time that a single worker could, all pair troubleshooting would do is double your labor costs! However, my impetus to experiment with pair troubleshooting originated from my very positive experiences with "pair programming" in the world of software development. Researchers Alistair Cockburn and Laurie Williams studied pairing in that context and found:

> The significant benefits of pair programming are that
> • many mistakes get caught as they are being typed in rather than in QA test or in the field (continuous code reviews);
> • the end defect content is statistically lower (continuous code reviews);
> • the designs are better and code length shorter (ongoing brainstorming and pair relaying);
> • the team solves problems faster (pair relaying);
> • the people learn significantly more, about the system and about software development (line-of-sight learning);
> • the project ends up with multiple people understanding each piece of the system;
> • the people learn to work together and talk more often together, giving better information flow and team dynamics;
> • people enjoy their work more.
> The development cost for these benefits is not the 100% that might be expected, but is approximately 15%. This is repaid in shorter and less expensive testing, quality assurance, and field support.
>
> **Alistair Cockburn and Laurie Williams**, **"The Costs and Benefits of Pair Programming"** [1]

To the best of my knowledge, no one has studied "pair troubleshooting," but personal experience tells me the benefits

are similar. Apart from software development, I also paired up systems administrators on my teams and had them troubleshoot together. I would overhear them catching each other's errors—mistakes that would have been costly to fix later on! I was also impressed with how knowledge got spread around by team members collaborating and talking to each other (the study mentions this benefit as "the project ends up with multiple people understanding each piece of the system").

I think the strongest argument for the adoption of pair troubleshooting is the quality of work produced. The Cockburn/Williams study showed that pairs produced software with 15% fewer defects (up to a 50% reduction in other studies) with only an increased cost of 15%. Depending on the circumstance, higher quality work can more than justify the additional cost of pairing. If you're in an industry where defects in repair work are extremely costly (or deadly), then it warrants your consideration. Software bugs can have a huge cost: they suck up the time of customer service reps, field technicians, and must be eventually found and fixed (for more on the topic, see the "Economics" section in the Cockburn/Williams study, it's quite compelling). Likewise, "bugs" in your repair work can also exact a hefty price: you know this all too well if you've ever been called back to fix something you thought was already fixed. Customers want the confidence of knowing it was fixed right the first time!

Pair troubleshooting won't be a good fit in all circumstances: your industry's economics and organization's culture will circumscribe the possibilities for introducing teamwork. Personnel is a factor too: I've worked with some "lone wolf" types that were quite resistant to the concept. Can you convince them to hunt with the pack?

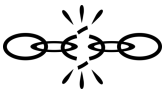
References:

- Header image: "*Workers disembark…*". sol, photographer. Retrieved from Unsplash, https://unsplash.com/photos/tZw3fcjUIpM.
- [1] Alistair Cockburn and Laurie Williams, "The Costs and Benefits of Pair Programming."

Team Spirit was originally published April 28, 2013.

Notes:

Bottlenecks



> Once you solve your top problem, you've got a new top problem.
>
> **Alex Chaffee**

Interconnected systems can suffer from the presence of bottlenecks: individual components that limit the speed of the system as a whole. Limitations like these are an interesting concept because *every* system has a ceiling to its throughput. If you buy a printer that can churn out a maximum of 10 pages per minute, you probably wouldn't think of it as having a bottleneck, even though there's some component inside that ultimately restricts it to that particular speed. However, a throughput of 10 pages per minute would definitely be an annoyance for a printer that was advertised as capable of 30 pages per minute. When a machine is operating normally, its capacity, though finite, is simply part of your assumption for how it should function. However, when a machine's capacity fails to meet expectations—now we're dealing with a bottleneck!

My point isn't that bottlenecks aren't real, but they are qualitative and exist in relative terms: **"It's going too slow!"** is the cry that is often heard. But, slow compared to *what*? When a system's capacity is not enough to meet demand (compared to some *ideal standard*), then the cause of the limited throughput is called a **bottleneck.** Identifying the standard is important, because dealing with bottlenecks often blurs the line between troubleshooting and engineering. When a system is bottlenecked because of a malfunction, then this is clearly the domain of troubleshooting: the system had a certain throughput in the past, and the goal is to get it back there. However, when a system is employed in a new way, or when it's asked to do more than it should, it may bump up against its design limitations. When that's the

source of the slowdown, overcoming those limits by expanding capacity or performance tuning isn't technically troubleshooting.

I point this out because the people who say "Just make it go faster!" might not be aware of the distinction. From their perspective, all they know is that the system isn't performing up to their *expectations*. It's important for you, the troubleshooter, to keep this difference in mind: one path is about restoring functionality and the other is about making process improvements (or resetting expectations). If you're responsible for a machine, you'll likely be asked to investigate in either case, so we'll discuss both. After explaining the situation to whomever is experiencing the problem, if you also include a clever way to improve capacity, you'll look really sharp.

Bottlenecks restrict flow. For this party, that might've been a good thing.
(image: Erich Ferdinand / CC BY 2.0)

The Weakest Link

Chained systems flowing along a single path will inevitably have nodes that complete their tasks at different speeds. There's nothing wrong with this, it's simply the nature of whatever the machine was designed to accomplish. If we're talking about an oil refinery, the various sequential steps (fractional distillation, processing/cracking, treating/blending) are in different states of technological advancement. That one phase takes longer than another is a reflection of the nature of the work being performed and the amount of brainpower (i.e., innovation) that has been brought to bear on the problem. Processes that were once very slow might be comparatively quick now.

The problem with chaining processes together is that the overall throughput of the system will be limited by the rate of the slowest one. If a node in a chain can only handle one item at a time, it must wait for the next node to be clear before it can send its output down the line. If nodes have different completion times, some will finish early and sit idle. The strict definition of a bottleneck is "a narrowing": the type of narrowing relevant to our discussion is **a reduction in the rate of flow,** observed when crossing between two nodes. I want you to see how this works, so let's look at a simple 3-node assembly line system that paints Christmas Tree ornaments. Station A cleans the ornament, B paints it, and C dries the paint. A → B → C:

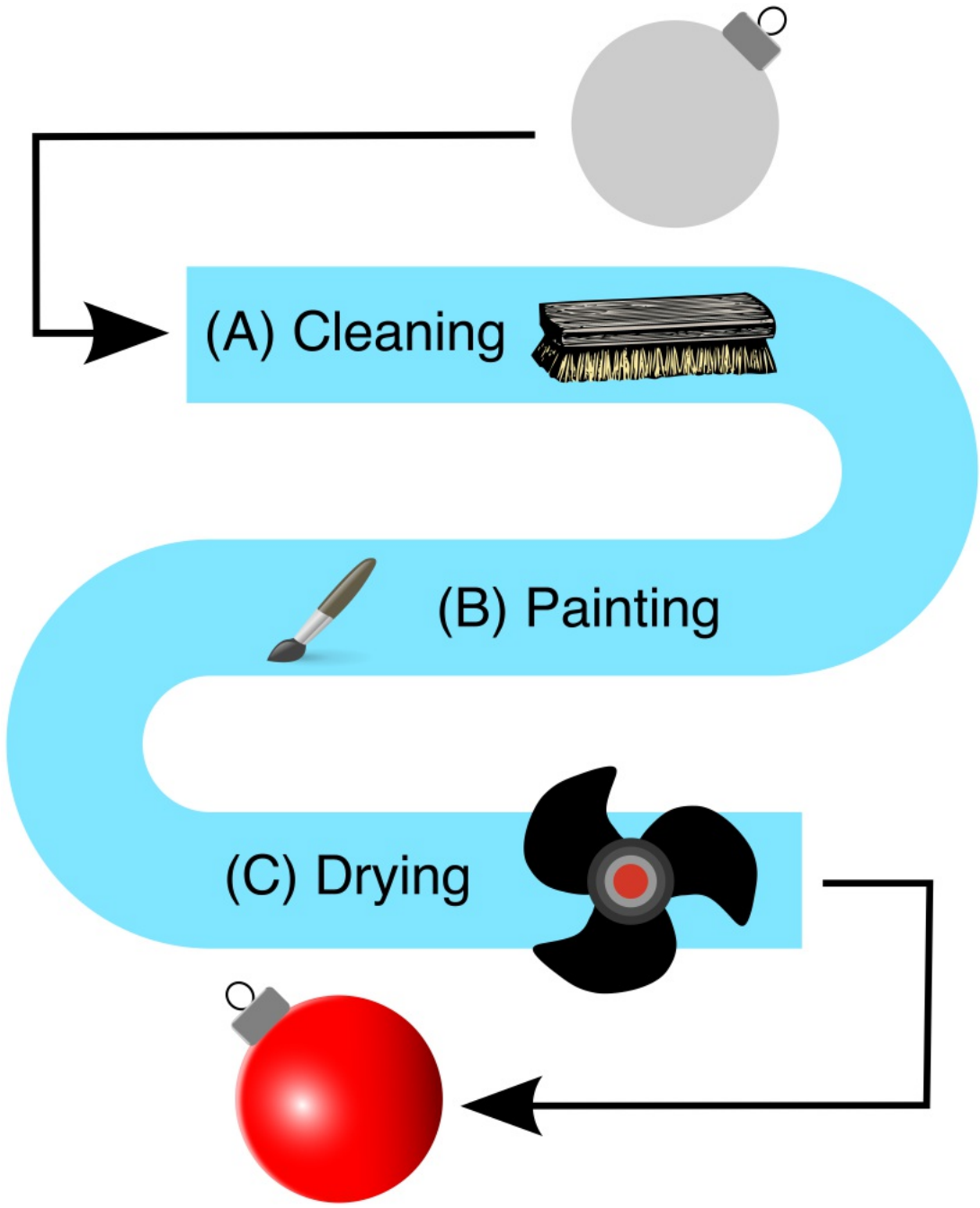


Diagram: an assembly line that cleans, paints, and dries Christmas Tree ornaments.
(image: © Jason Maxham)

The three steps have different completion times:

Station	Work Performed	Completion Time (mins.)	Ornaments Per Hour	% Change In Speed
A	Cleaning	5	12	N/A
B	Painting	10	6	-50%
C	Drying	20	3	-50%

Technically, we've got two bottlenecks here, right in a row! Going from A → B represents a 50% reduction in the rate of speed: 12 ornaments per hour to 6. Likewise for B → C, which goes from 6 to 3. Each of these transitions is a "narrowing" of the flow rate and therefore a bottleneck, but the slowest step (C) will ultimately control the overall rate of this system. Here's a chart showing the progress of 4 ornaments (numbered 1-4) passing through the stations (A-C) of the painting assembly line:

Elapsed Time	Ornament #1	Ornament #2	Ornament #3	Ornament #4
00:00	START	START	START	START
00:05	A	START	START	START
00:10	B	A	START	START
00:15	B	A	START	START
00:20	C	B	A	START
00:25	C	B	A	START
00:30	C	B	A	START
00:35	C	B	A	START
00:40	DONE	C	B	A
00:45		C	B	A
00:50		C	B	A
00:55		C	B	A
01:00		DONE	C	B
01:05			C	B
01:10			C	B
01:15			C	B
01:20			DONE	C
01:25				C
01:30				C
01:35				C
01:40				DONE

Time Summary

	Ornament #1	Ornament #2	Ornament #3	Ornament #4
Waiting Time	00:00	00:15	00:25	00:25
Completion Time	00:35	00:50	01:00	01:00

First off, notice that the first ornament passed through the line in 35 minutes. This is just the sum of the completion times for the individual stations (5 + 10 + 20 = 35). Then things slow down a bit: the second ornament passes through the line in 50 minutes. What gives? You can see from the table that Ornament #2 spent 15 minutes waiting for Ornament #1 to finish (idle moments are shown in red, time spent working in green). Because a station can only handle a single ornament at one time, when it's occupied the ornament coming behind must wait. For example, at the 10 minute mark, Ornament #2 was done with cleaning at Station A; however, it had to wait another 5 minutes for Ornament #1 to vacate Station B so it could move on. Things slow down even further with Ornament #3, which takes 60 minutes to get through the line. At this point, the total completion time stabilizes: Ornament #4 also takes 60 minutes to finish. Ornaments #3 and #4 each spend 25 minutes waiting for subsequent stations to open up.

Once the system is fully loaded, an ornament gets done every 20 minutes: 00:40, 01:00, 01:20, and 01:40. This is what we were talking about when we said that a chained system's rate will be limited by the slowest node (in this case that's the drying station, C).

Not Every Increase Matters

Now that we've got a basis for comparison, let's look at what happens when a problem occurs in the painting line. Let's say that the cleaning and painting stations (A and B) both use compressed air from the same air compressor. Over time, the compressor has degraded and now it takes longer to recharge and maintain the desired pressure level. This result is an increase in the work time for Stations A and B of 5 minutes each:

Station	Work Performed	Completion Time (mins.)	Ornaments Per Hour	% Change In Speed
A	Cleaning	10	6	N/A
B	Painting	15	4	-33%
C	Drying	20	3	-25%

We still have bottlenecks from A → B (-33%) and B → C (-25%), but they're not as large, percentage-wise, as before. What remains the same is that Station C is still the slowest to complete its work at 20 minutes per ornament.

With the line slower, let's see what happens when we send through four more ornaments (numbered 5-8):

Elapsed Time	Ornament #5	Ornament #6	Ornament #7	Ornament #8
00:00	START	START	START	START
00:05	A	START	START	START
00:10	A	START	START	START
00:15	B	A	START	START
00:20	B	A	START	START
00:25	B	A	START	START
00:30	C	B	A	START
00:35	C	B	A	START
00:40	C	B	A	START
00:45	C	B	A	START
00:50	DONE	C	B	A
00:55		C	B	A
01:00		C	B	A
01:05		C	B	A
01:10		DONE	C	B
01:15			C	B
01:20			C	B
01:25			C	B
01:30			DONE	C
01:35				C
01:40				C
01:45				C
01:50				DONE

Time Summary

	Ornament #5	Ornament #6	Ornament #7	Ornament #8
Waiting Time	00:00	00:10	00:15	00:15
Completion Time	00:45	00:55	01:00	01:00

Looking at the numbers, some amazing and very counter-intuitive things have happened. First, idle times while in the queue have actually *decreased*! Ornament #6 waited 5 minutes less (versus Ornament #2 in our previous test run), while Ornaments #7 and #8 waited 10 minutes less each (versus #3 and #4). As far as time in the queue is concerned, Ornaments #7 and #8 took one hour to make it through the line, the same as before the compressor was

having problems. Perhaps most astonishing is that the throughput of the system was **exactly the same**: once the system was loaded, an ornament rolled off the line every 20 minutes at 00:50, 01:10, 01:30, and 01:50!

Once again, we observe that the slowest station controls the pace of a chained system. We've also learned something new in this run: the limiting node can mask problems elsewhere. Because the total throughput remained the same, you might not have noticed the compressor problem until it was too late. Most importantly, if you're troubleshooting a "it's too slow" kind of problem in an interconnected system, you must focus on the *narrowest* bottleneck (i.e., the slowest station) for your efforts to have any effect.

Show Me The…Data!

How do we find that narrowest bottleneck so we know where to direct our efforts? The best case scenario to rapidly pinpoint the location of a bottleneck is operational data. If you had monitors recording the flow rate at each of the stations in the ornament assembly line, you could easily see a decrease in speed either numerically or graphically. However, that's a level of preparation I rarely see, except for those organizations that have been badly burned by bottlenecks in the past (like mine!). Only at the *very end* of my tenure as a CTO could I lay claim to such a level of preparedness (by then I was a data fanatic), but even so we didn't monitor everything. To find a bottleneck in a mass-produced, off-the-shelf item, it might be hard to add the type of probes necessary for this level of awareness.

Slow Your Roll, Even More

Let's say you have a hunch that a particular link in a chain is the narrowest bottleneck. We know that the throughput of a chained system is controlled by the slowest node, so an easy way to vet a suspect is to **slow it down even further!** If the rate of output decreases further, you've found the limiting bottleneck. This is a great technique for digital systems, where flow rates can easily be adjusted with a line in a configuration file or some code added to a computer program.

In our ornament assembly line, Station C was the slowest and therefore controlled the pace of throughput. Therefore, if you slowed Station C down even further, the immediate result would be to decrease the rate that ornaments rolled off the line. Busted, Station C!

One important caveat for this technique is to choose the *smallest* possible increment when adding additional sloth to the suspected node. Remember, in a single-track system *any* node can become the controlling bottleneck. If you tip the scales with too heavy a hand, all you'll learn is that you can make new bottlenecks. I think we knew that already!

Get Out That Stopwatch

Another way to determine the location of a bottleneck is to send a tracking probe down the chain and time its progress. In our painting assembly line, that would mean putting an ornament in the queue and taking note of when it enters and exits each of the stations. Digital systems will have these tools available in software: a programmer can add code to note the time spent in various functions, making database calls, etc. Analog or digital, having precise data of how time is spent in the system is invaluable. That's because there's another possibility we haven't discussed: two or more identical "narrowest" bottlenecks! In our painting assembly line, if all three stations (A, B, and C) took 20 minutes to complete their work, you'd need to speed up *all* of them to achieve a faster system.

Half-splitting To Find The Bottleneck

Ideally, you'd have timing data on every node in a system so that there would be no mystery as to the location of the bottleneck. However, there will be times when collecting such data will come at too high a price: either the probe will be hard to track as it moves through the system, access may be limited, or it just requires a lot of manual labor.

What if you could only collect *one* data point at a time? What would give you the most bang for your buck? Ding, ding, ding! You're right: we can half-split our way to finding the bottleneck and save a lot of effort. We reviewed half-splitting in detail in "Clear Up To Here": remember the teddy bear assembly line? Let's use that example again, but this time in the context of a slowdown. Imagine that a bear normally completes the trip from "naked" to "Christmas fabulous," passing through all 45 assembly line stations, in about 15 minutes (an average of ~20 seconds per station). However, today it's taking 60 minutes for a bear to emerge from the line. I smell a bottleneck.

We'll start by dividing the system in two and then collecting data at the mid-point (again, Station #23). We'll mark a special test bear with a big red "Test" across its face to distinguish it from the rest, and then we'll send it on down the

line. Opening up the access hatch over Station #23, we'll wait for our test bear to pass the mid-point, noting the time when it crosses (in addition to the starting and ending times):

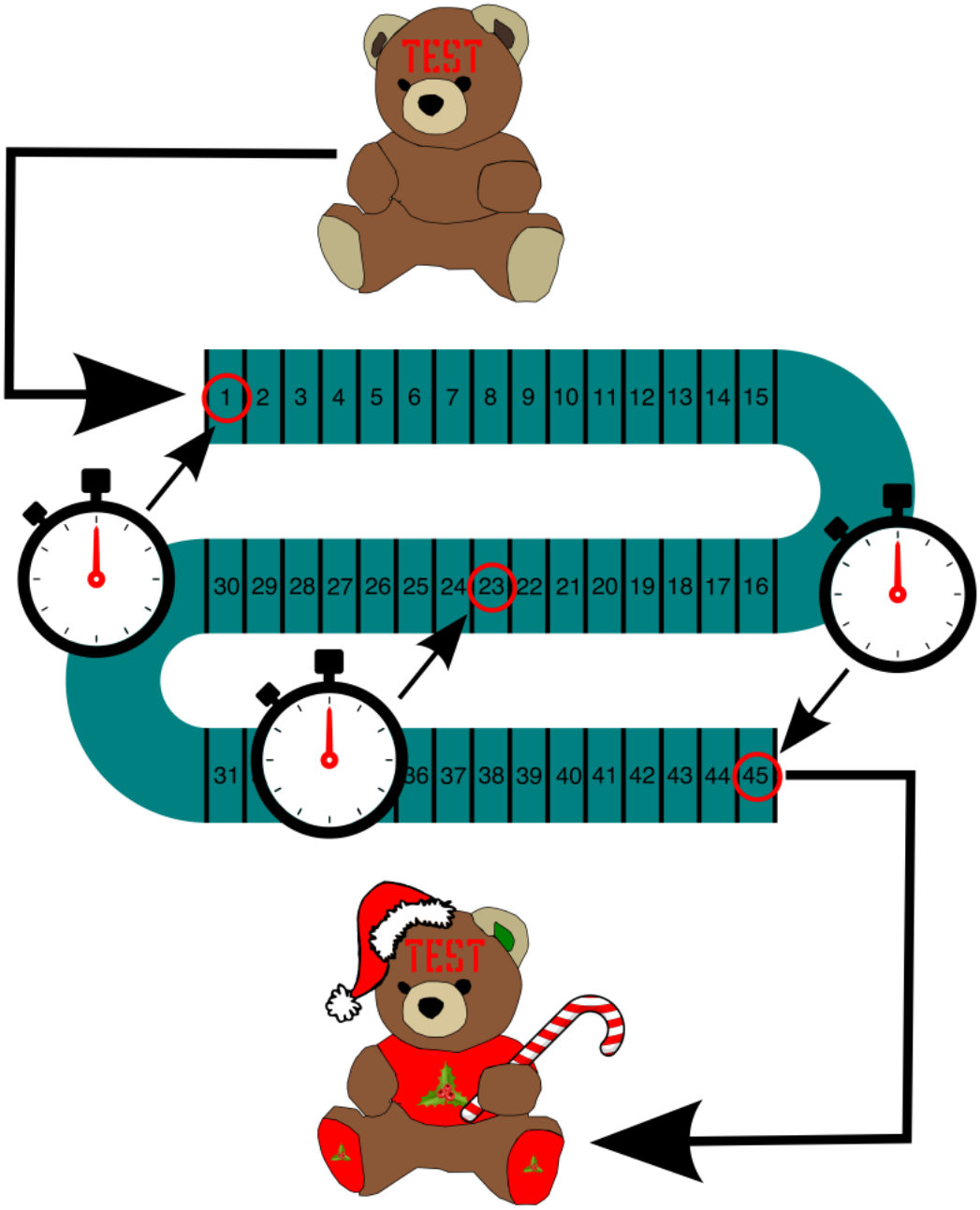


Diagram: using half-splitting with time measurements to pinpoint a bottleneck. The time is noted at the beginning, middle, and end.
(image: © Jason Maxham)

These are the timings we get:

Station Location	Time Elapsed
Start	00:00
#23	00:48
End	01:00

You can see that it took 48 minutes for the test bear to reach Station #23. That means the time it takes to go through the

first half of the line (48 minutes) is **4 times** longer than the second half (12 minutes)! This is a very strong indication that the bottleneck is located between Stations #1-23. Just like before, we can continue half-splitting until we've narrowed it down further. If that was necessary, the next step would be to send another test bear down the line and take a reading at Station #12. But, maybe the first split is good enough to jump-start an investigation. Remember, half-splitting is a very efficient way of *eliminating possibilities*. Let's say, based on other evidence, you had a solid list of three suspects: Stations #12, #30, and #40. The timings from our initial split indicate the bottleneck is in the first half of the assembly line, thereby eliminating #30 and #40. At that point, I wouldn't split further, and instead head right to Station #12.

Make It Go Faster

I've spilled a lot of ink talking about how bottlenecks work and how to find them. Of course, there's that last step: making the system as a whole go faster. If you find that the narrowest of narrows is the result of a malfunction, then it's just a matter of making the correct repair. However, another possibility is that the system is now being asked to do something that's beyond its capability or capacity. Or, the system can be performing exactly as intended: some machines are *designed* to slow down under certain types of workloads. Rectifying these scenarios can be much harder. Basically, your options for the bottlenecked node are:

1. Make it go faster.
2. Make it do less.
3. Stockpile.
4. Parallelize.

The first one is self-explanatory: zoom zoom! As for number 2, "doing less" can mean any number of optimizations, but the basic idea is that scrutiny will often reveal *unnecessary work* being done by the bottlenecked node. Stockpiling means running the bottlenecked system more (perhaps even 24/7/365) and storing its output to smooth out swings in demand. Unfortunately, this entails inventory and storage costs, and doesn't help systems that must do "on-demand" work. Parallelizing is where you deploy multiple systems to simultaneously handle the workload of the bottlenecked node. Like widening a one-lane road, this means work can be performed in parallel along multiple pathways. In our painting assembly line, drying was our bottleneck. To double the speed of that task, we could add a second drying station:

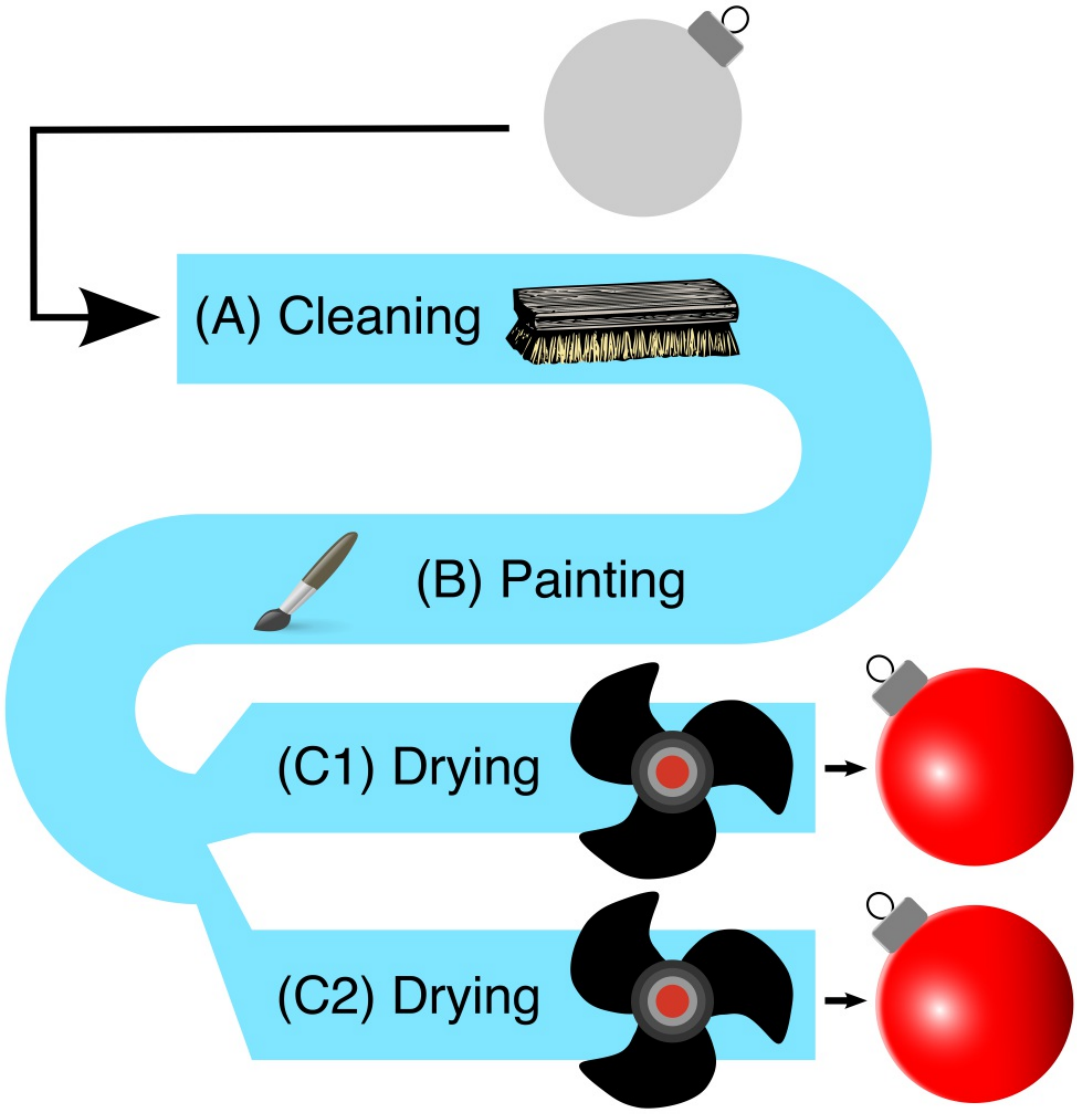


Diagram: parallelizing the Christmas Tree ornament painting line with dual dryers.
(image: © Jason Maxham)

This is the kind of thing that's very easy to mock up in PowerPoint, but much harder to actually pull off in a working factory. Removing bottlenecks with parallelization to meet growing demand can be a golden ticket or your worst nightmare. When it comes to scaling, some companies make it, and some don't. Finally, remember that optimizing bottlenecks is like a game of Whac-A-Mole: speeding up one part of a system will make some other part the slowest. Once you've dispatched a particular bottleneck, another one will appear! With Whac-A-Mole, you stop when the game is over. With bottlenecks, you stop when your company is wildly profitable.

Things Need To Be Just Right

> Then Goldilocks went upstairs into the bedchamber in which the three Bears slept. And first she lay down upon the bed of the Great, Huge Bear; but that was too high at the head for her. And next she lay down upon the bed of the Middle Bear; and that was too high at the foot for her. And then she lay down upon the bed of the Little, Small, Wee Bear; and that was neither too high at the head, nor at the foot, but just right. So she covered herself up comfortably, and lay there till she fell fast asleep.
>
> **The Annotated Classic Fairy Tales** [1]

After you've fixed your fair share of bottlenecks, you might achieve that rare honor of making something go *too fast*. Your record-setting innovation may flood downstream nodes with an excessive amount of work. That's happened to me: I've released bottlenecks that were actually *protecting* the system with their sluggish ways. When that happens,

you'll need to install governors to return the flow rate back to a level that can be sustained. Tuning involves fixing things that are too slow, as well as too fast. The best of luck as you work to get things *just right*.

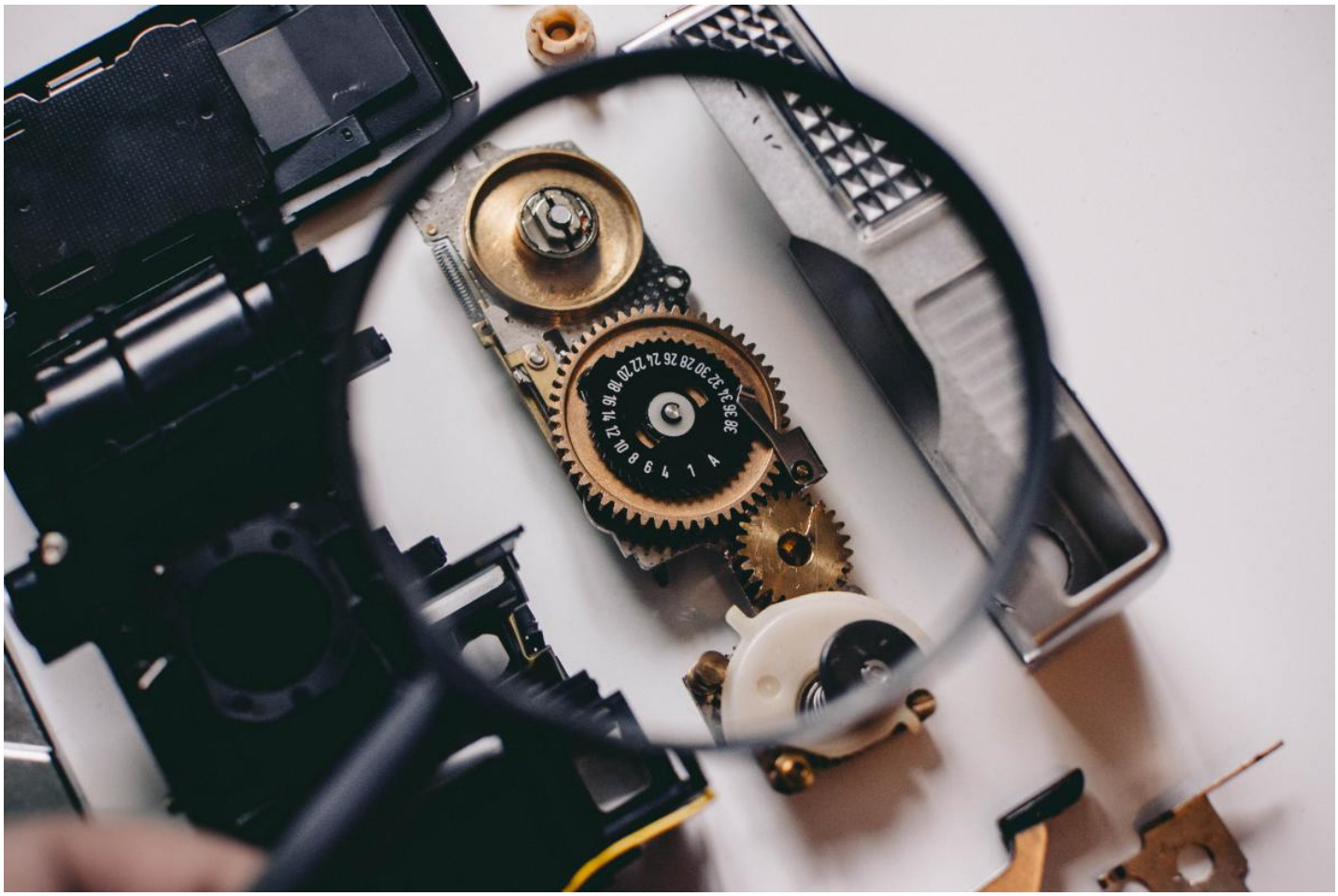
References:

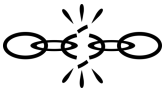
- Header image: *Road construction delays traffic on West Side Highway, at 79th Street, New York City, during rush hour*. World Telegram photo by Al Ravenna. New York, 1951. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/94505641/.
- [1] Maria Tatar, *The Annotated Classic Fairy Tales,* (New York: W. W. Norton & Company, 2002), pg. 249.

Bottlenecks was originally published May 7, 2013.

Notes:

How Is It Supposed To Work?





> I read a surprising amount of the manual... There's always a detail I overlooked, a switch I missed, or something I didn't get quite right.
>
> **Karl Kuehn**

A breakdown is a deviation from a machine's operational state; the process of fixing something involves moving from that broken state back to the ideal one. What is the ideal state? Well, you definitely know it by the desired *outcome*: to have the machine perform like it did before the failure, to carry on doing useful work. However, that's a far cry from being able to describe what should be happening inside a system to realize that goal. Understanding a machine on this level makes comparing "broken" and "working" possible—the difference between the two will point the way to the appropriate remedy. Knowing how a system works is like having a map and compass for your repair: "broken" is where you are and "working" is where you want to go.

Heading in the right direction?
(image: airguy1988 / CC BY-ND 2.0)

Where Am I?

Before we get into a discussion of moving towards a machine's ideal state, we should dwell on the importance of knowing its **current state.** You may say it's "broken," but that's not very specific. Sticking with our map analogy, point-to-point navigation requires knowing both your current location and the desired destination. A heading is relative to where you are right now. If you want to get to San Francisco and you are in Los Angeles, you go north; if you're in Seattle and you want to get to Frisco, then you need to head south.

However, imagine being blindfolded and dropped off in the middle of a forest. From this unknown starting point, it's not enough to know that your destination is San Francisco. Which way is it? You wouldn't know, so your first order of business would be to answer the question "Where am I?" Don't worry, unlike some fraternities, being blindfolded and finding your way home from a strange place isn't required to join the Troubleshooters Guild. Unless you want to…it sounds like a great way to build some character, and everyone needs at least one good cocktail party story.

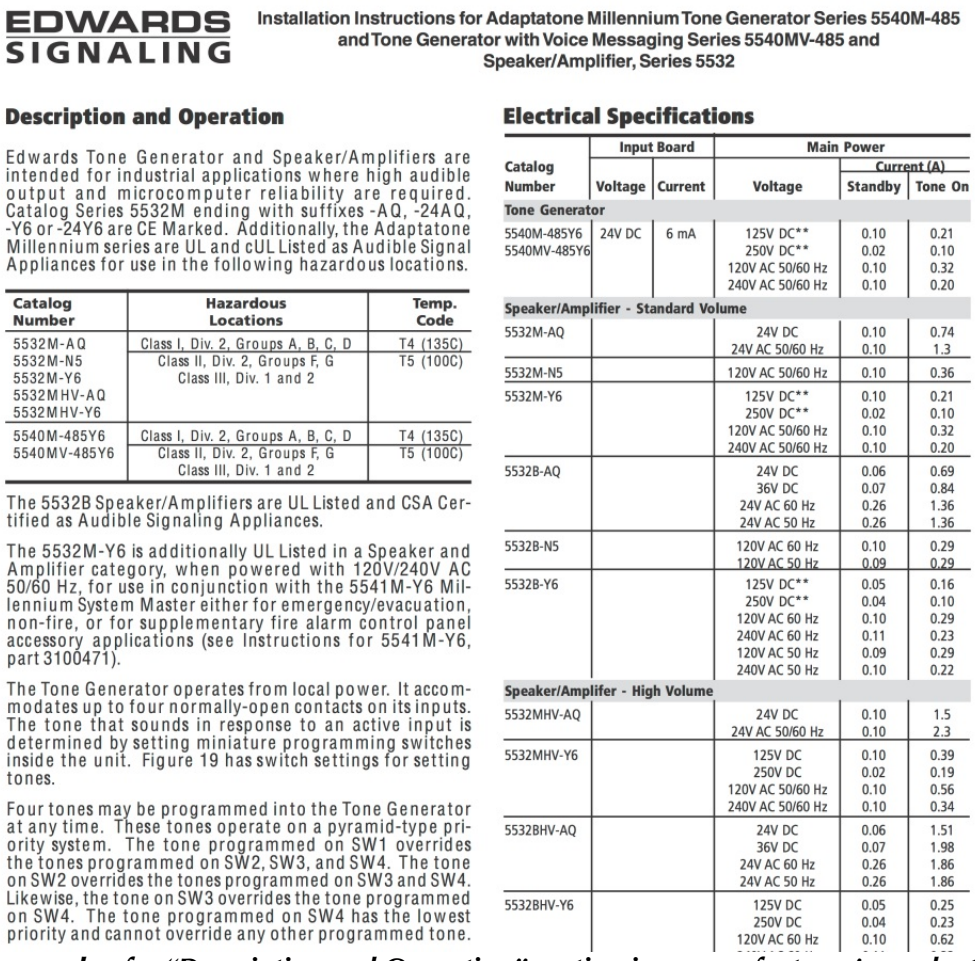
Discovering the current state of a system is a common thread that winds through many of the strategies in *The Art Of Troubleshooting*. Therefore, I'll just briefly review some of the best tactics:

- **Inspect:** use your eyes, ears, and nose.
- **Indicator Lights/Error Messages:** when the machine is trying to tell you what's wrong (its current status), please pay attention.
- **Built-in Diagnostics:** can you ask the machine how it's day is going?
- **Gauges/Probes:** some machines come with gauges to tell you the state of various internal parameters. If you've embarked on your own data collection project, then maybe you're added your own. Either way, check these out.
- **Logs/Records:** for digital devices, there is typically a place where the system will record its goings-on. For mechanical machines, the logs may be analog, but they're just as useful. It would be a shame if someone had already noted the problem you're experiencing and you wasted your time duplicating their efforts.

In addition to these ideas, there's a very useful method from the medical profession called a **"review of systems."** In a review of systems, a doctor goes from head to toe, asking you about each part of your body. Is your eyesight good? Do you have any trouble hearing? Any digestion problems? The goal is to solicit information about any recent changes or troubling conditions systematically. Going over the body in this way ensures that no important detail is missed.

In the same way, you can add a review of systems to your troubleshooting routine. If you're thorough, it really is the ultimate way to know the "current state." Your review can cover a specific machine or an entire factory. Whenever we

had a customer complaint at Discovery Mining, I would divide my team up and do a brief review of systems. There were several major components powering our web site, each of which could be the source of trouble: the network, the database, our Internet connection, the web servers, etc. Whenever resources allowed, I always liked to check in with each of these parts and make sure they were operating within acceptable limits. When my obsession with data collection finally began to bear fruit, these review of systems became very easy for my team. Each subsystem had its own set of easily-accessible, continuously updated graphs that made a full inspection possible within just a few minutes.



EDWARDS SIGNALING

Installation Instructions for Adaptatone Millennium Tone Generator Series 5540M-485 and Tone Generator with Voice Messaging Series 5540MV-485 and Speaker/Amplifier, Series 5532

Description and Operation

Edwards Tone Generator and Speaker/Amplifiers are intended for industrial applications where high audible output and microcomputer reliability are required. Catalog Series 5532M ending with suffixes -AQ, -24AQ, -Y6 or -24Y6 are CE Marked. Additionally, the Adaptatone Millennium series are UL and cUL Listed as Audible Signal Appliances for use in the following hazardous locations.

Catalog Number	Hazardous Locations	Temp. Code
5532M-AQ	Class I, Div. 2, Groups A, B, C, D	T4 (135C)
5532M-N5 5532M-Y6 5532MHV-AQ 5532MHV-Y6	Class II, Div. 2, Groups F, G Class III, Div. 1 and 2	T5 (100C)
5540M-485Y6	Class I, Div. 2, Groups A, B, C, D	T4 (135C)
5540MV-485Y6	Class II, Div. 2, Groups F, G Class III, Div. 1 and 2	T5 (100C)

The 5532B Speaker/Amplifiers are UL Listed and CSA Certified as Audible Signaling Appliances.

The 5532M-Y6 is additionally UL Listed in a Speaker and Amplifier category, when powered with 120V/240V AC 50/60 Hz, for use in conjunction with the 5541M-Y6 Millennium System Master either for emergency/evacuation, non-fire, or for supplementary fire alarm control panel accessory applications (see Instructions for 5541M-Y6, part 3100471).

The Tone Generator operates from local power. It accommodates up to four normally-open contacts on its inputs. The tone that sounds in response to an active input is determined by setting miniature programming switches inside the unit. Figure 19 has switch settings for setting tones.

Four tones may be programmed into the Tone Generator at any time. These tones operate on a pyramid-type priority system. The tone programmed on SW1 overrides the tones programmed on SW2, SW3, and SW4. The tone on SW2 overrides the tones programmed on SW3 and SW4. Likewise, the tone on SW3 overrides the tone programmed on SW4. The tone programmed on SW4 has the lowest priority and cannot override any other programmed tone.

Electrical Specifications

Catalog Number	Input Board Voltage	Input Board Current	Main Power Voltage	Current (A) Standby	Current (A) Tone On
Tone Generator					
5540M-485Y6 5540MV-485Y6	24V DC	6 mA	125V DC** 250V DC** 120V AC 50/60 Hz 240V AC 50/60 Hz	0.10 0.02 0.10 0.10	0.21 0.10 0.32 0.20
Speaker/Amplifier - Standard Volume					
5532M-AQ			24V DC 24V AC 50/60 Hz	0.10 0.10	0.74 1.3
5532M-N5			120V AC 50/60 Hz	0.10	0.36
5532M-Y6			125V DC** 250V DC** 120V AC 50/60 Hz 240V AC 50/60 Hz	0.10 0.02 0.10 0.10	0.21 0.10 0.32 0.20
5532B-AQ			24V DC 36V DC 24V AC 60 Hz 24V AC 50 Hz	0.06 0.07 0.26 0.26	0.69 0.84 1.36 1.36
5532B-N5			120V AC 60 Hz 120V AC 50 Hz	0.10 0.09	0.29 0.29
5532B-Y6			125V DC** 250V DC** 120V AC 60 Hz 240V AC 60 Hz 120V AC 50 Hz 240V AC 50 Hz	0.05 0.04 0.10 0.11 0.09 0.10	0.16 0.10 0.29 0.23 0.29 0.22
Speaker/Amplifer - High Volume					
5532MHV-AQ			24V DC 24V AC 50/60 Hz	0.10 0.10	1.5 2.3
5532MHV-Y6			125V DC 250V DC 120V AC 50/60 Hz 240V AC 50/60 Hz	0.10 0.02 0.10 0.10	0.39 0.19 0.56 0.34
5532BHV-AQ			24V DC 36V DC 24V AC 60 Hz 24V AC 50 Hz	0.06 0.07 0.26 0.26	1.51 1.98 1.86 1.86
5532BHV-Y6			125V DC 250V DC 120V AC 60 Hz	0.05 0.04 0.10	0.25 0.23 0.62

Excerpt: example of a "Description and Operation" section in a manufacturer's product manual.
(source: Edwards Signaling, Installation Instructions for Adaptatone Millennium Tone Generator Series 5540M-485")

Description And Operation

If you happen to be the designer of a system, then you should know how it's supposed to work. However, in our world of mass-produced goods, it's rare that a machine was crafted by your own two hands. But, you're in luck because the manufacturer will often provide you with a metaphorical map and compass. Among these resources, you'll find manuals, schematics, blueprints, service bulletins, troubleshooting trees, and how-tos. These materials should be consulted first: the complexity of some of today's machines is astounding and so the knowledge of the people who were responsible for their design is indispensable. Years of careful thought may have gone into a product's evolution, leading to counter-intuitive engineering tradeoffs that are difficult to understand without the same context as the original designer. Don't be surprised by these things, when instead you could be informed of them in advance.

Some industries are better than others when it comes to providing the details of "how it should work." I find the automotive industry to be particularly rich in documentation: I've looked at auto shop manuals and the level of description provided for even the smallest components can be impressive. Cars are expensive and important, so the professional troubleshooting industry that serves automobile owners is generally well equipped. In other contexts, manufacturer resources can be scant. Back when I was building computers, I purchased many parts that came with only a sentence or two regarding normal operation, usually on a blurry photocopy that was barely legible. In these

cases, you may need to contact the manufacturer's service department to get the necessary details to help you troubleshoot.

The last option, the hardest, is figuring out how something is supposed to work with just the machine as your source of information. Results will vary greatly, depending on how complicated the machine is and your level of expertise. Given enough resources (primarily time), every machine can be reverse engineered. Those engaged in historical restorations (e.g., classic cars) or those who are tasked with maintaining very old systems will eventually have to resort to the intelligent guesswork of trial and error. In "Duplicate The Problem," I noted how the fix-it knowledge of mass-produced systems decays over time: documentation is lost, manufacturers go out of business, and human know-how withers as whole industries are upended in the creative destruction of capitalism. When that happens, you'll need to fill in the gaps by yourself.

Finally, we should take a moment and recognize the invaluable role that education plays in understanding how "it's supposed to work." In my interviews with great troubleshooters, many of them cited reading manuals and taking classes as the foundation of their skills. This "general education" is an essential building block to understanding the systems under your care, even if most of what you learn isn't applicable immediately. Can you really say you're prepared to help without self-study or formal education? Rich Kral, a veteran HVAC repairman, would say otherwise:

> "I think when you first get into a trade, you need to read the manual completely. After awhile, you can get to the point where you scan it… But I do believe that if you want to take care of a piece of machinery for a customer, you should know how it works. Don't wait until you get a service call and someone wants their air conditioning fixed and you're sitting up there [on the roof] reading the book."
>
> **Rich Kral**

Expectations Versus Normal Operation

How something should work is not just about schematics and manuals, it can also be a matter of *expectations*. Whenever you interact with a new machine, you draw upon all your previous experiences, leveraging your assumptions for how it ought to function. These assumptions are very useful: after you learn how to drive a particular car, you can use that experience to help you drive *any* car. However, not all machines are created equal: how one works may not carry over to others in its class.

Sometimes, a repair is just figuring out the difference between someone's expectations and how a machine was designed to work. A great example of this came up in one of the interviews I conducted for *The Art Of Troubleshooting*. Seasoned auto mechanic Dan McCormick related this encounter with a customer:

> You have to know the system. We had a person come in and say, "I was driving down the road and the car starts chiming!" I had to think about that for a while…the car's chiming?! Did you have the direction light on? He says, "No." Are you sure you didn't have the direction light on? "Well, I don't think so." That was one of the features of this particular car: if you left the direction light on for more than 3/10 of a mile, the computer would pick it up and start chiming to let you know you left it on. The customer went for a drive, intentionally leaving the direction light on, and it chimed the same way. He came back and admitted that's what was happening! You see, a problem to him was actually just normal operation.
>
> **Dan McCormick**

This story resonates with me: many times I've been called in to "fix" things that weren't broken. The "repair" was simply to align someone's expectations with the ways of the machine. Attentive listening, combined with system knowledge, can save a lot of time and prevent you from searching for problems that don't exist. Once again, the mind is mightier than the wrench.

References:

- Header image: *"Camera Parts"*. Shane Aldendorff, photographer. Retrieved from Unsplash,

https://unsplash.com/photos/mQHEgroKw2k.

How Is It Supposed To Work? was originally published May 17, 2013.

Notes:

Repair Or Replace?

> All economic activity is based upon an uncertain future. It is therefore bound up with risk.
>
> **Ludwig von Mises**

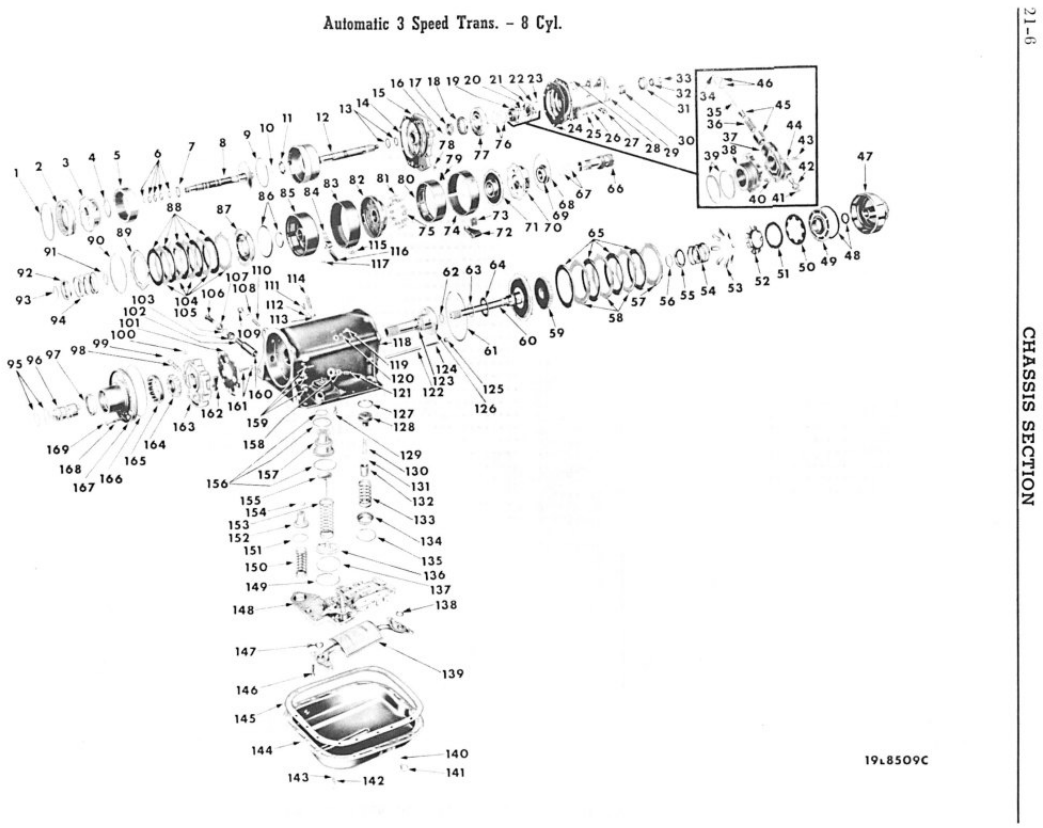
As the economist Ludwig von Mises once said, human action is "purposeful behavior." [1] Machines amplify our actions and are employed *purposefully*, to satisfy a specific want. When a machine breaks down, the need it was fulfilling persists—this is the driving force behind every repair. Therefore, every troubleshooting project has the following 3 elements: a need, a broken system, and a finite supply of repair resources. That basic setup leads to the following options:

1. Repair the system
2. Replace the system
3. Upgrade or downgrade the system

"I wanted a new one anyway."
(image: Smithsonian Institution)

In all endeavors, we want to fulfill our desires in the most efficient way possible: the resources we save can be put to other uses. The **need** a broken machine was serving is the most important consideration: if it can be met at a lower cost some other way, fixing becomes unnecessary. If the need is still present, but has grown or diminished, then upgrading or downgrading should be on the table.

As a troubleshooter, you will always be thrown into the middle of the "repair vs. replace" dilemma. The cost of your efforts to find the problem and execute the proper fix can tip the balance in favor of either "repair" or "replace." Therefore, it's vital that you understand the economics involved and your role in the process. Also, the ability to competently counsel your customers on this critical question builds trust and leads to the kind of long-lasting business relationships that you want to cultivate.



Let's have an assembly contest. It's you versus the manufacturer. Go!
(image: Chrysler Parts Catalog, 1960)

The Troubleshooter As Manufacturer

> The greatest improvement in the productive powers of labour, and the greater part of the skill, dexterity, and judgment with which it is anywhere directed, or applied, seem to have been the effects of the division of labour.
>
> **Adam Smith, Wealth of Nations** [2]

I want you to conduct a little thought experiment: imagine all the pieces of an automobile, strewn across the floor of a garage. Your task is to assemble these parts into a working car. To make things interesting, we'll have a contest: if you can do it faster than the manufacturer did it, you get to keep the car. Go!

Not interested? This thought experiment is useful because it highlights one of major differences between "repairing" and "replacing." If we go back to my Universal Troubleshooting Recipe, you'll recall that there are two steps for making any repair:

1. Find the problem.
2. Fix it.

The second step, fixing something, is a recreation of the manufacturing process. Getting a machine back to working, **you tread the same steps that were done at the factory.** The troubleshooter's role as a *de facto* manufacturer becomes clearer when you think about how many times a machine can be *re*-built over its lifetime. An analogy might be useful: you may have heard the myth that "every cell in your body is replaced every 7 years." There's a kernel of truth to this: certain types of cells are being continually replaced. Dr. Jonas Frisen, a stem cell researcher at the Karolinska Institute in Stockholm, believes that the *average* age of your cells is between 7 and 10 years (some cells, like the neurons in the cerebral cortex in your brain, are never replaced). The human body is similar to many machines in this respect: some parts bear the brunt of wear and tear and must be frequently replaced (like the tires on a car). Other parts can last the entire lifetime of the machine: the engine on a modern automobile, when properly maintained,

can easily be driven beyond 1 million miles. Over the long run, *all* parts must be replaced: you could maintain (i.e., continually rebuild) a machine indefinitely if you had an infinite supply of all the necessary components and the know-how to install them.

The problem is, the manufacturer will always be better at putting together machines. That's the whole point of their existence: to efficiently turn raw materials into finished products! To that end, they have all kinds of advantages over you in your role as a troubleshooter. Let's start with the inputs: a manufacturer strives to find the most efficient way to source the raw materials it uses, forming relationships with suppliers to acquire materials in bulk. You can attempt the same, but buying parts by the thousands or millions and getting the associated savings won't realistically be an option. Whereas manufacturers purchase components in large quantities, to fix something you usually need only one of a part (or just a few). Therefore, on average, the troubleshooter's cost of materials is going to be higher.

When it comes to how labor is used, the troubleshooter is a jack-of-all-trades: wherever the problem lies within the machine, you must go there and be prepared to fix it. Contrast that with the highly specialized labor used in manufacturing, it's rare for a machine to be built by a single person anymore. In general, the troubleshooter must know much more about a machine than the individual workers in an assembly line. However, that knowledge will be shallow compared to the deep expertise of a worker who performs the same operation, day in and day out, on a specific part of a machine. That's one of the advantages of assembly-line production: each worker becomes highly proficient at their individual task versus the "artisanal" approach where one person builds something from start to finish.

Think about just a simple operation, like placing a tire on a car. Done by the side of a road, this involves a jack, a tire iron, and some grunting (or maybe that was swearing I heard…?). Done in a high-tech factory, the car is already off the ground, perhaps the wheel is automatically lifted into the perfect position with a crane, while the nuts are tightened down with a pneumatic wrench in a fraction of a second. Even for the most proficient troubleshooter, the amount of time spent on even this simple task is going to be orders of magnitude longer versus the time it took at the factory.

A manufacturing environment is designed to be entirely predictable. Creativity may be an asset for the person who dreams up a "better mousetrap" and for those who discover, for the first time, how to efficiently produce it. Churning out carbon copies, however, favors conformity: manufacturers strive to create processes that are so straightforward and routine that unskilled labor can be used for the majority of the tasks. Contrast that with the skill required to pull off a complicated repair: malfunctions are usually anything but "routine" and the amount of knowledge required to point yourself towards the correct fix can be formidable. Therefore, a troubleshooter's know-how requires a combination of product design *and* assembly skills. That blending of talents is harder to locate in a single person.

To add to the manufacturer's advantage, fixing something often requires *disassembly*. That means first *reversing* what was done at the factory, and then replicating the manufacturing process from that point. That's a double whammy in terms of efficiency. Piling on top of that, we return to Step #1 in the Universal Recipe: you need to initially find out where the problem lies. Resources like *The Art Of Troubleshooting* aim to make problem discovery more efficient, but the fact remains that this is a step the manufacturer doesn't have to take. Of course, manufacturers also have to discover and overcome problems with their products, but they usually do this up-front in the prototyping and testing phases. Once a product's design is set, manufacturing is about efficiently replicating that model, over and over.

So what? Well, the difference in efficiency between the troubleshooter and the manufacturer looms large in the "repair versus replace" question. The only hope for troubleshooting to be economically advantageous is to use a *small fraction* of the labor and parts that went into the original machine. Because the cost of replacement is always the standard to beat, the basic equation for an efficient troubleshooting exercise is:

Cost of Discovery + Cost to Fix < Cost of Equivalent Replacement

All things being equal, the cost of finding the problem and making the repair must be kept below the cost of replacement. We'll delve into all the nuances below, but this basic accounting is the starting point for our discussion. In addition to this calculation, there are many other relevant factors, some of which are difficult to quantify with numbers. Finally, note that this is a *forward-looking* equation: it doesn't matter how much you've previously spent on a machine (aka, "sunk costs" which cannot be recovered). I'm often surprised at how brutal depreciation can be to a machine's value! You can't be emotionally attached to what you spent on it when it was new: the only relevant consideration is the current replacement cost.

Once this thing is launched into space, repair becomes a touchy subject.
(image: SDASM Archives)

Repair

In the "repair vs. replace" matchup, you'll notice that repairing has **two** separate costs:

1. Problem discovery
2. Executing the fix (includes parts and labor)

The first component, figuring out what is wrong, can be highly variable. For low-value items, it's entirely possible to exceed the replacement value of the item solely in the problem discovery phase. This is a good time to talk about risk, because repair typically involves many more unknowns than replacing. In addition to the risk that you waste a lot of resources just figuring out what's wrong, executing a fix has its own hazards. Whenever you take something apart, there's always a chance you won't be able to put it back together again. Certain repair operations may require a very long series of steps to be implemented perfectly. It's also easy to waste money on parts: I've botched repair jobs and been left with a pile of replacement parts that couldn't be used (many times, even if you can return the parts, you must

pay a restocking fee). I've also installed replacement parts incorrectly and destroyed them in the process. Oops, there goes good money down the drain!

All of the above risks can be mitigated. Skill and experience can reduce the time needed to discover problems, aid in the execution of difficult repairs, and prevent the waste of materials like replacement parts. However, the above risks will still be present and can rear their ugly heads at any moment. Repair involves many elements of uncertainty, which are similar to the risks faced by entrepreneurs in the world of business. Entrepreneurs take risks with capital and their time, in hopes of turning a profit. Troubleshooters make similar bets with their repairs, chasing wins in the form of savings versus the cost of replacement.

Apart from the added risk, troubleshooting does have one big advantage versus replacement: the knowledge gained from investigating an issue and making a fix can be re-used later on. Once you've figured out the root cause of a problem, it will be much easier to recognize and handle if it happens again. You may pay a large up-front cost the first time you tackle a particular issue, but that investment can pay dividends later on if the problem recurs. Likewise with the actual fix, you'll discover shortcuts and tricks that will speed future repairs. Speaking of discovery, you'll also learn invaluable things about your machines, processes, and organization when you troubleshoot: these nuggets can be built upon with any number of "continuous improvement" paradigms. Lastly, it might not have economic value, but there's also the soul-stirring satisfaction of knowing you can solve your own problems. Any time you can cultivate an independent spirit of your own, grab the opportunity!

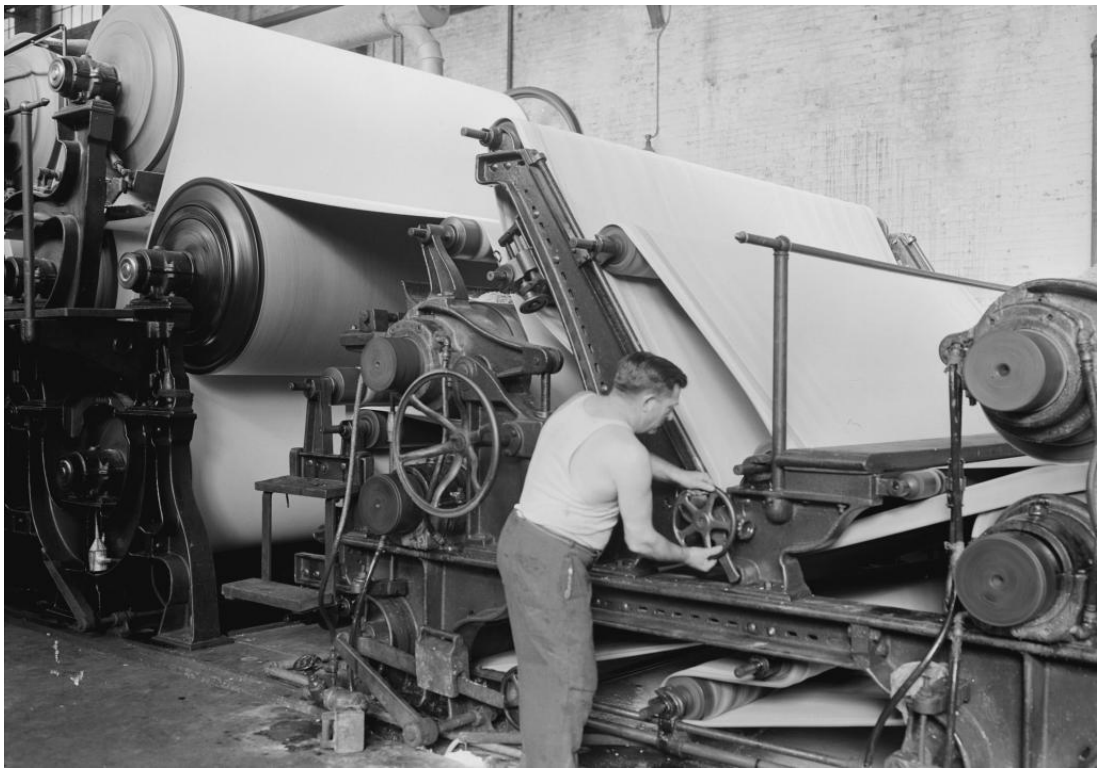
Shotgunning: Replacing Instead Of Discovering

As we've seen, the time and resources required to pinpoint the cause of a malfunction can easily exceed the cost of replacing a broken machine. To make repair more efficient, you can short circuit the discovery process by replacing multiple components in lieu of definitively diagnosing the problem. This technique is called "shotgunning" and evokes the blast of a shotgun shell, which sprays pellets onto a relatively wide target area. Following the analogy, most troubleshooting is like sharpshooting: you try to put a single bullet right on the center of the target. However, the precision strategy is not appealing when the cost of discovery is relatively high and the cost of replacement parts is relatively low. Shotgunning aims for a different balance, substituting parts for labor, with the aim of achieving savings versus a definitive diagnosis that takes a lot of resources.

An example: you're repairing a TV and have isolated the problem down to three possible microchips. The cost of the chips is trivial, perhaps just a few dollars each. However, figuring out which *one* is causing the problem would take hours of testing with the aid of specialized equipment and software. Why not replace them serially, paying respect to the core principle of changing just one thing at a time, so that you know which one is the culprit? That's a good instinct, but there are circumstances which can make serial replacement uneconomic, like when there is a very high overhead to testing a fix. Let's say that, on this particular TV, verifying the efficacy of *any* repair requires running a lengthy 4-hour diagnostic check. If we got unlucky and the failed component happened to be the third one, that would mean spending 12 hours running tests! Instead, you opt to replace all three chips simultaneously. Doing so, you avoid both the cost of discovering which one is the cause and the potential lengthy overhead of the multiple tests needed for serial replacement.

Caveats:

- Shotgunning requires the use **known good components.** Make sure you are using replacement parts that have been verified to work. Otherwise, you run the risk of turning a situation with one failed component into a scenario with *multiple* failed components.
- If you are conducting a repair for a third party, replacing components that aren't technically broken may bring up ethical or legal issues. The aim is to save money, so most people will be on board with the overall goal of shotgunning. As with most things, problems can be headed off with transparency: make it the customer's choice by presenting them with the option to shotgun and let them make the final decision. After all, it's their money!



Because of its immense size, a paper machine is a good example of something that's difficult to replace.
(image: Lewis Hine / The U.S. National Archives)

Replace

The argument for replacement revolves around certainty. As noted, repair can involve a good deal of risk: the time needed to discover the problem can be highly variable, and then the repair must also be executed correctly. Contrast that with replacement: by purchasing a known working system, you can completely bypass these concerns. To guarantee operation within a certain time frame, swapping may be your only option, given the uncertainties associated with repair.

However, replacement isn't all sunshine, unicorns, and rainbows. There are risks and downsides here too. For example, a replacement machine may require a lengthy break-in period to become fully functional. Extensive configuration or tuning might be needed to integrate a new machine into your operations. Additionally, that old machine might have contained "embedded knowledge" in the form of long-forgotten customizations; these subtle differences often won't become apparent until you start testing the new machine. The other tricky thing about finding a suitable replacement is compatibility: the manufacturer may claim that a new machine will "run just like the old one," but I've been bit enough times to be skeptical of such claims! Machines can be like wine vintages, just because it's newer doesn't mean it's better. Mass-produced machines are designed to appeal to the masses. While this is great for the manufacturer, new models can be a step forwards or backwards when it comes to your specific purpose.

The last thing to consider when replacing a machine is the cost of installation: if a machine was difficult to get into place, it will likely be difficult to remove and install something else in its stead. On the extreme end of this is something like a paper machine, which can take up an entire warehouse-sized building. You're not going to pick up something like that and swap it out in an afternoon!

The Need, Reassessed (Upgrading And Downgrading)

When a machine isn't bothering you, it's easy to forget about the need that led to its acquisition. When it goes up in smoke, your knee-jerk reaction may be to reach for your wallet. However, before you lay down money to replace or repair it, go back to that original need and see if it's still present: you may find it has grown or shrunk.

Reassessing the purpose served, you may find that you've overbuilt: the broken machine was simply too large, fast, or feature-rich for how it was actually being used. This happens to me all the time: I'm a sucker for more horsepower, watts per channel, lumens, and pixels. I often find myself with more machine than I need! In those cases, **downgrading** to a smaller capacity machine will still meet the need and save you money in the process. Less

capable models may also be less complex and therefore increase reliability: bells and whistles are great but they also expand the number of things that can go wrong with a machine.

The other possibility is that the machine in question was underserving your needs: this is the case for **upgrading.** Maybe it was a bottleneck in your workflow and limiting the throughput of your business. Perhaps a new feature can save you a lot of labor, or the latest models use less energy and will therefore cost less to operate. This is all great, but the numbers need to be right: make sure you calculate the "return on investment (ROI)" to ensure the additional benefits will pay for themselves. When upgrading, what we're really talking about is "spending money to make money." This is exactly what entrepreneurs do, so consider yourself an honorary member of that club.

Technological change and obsolescence should also be a consideration in your decision to upgrade. The older a machine gets, the further away it gets from that sweet spot of know-how and available fix-it resources that is generated when a large segment of the population uses a particular machine. Sticking with the "herd" has advantages: maintaining a fleet of Ford Model T's was (relatively) easy in the 1910's and 1920's. Replacement parts, tools, mechanics who knew how to work on them, and the availability of service information was at its peak. In addition to the fact that modern cars have so many advantages compared to the venerable Tin Lizzie, maintaining a fleet of Model T's today would be frustrating because the herd has moved on to greener pastures.

Beware, upgrading can mean an increase in the use of consumable inputs. Whatever your machine is used to eating, after upgrading it will be even hungrier! Increased energy consumption is common after an upgrade. Again, will the cost of these additional expenses be worth it? Make sure to plan ahead and verify that your infrastructure can handle the increased load—upgrading your equipment when you're already maxed out is a recipe for trouble. Likewise, downgrading usually means using less. If the previous machine had lots of excess capacity, the savings from a downgrade can add to your bottom line (e.g., a lower energy bill).

You may discover that you didn't build it big enough the first time…
(image: Caroline Gagné / CC BY 2.0)

Repair AND Replace

Possibilities for repair lie along a spectrum: from a quick fix that might only last a few minutes, all the way to a beautiful titanium-encased refurbishing that extends the life of a machine far beyond the original design. Faced with an emergency situation, you may even have to contemplate fixes that *lessen* the life of a machine or part. For example, should you have a flat tire, you can use a fast-acting spray sealant that fills the leak from the inside. However, a tradeoff for some of these products is that their use will eventually destroy your tire. Therefore, repair or replace isn't either-or. A combination of the two might be the best answer, with repair pursued now and replacement in the

medium or long-term.

Used vs New

"Used or new?" This is a question for replacement, but also for repairs as parts can be obtained second-hand. The usual tradeoff is that new things will incur fewer maintenance-related expenses. The reason for this is because the normal wear and tear that degrades components (and leads to malfunctions) hasn't happened yet. With a new machine or part, statistically you've got more time between deployment and that first malfunction. The other reason why new things have fewer expenses is because of warranties, which is a pledge by a manufacturer to provide protection from repair costs for a limited time after purchase.

However, that shiny newness and warranty comes at a higher price, so the question is always: **"Is the additional expense worth it?"**

A Lot Of Ins and Outs, What-Have-You's And Strands To Keep In Your Head

All of the above considerations regarding whether to repair or replace may be difficult to keep straight in your mind (I know it was for me when writing this section), so I've created this table summarizing the main points:

Issue	Repair	Replace with equivalent	Upgrade the system	Downgrade the system
Problem discovery (i.e., figuring out what's wrong)	– MINUS	+ PLUS	+ PLUS	+ PLUS
Fix must be executed properly	– MINUS	+ PLUS	+ PLUS	+ PLUS
Possible waste of materials (spare parts, etc.)	– MINUS	+ PLUS	+ PLUS	+ PLUS
Reuse of knowledge for future fixes	+ PLUS	– MINUS	– MINUS	– MINUS
Opportunities to learn from failures (RCA, etc.)	+ PLUS	– MINUS	– MINUS	– MINUS
Use of consumables (energy, etc.)	± NEUTRAL	± NEUTRAL	– MINUS	+ PLUS
Compatibility	+ PLUS	– MINUS	– MINUS	– MINUS
Break-in period	+ PLUS	– MINUS	– MINUS	– MINUS
Configuration/tuning	+ PLUS	– MINUS	– MINUS	– MINUS
Installation costs	+ PLUS	– MINUS	– MINUS	– MINUS
Time to resolution	– MINUS	+ PLUS	+ PLUS	+ PLUS
Throughput/features	± NEUTRAL	± NEUTRAL	+ PLUS	– MINUS
System complexity	± NEUTRAL	± NEUTRAL	– MINUS	+ PLUS
Warranty protection	– MINUS	+ PLUS	+ PLUS	+ PLUS
Future repair ecosystem: availability of parts, know-how, etc. (i.e., sticking with the "herd")	– MINUS	+ PLUS	+ PLUS	+ PLUS
Protection from obsolescence	– MINUS	– MINUS	+ PLUS	– MINUS

Please note that I chose what I believe is the most common outcome or experience for each of these scenarios. To give you just one example of how difficult it is to summarize these parameters, take a look at "Use of consumables" for the "Upgrade" category. I mark it as a "minus" because, in my experience, when you upgrade to a bigger/faster/stronger system, consumption of inputs usually increases (especially energy). However, technology can mitigate this tendency: if a lot of innovation has occurred, newer models can be better in *all respects*. That is, they can be cheaper, faster, longer-lasting, and more energy efficient all at once!

The above table shows how complicated the "repair vs replace" decision can be, especially if you also add "used vs

new" to the mix. Depending on your circumstances, any one of the factors discussed can be in the driver's seat and overshadow the rest. Here are some examples that solidify the point:

- **Reuse of knowledge:** a clothing factory has recently purchased and installed 1,000 identical sewing machines. One breaks down. The know-how gained from diagnosing and completing the repair can be applied to the other 999 sewing machines in the factory. Even if this particular repair turns out to be uneconomical, what you learn will be invaluable for future "repair vs replace" decisions. The value of the experience will literally be multiplied a thousand-fold. **Verdict: Repair.**
- **Consumables:** a home's old air-conditioner has broken down. Between now and when it was installed 20 years ago, the cost of electricity has skyrocketed and newer models are much more efficient. Even though repair is a cheaper option, the homeowner calculates the energy savings from a new unit would pay for the difference in less than 6 months. **Verdict: Replace.**
- **Time to resolution:** the switch at the heart of a bank's computer network malfunctions at the end of the business day. Basic troubleshooting yields no resolution. Just a single day's lost business from a network outage would be 1,000 times the cost of a new switch, which can be procured from a local electronics retailer and installed the same night. **Verdict: Replace.**
- **Installation costs/break-in period:** the main ingredient mixer at a fruitcake bakery malfunctions in late October. This particular model had a long, 3-month break-in period where it could only be run at half-speed. Even though fixing it might be expensive, replacing it would severely bottleneck production right before the busy Christmas season (because of the slowness associated with the break-in period of a new unit). The mixer is also very heavy and bolted to the floor; moving it would take an entire day. **Verdict: Repair.**
- **Protection from obsolescence:** a company that specializes in safety reflectors for bicyclists suffers a break down to the machine that applies the reflective coating to their various products. The market this company serves has been moving towards a new and brighter reflective coating technology. Unfortunately, the broken machine doesn't have the capability to apply the new reflector coating material. Because it was scheduled to be replaced anyway, it makes sense to accelerate the purchase of the new machine. **Verdict: Replace.**

In conclusion, the repair or replace dilemma is framed by the need that persists after a machine breaks down. This unmet necessity is primary and your resources are limited, so be sure to compare the two paths before making a decision. For fixing to be competitive, the cause must be quickly identified and the pitfalls associated with repair deftly avoided. On the other hand, finding a suitable replacement has its own costs and perils. The desire to save resources drives the search for the optimal solution; this goal-directed action, done in the face of uncertainty, is why the troubleshooter and the entrepreneur are kindred spirits.

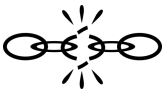
References:

- Header image: *"Phone repair"*. Kilian Seiler, photographer. Retrieved from Unsplash, https://unsplash.com/photos/PZLgTUAhxMM.
- [1] Ludwig von Mises, *Human Action,* Chapter 1. Acting Man (pg. 11).
- [2] Adam Smith, *Wealth of Nations*, I. Of the Division of Labour.
- Nicholas Wade, "Your Body Is Younger Than You Think," *The New York Times,* August 2, 2005.

Repair Or Replace? was originally published June 5, 2013.

Notes:

The 50 Percent Rule: Repair Or Replace, Revisited



> Human action is the use of means to arrive at preferred ends. Such action contrasts to the observed behavior of stones and planets, for it implies purpose on the part of the actor. Action implies choice among alternatives.
>
> **Murray Rothbard**

Introduction

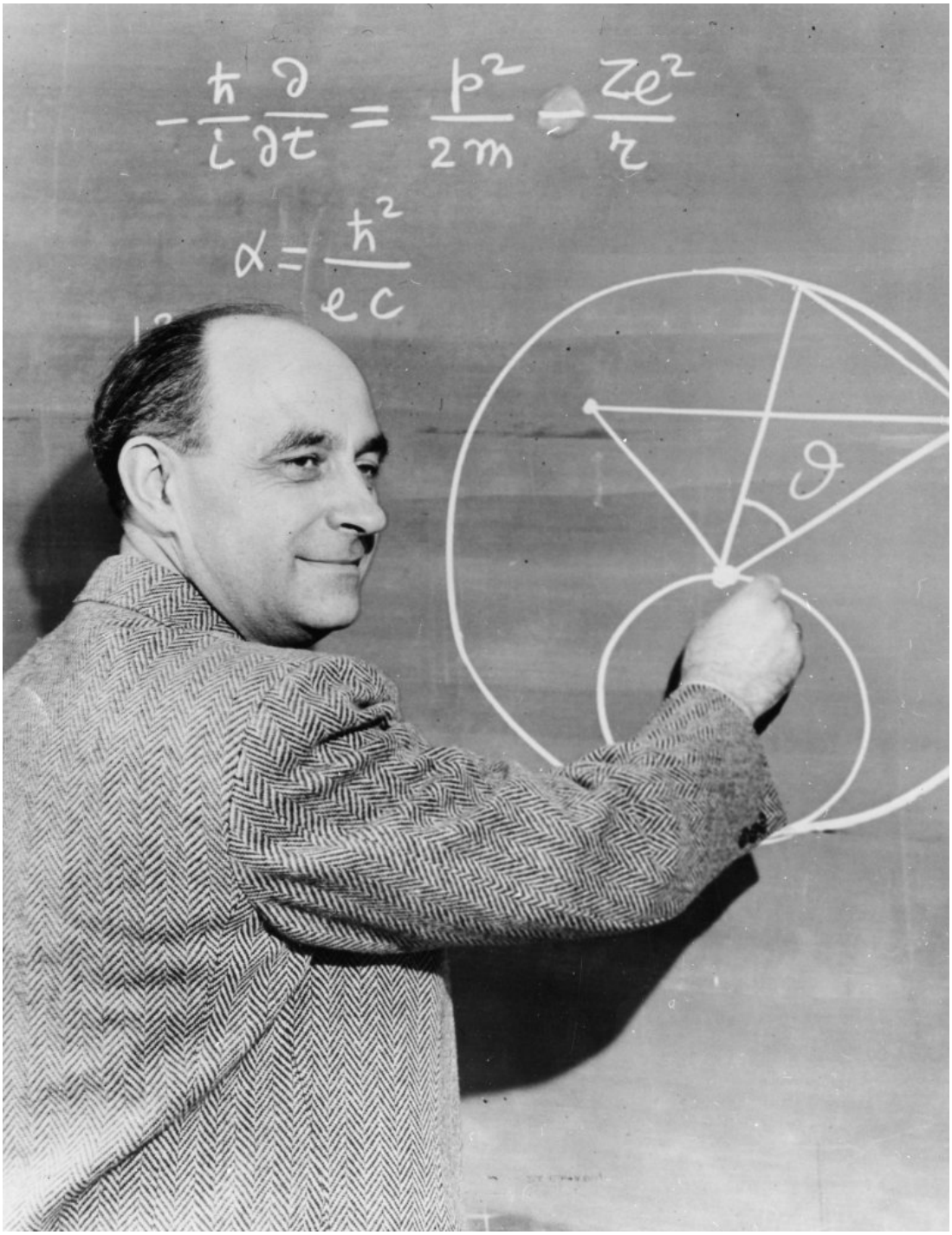
When I first sat down to write about the "repair or replace" dilemma, in the back of my mind there was a vague notion that a simple, calculation-based method would be the ultimate solution. I had heard of the 50% Rule and thought that it (or something like it) would be the obvious centerpiece of the decision-making process.

However, my thinking on the topic quickly evolved. As I reviewed my own history of grappling with "repair or replace," especially in the context of my role as a CTO, I found it wrapped up in all the glorious complexity of the problems faced by the entrepreneur. The need a machine fulfills is the driving force behind the decision-making process. The myriad ways in which we use machines to satisfy our desires, along with the constraints and benefits presented by a particular situation, are difficult to express in an equation.

Both repair and replacement have a long list of pros and cons associated with them. Depending on the circumstances, the controlling factor that ultimately drives your decision can vary widely. Can a formula take into account all of this complexity, and spit out the correct answer? My first pass left the 50% Rule wanting as a tool: it had no obvious way of incorporating the aspects which I felt were primary considerations, so I completely cut any mention of it from my

original piece, "Repair Or Replace?".

Still, I was determined to revisit the issue and test the 50% Rule as a decision-making guide. That's because I found so many references to it in various advice columns pitched to everyday consumers. Even the vaunted Consumer Reports magazine cites it as a rule of thumb to resolve the repair or replace conundrum. Perhaps the 50% Rule couldn't describe the complicated decision matrix used to choose a machine for a particular purpose, but maybe it did have a narrower context in which it should be called upon? I wanted to find out.



Can the repair or replace decision be reduced to an equation?
(image: Smithsonian Institution)

It's Still Worth Something, Right?

Before we get to the 50% Rule, I want to discuss some basic ways to think about the economic value of repair. Let's start at the beginning, with the broken system itself: it's easy to forget that, although a machine is broken, it usually retains *some* worth. If you choose to forgo repair, you can often receive something in exchange for a non-

working system. That something is called the **salvage value.** In accounting, the term has a specific meaning with respect to depreciation: "the estimated value that an asset will realize upon its sale at the end of its useful life." Of course, whether or not to extend the "useful life" of a machine is exactly the decision we're trying to make.

Salvage values will vary widely depending upon the type of machine and the market for its components. Many times a car, even in a state of total disrepair, can be sold for parts to a local junkyard. Consumer electronics often fare much worse. For instance, considering a new one can be had for almost nothing, a broken DVD player will likely have no residual worth to others. There's one more twist: sometimes a machine's salvage value can be **negative** if there is a disposal cost involved. While I have sold cars to the junkyard in the past, I haven't always been successful in this endeavor. I remember when my first mini-van (don't laugh, it was a pretty sweet ride), which I drove out West when I first moved to San Francisco, was involved in an accident and completely wrecked. I called several local junkyards and was surprised that nobody wanted it—even when I offered it to them for free! I contacted some local tow-truck drivers, who had personal relationships with many more firms than I tried initially, and even they couldn't give it away. That mini-van ending up having a negative salvage value of several hundred dollars, which included a disposal fee and the cost of a tow.

Scarcity and demand drives the salvage value: you can easily see this in the price paid for vintage cars and their parts. For example, there were only a couple thousand Plymouth Superbirds built. Working or not, these cars are highly sought after by collectors and restorers, commanding a much higher price than similar vehicles from the same era. Finally, on the extreme end of salvage values, I present this fine specimen to you: in 2011, Tata Motors unveiled an insane version of their Nano, slathered in about $4 million worth of precious stones and gold (80 kg, 22 karat!). The grandiose bling-o-rama of this publicity stunt has a curious effect on the salvage value of such a car:

- Working gold-and-jewel-encrusted Tata Nano: **$4 million**
- Not working gold-and-jewel-encrusted Tata Nano: **$4 million**

Functional or not, this car is worth about the same. This is one of those rare examples where a machine's value is independent of its working state.

Value-Added Repair

Whereas a breakdown takes away from a machine's value, repair should add it back. However, recapturing that added value when it comes time to sell is constrained by the other options available to buyers in the market. You may be objectively improving a machine's capabilities by having it fixed, but that investment won't necessarily be recognized by buyers and returned to you later on. Let's walk through a classic example of an uneconomic repair scenario. You purchase a used car at the righteous price of $500. It may not look like much, but hey, it's transportation:

"She may not look like much, but she's got it where it counts, kid."
(image: hobvias sudoneighm / CC BY 2.0)

After driving it for a few months, it breaks down. After consulting with a mechanic, it is determined that the transmission is bad and needs to be replaced. The total bill (parts and labor) for a new transmission is $2,000. You agree to the repair, thinking "If I spend $2,000 fixing the car, it should be worth at least that amount afterwards, right?" A year later, you want to upgrade your wheels and put the car up for sale, but are unable to find a buyer in the $2,000

range. In fact, the best offer you get is $500, what you originally paid for the car in the first place. What happened to your significant repair investment?

Well, when you bought the car, you had many options of what to do with your $500. Likewise, your potential buyers do as well. Car shoppers won't care what you've spent on the car in the past: they're only concerned with what their money can buy them *now*. If the prevailing market price for a similar used car is $500, then that is where you'll need to price yours if you want it to sell. Sadly, all your $2,000 investment did was to restore the car to the point where it could be considered an equal amongst the other options in the $500 range. Sure, it has a brand new transmission, making it an undeniably better car in that respect, but look at the options from the perspective of a buyer. They could take your asking price of $2,000 and buy four (4 x $500 = $2,000) equally bedraggled chariots, driving each one in turn until it dies.

The lesson here is that the market will limit your ability to recapture the cost of repairs when it's time to sell. We can describe this situation mathematically and attempt to discover in advance if we're about to create a loss. I'll show you the formula for the implied value of a repair, to see how the market values it. First, let's define a couple of variables:

- $m_{salvage}$ = market value of the broken machine (aka, the salvage value)
- $m_{post\text{-}repair}$ = market value of the machine **after** repair
- $r_{value\text{-}added}$ = value added to the machine by the repair

which are related like this:

$$m_{post\text{-}repair} - m_{salvage} = r_{value\text{-}added}$$

For our transmission example, let's say the salvage value of our car was $100. Sadly, we found out that the resale value of the car, even after making our costly repair, was only $500. Therefore the implied value of the transmission repair was:

$500 – $100 = **$400**

This fix raised the value of the car from its salvage value of $100 to $500, so we can say that the **market value of the repair** was the difference: four hundred dollars ($400). I know what you're thinking: unfortunately, we paid $2,000 for this work!

Let's prevent this from happening again. We'll define one more variable:

r_{cost} = the direct, out-of-pocket cost of the repair

and note the relationship for an economically sound repair decision:

$$r_{cost} \leq r_{value\text{-}added}$$

This inequality says that the improvement in the market value of a machine should be less than or equal to the cost of the repair. In our case, that was **not true.** We paid $2,000 which resulted in an increased value of only $400. Instead, we might have had more fun losing it at a casino, or created an art film of ourselves burning the money. At least that would have made for a good cocktail party story.

Therefore, we can calculate a profit or loss from a repair as follows:

$$r_{value\text{-}added} - r_{cost} = r_{profit/loss}$$

In the case of our $2,000 transmission repair, that would be a loss of:

$400 – $2,000 = **-$1,600**

Ouch. However, if you stay on the sunny side of this equation, there's also the chance to make gains from your repairs. Let's say you have an amazing pair of mechanics at your disposal, like the deeply expert Tappet Brothers (perhaps the most famous troubleshooters of our time). They take your hunk of junk, make a few clever jokes (at each other's

expense), and fix the transmission for $50. This allows you to sell it for $500 to some other sucker. Now, the profit/loss calculation ($r_{(value\text{-}added)} - r_{(cost)} = r_{(profit/loss)}$) is a very happy one:

$400 – $50 = $350

A $50 repair that improves the market value by $400 is a slam dunk.

Where Did The Dough Go?

The risk of not recapturing your investment is present in so many different situations in life and business. Home renovations instantly come to mind: you can do the same analysis of a dwelling's price before and after a remodel. It's very easy to spend money on renovations that do not increase the value of a home by an equal amount, just like in the transmission repair example above.

But…where did the money go? How can it just disappear? This brings us to a very interesting aspect of making improvements to your possessions. Whether it's repairing a car, house, refrigerator, or computer, if you intend to resell it, you have become a *de facto* entrepreneur. No longer is the decision about the subjective value you receive from a successful repair, but rather about anticipating the future needs of buyers and the market conditions when it comes time to sell.

Fixing a broken machine with the intent of eventually reselling it is conceptually no different from what any manufacturer does. A pencil-maker assembles things like brass, cedar, factice, glue, graphite, lacquer, pumice, and wax. Combining and transforming these raw materials with labor, he tries to correctly anticipate the public's demand for writing instruments, hoping to sell the pencils for a gain. (Please read "I, Pencil" for an eloquent and eye-opening narrative of the coordination and complexity required to produce this consumer staple you've probably taken for granted.)

In the case of the transmission repair, even though our product was a one-off, we temporarily became a manufacturer. We took some intermediate goods (a broken down car and replacements parts), hired some labor (the efforts of a mechanic), and created a new product: a lousy car with a brand-new transmission. However, is this something that potential buyers want and are willing to pay for at the price we will offer later on? I think it's safe to say that the people who desire run-down cars want them precisely because they *don't* have brand-new transmissions. That is, they're trying to save money!

We could drive that car another 500,000 miles until the transmission died again, making the repair solely for our own consumption. In that case, any market-based loss would only be theoretical. Whatever the outcome, devoting resources to the fix could be justified exclusively on our own hierarchy of values. Again, the market price is different from how much a repair is *personally* worth to you.

Of course, we'd still need to contend with opportunity costs, that nagging suspicion that our money could have been better spent elsewhere. The $2,000 we dropped on the transmission repair could easily have been allocated differently. Our beater could have been sold for its salvage value of $100. That amount plus $400 from our savings could have been spent on another fabulous jalopy. Our transportation problem solved, the remaining $1,600 could then have been used on…a fabulous vacation to Hawaii! No matter how we forked out that $2,000, we always run the risk of deciding, in retrospect, that it could have been better spent.

50%…Of What?

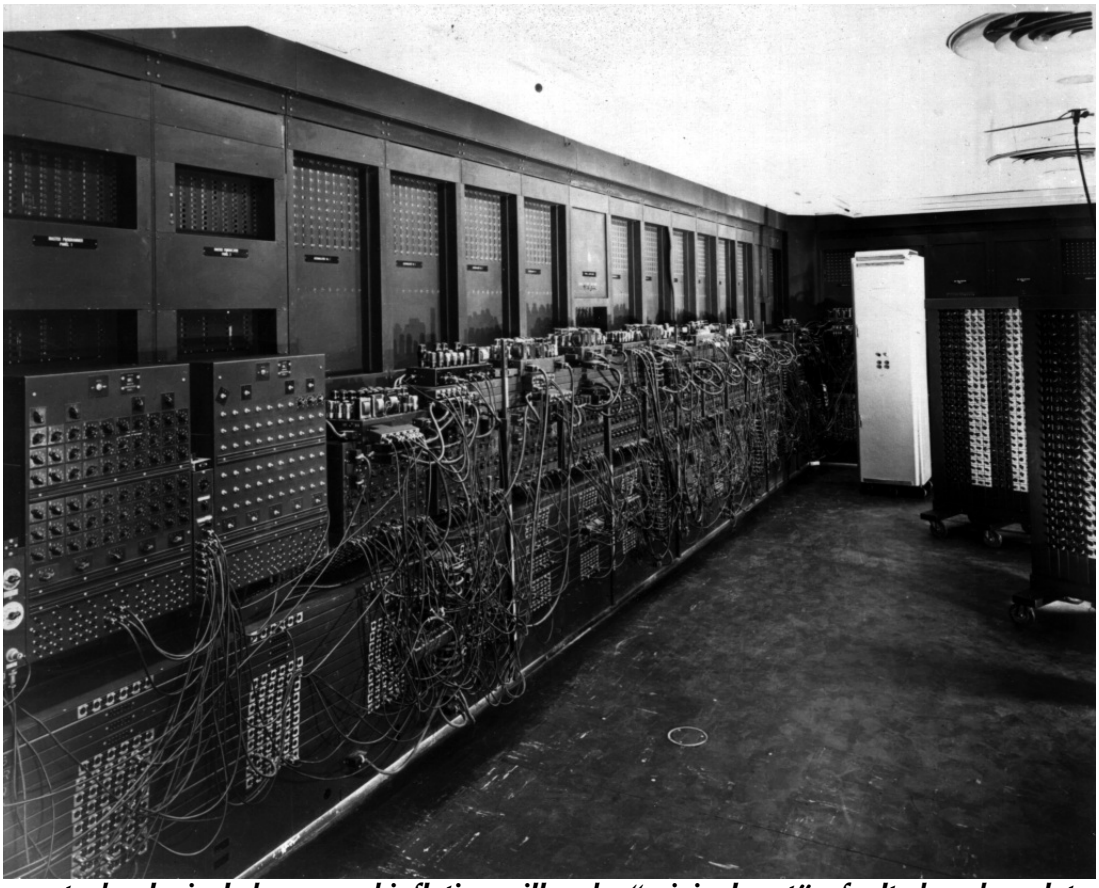
Now that we've laid out the basics for how to think about the value of broken machines and repairs, let's move on to the 50% Rule. First off, we should define the maxim. The basic idea is that you compare the cost of repair to the cost of replacement. If repair exceeds 50% of a particular threshold, the rule says you should opt to replace. But, if a repair can be completed for less than 50% of the baseline, then you should choose to fix. So, the inequality favoring repair looks like this:

Repair Cost < Replacement Threshold × 50%

Unfortunately, I couldn't find an official definition of the 50% Rule, at least with respect to the repair or replacement of machines (it seems more well-defined in the world of real estate, insurance, and disaster recovery). What's especially problematic is that the threshold value that repair is weighed against is variously cited as:

- The original purchase price of the broken machine.
- The current replacement value of an equivalent machine (i.e., a similarly used one).
- The cost of a new machine.

For a given situation, each of these benchmarks could provide a wildly different answer, tipping the scales in either the direction of repair or replace. Figuring out what the standard should be is going to be our first challenge, and we haven't even gotten to the math and statistics behind the rule.



As time passes, technological change and inflation will make "original cost" a faulty benchmark to use for the repair or replace decision. Adjusted for inflation, the ground-breaking ENIAC computer cost $6 million. Even at its original price of $500,000, the equivalent computing power could be purchased today for almost nothing.

(image: US Army Photo)

The (Original) Price Is…Wrong

Using the **original purchase price** as a standard for the 50% Rule would seem to be the most problematic of all our choices. If a machine is old, inflation will have eaten away at the value of the price you paid for it, making it appear smaller. Using this as a benchmark will sadly tell you more about the misguided policies of central banks and the debasement of your money over time (leading to a wholly different "repair or replace" discussion). While I encourage that introspection, it doesn't advance our goal of figuring out what to do with a broken machine. Also, it seems like a lot to ask the average consumer to get out a table of inflation statistics and calculate constant dollars, tied to some benchmark year, like economists do. Although, the folks at the US Bureau of Labor Statistics have made it awfully easy to do just that, with this handy web-based inflation calculator.

Counterbalancing the effect of inflation on the original cost is technological change, which will make the original price paid seem much *larger*. For example, the first all-purpose digital computer was named ENIAC and built at the University of Pennsylvania in 1946. This "giant brain" was built at a cost of $500,000 (about $6 million today). Using that sum to weigh a repair decision, with the idea that you would pay to have another ENIAC built (even just a few years later), is folly. When it comes to industries where rapid innovation is taking place, current designs are often better in all respects *and* cheaper too. This is definitely the case for most consumer electronics, especially those subject to Moore's Law.

Get Me Another One Like This (That Works!)

The **replacement cost** of an identical machine (i.e., a used one in similar condition) would seem like a better choice as a basis for comparison, given that it keeps the variables in play at a minimum. This standard attempts to make an apples-to-apples comparison by asking "How much would it cost to acquire a machine *exactly* like this?" Of course, you want a machine just like the broken one, with one important difference: the replacement should work! Since the replacement cost is the price as of *now*, this standard won't suffer from time-based distortions like inflation or technological change.

However, we should be clear about the practical problems with this as a standard to be employed in real-world situations:

- Finding an available identical replacement is often impossible. You may be contemplating a repair or replace decision for a 2012 Porsche Carrera with 10,000 miles, but what if the closest you can find in your area is a 2011 model with 25,000 miles? To use replacement cost as a benchmark implies that a replacement can actually be purchased! You may be able to look up the theoretical value in a price guide or see recent auction closing prices, but that doesn't mean that one will be readily available to actually buy.
- Even if you can find an identical machine, it will not have been used the same way. Imagine two cars, rolling off the assembly line one after the other at a car factory. These twin specimens may be the same make, model year, and have the same installed options. However, one car is delivered to a suburban family and the other to the fleet of a taxi company. After 10 years of use, these "identical" machines will be quite different: their wear patterns will reflect their much different working lives.
- For relatively rare machines, there may not be a thriving secondary market from which to obtain pricing information. How much would it cost to buy a 1931 Royal Enfield Bullet? It might have been years since one changed hands! Also, are the parties willing to disclose the price paid? How much is your time worth to obtain this information? Remember the concept of opportunity costs before embarking on a wild goose chase for something rare.

This amazing photo is a superb visual representation of the pace of technological change, even in just a relatively short period of time. From left to right are comparable circuits boards from the ENIAC, EDVAC, ORDVAC, and BRLESC-I computers. The timeframe between these four specimens is only 16 years (1946-1962), but look at the dramatic miniaturization taking place. When using a benchmark like the original

purchase price or replacement value (i.e., comparing a machine to itself), innovation is a key piece of contextual information omitted by the 50% Rule.
(image: US Army Photo)

You've Moved On

As time moves on, your needs will inevitably change. This is the biggest problem with automatically assuming an old machine (in the guise of replacement cost) should be revived or searched for in the used market. Machines amplify our purposeful intentions; any rule of thumb that doesn't take this into account is missing the point about why we employ them. Although general purpose, the ENIAC was designed to solve WWII-era problems: the calculation of artillery firing tables. In contrast, the increased power of the later BRLESC-I (Ballistic Research Laboratories Electronic Scientific Computer, launched in 1962) was needed to solve Space Age and Cold War problems involving nuclear weapons, missiles, and satellites. The ENIAC could perform 5,000 operations per second, but the BRLESC-I could do 5 million. Repairing or replacing an equivalent machine isn't an option when your needs have changed. It doesn't matter if we're talking about a supercomputer, toaster, car, or backhoe.

Having a machine that was doing useful work for you malfunction is a loss: the resources you must now devote to its repair or replacement can't be used elsewhere. While this is a drain, you should make the best of it. Since change is being forced upon you, take the opportunity to fully reassess your needs *vis-à-vis* the machine in question. Often, you'll find that things have changed and your purposes could be better served some other way. At these pivot points, I've often made the decision to upgrade or downgrade, finding a better fit with either a more or less capable machine. (Please see the section titled "The Need, Reassessed" in "Repair Or Replace?" for the full discussion of this point.)

I've also decided to do nothing: that is, to *neither* repair or replace, opting instead to simply cut ties and sell a machine for its salvage value. Especially for those things we employ for leisure (aka, "toys"), you may find that the "need" was weak and your life is better off without them entirely. Maybe you impulsively bought a boat that you thought you'd use every weekend. But, the maintenance hassles and your busy life resulted in the craft only being used twice a year. Perhaps you'd like the garage space back. Remember the old saying: "The two happiest days in a boat owner's life are the day he buys it and the day he sells it." The sentiment behind this adage applies to so many things, not just boats!

Lately, I've gained a tremendous satisfaction from reducing the number of "things" in my life. Your stuff has an overhead which may carry an unwelcome cost of ownership, especially if your life is already full. In these cases, a malfunction may be an invitation to simplify and regain the most precious resource of all: your time.

Why Would I Pay More For The Same Thing?

Even though replacement cost has some potential issues, at least we've found a basis for comparison that's internally consistent and up-to-date. Unfortunately, there's a huge problem with using replacement cost to calculate the 50% Rule: it doesn't make any sense! Here's an example that shows why: let's say you buy a refrigerator and it stops working after the warranty period expires (drat). Yours is a common model that's sold millions of units, so equivalently used replacements are easily had for $400. The estimate for repair is $240. The 50% Rule is:

$$r_{cost} \div m_{replacement} < 50\%$$

In our case:

$400 ÷ $240 = **60%**

If repair costs $240 and an equivalently used replacement is $400, repair is 60% of the replacement cost. In this case, the 50% Rule says to **replace.** What?! Think about what this means; we are supposedly comparing apples-to-apples. If you could pay $240 or $400 for the exact same thing, which would you choose? That's right, the cheaper one! When comparing equivalent items, as long as repair is less than the replacement cost, you should repair (and vice versa). For this reason, there's no way we can use replacement cost as the benchmark for the 50% Rule. Bizarrely, it recommends spending up to 100% more—for the same thing!

New

Perhaps instead we should be comparing the cost of repair to buying a new machine. This has great appeal, given the problems often associated with finding a suitable and identically used replacement. Whether on-line, in a local store,

or via a phone call to a sales representative, manufacturers make it very easy to buy new. Information for new products is available in abundance: delivery dates, specifications, and, most importantly, pricing details. Return policies are often generous compared to the "as is" nature of used sales. Warranties mitigate the risk of a buying a "lemon" and give predictability to service costs (at least for a limited time). There's much to be said for buying new.

However, again we run into the fact that choosing new as the standard for comparison of the 50% Rule requires a very important judgement to be made first. Why incur the additional expense of buying new if a used machine will adequately meet your needs?

Aye, here's the rub: with new as the standard, we're not comparing apples-to-apples anymore. A new machine will have different capabilities, warranties, maintenance costs, and longevity expectations. Presumably, we'd like to assign a dollar value to these advantages and see if the additional expense was worth it. The only thing the 50% Rule would seem to say about the matter is that used is half as good as new. That's a pretty crude way to evaluate your choices.

To see how utterly ridiculous things can get when using new as the standard with the 50% Rule, let's return again to our broken down car that needs a $2,000 transmission. While there's nothing in the new market that can hope to compare with our finely dented steed, if forced to choose an equivalent new replacement, we'll go with a lower-end 4-door sedan for $17,000. Let's see what the 50% Rule ($r_{cost} \div m_{replacement} < 50\%$) says to do:

$2,000 ÷ $17,000 = **12%**

Because repair is less than half of the cost of a new one, the rule says to repair. Sorry partner, them's the rules. Of course, you'd be an idiot to do that given that we've shown the market value of such a repair would incur a huge loss. The follies don't end there: keep in mind that the rule would say that we should consider any repair up to $8,500 (50% of $17,000). More nonsense!

Game Over

I'm going to stop here and declare the 50% Rule to be a failure as a practical decision-making tool. The rule requires a benchmark and, as I've shown, the ones offered all have serious defects. Using them could result in some very flawed outcomes.

If you want numbers to guide you, the value-added model of repair:

$$r_{value\text{-}added} - r_{cost} = r_{profit/loss}$$

seems like a much better alternative. At a minimum, it should be consulted to warn against expenditures that will not be recoverable when reselling. I think that's the biggest risk most consumers face and a frequently violated expectation. Because the market value of the machine after repair ($m_{post\text{-}repair}$) is required to make this calculation, it forces you to consider an important alternative for your money: equivalent replacements available in the marketplace. Finally, the value-added calculation is up-to-date, forward-looking, and incorporates the current state of technological progress, while avoiding the effects of inflation and the temptation to consider sunk costs.

But beyond the numbers, you need to remain in touch with your needs and how a particular machine is serving them. Have those needs changed? What value does a contraption bring to your life or business? How does that compare to the alternatives available (like new models)? These considerations should come before any repair or replace calculations are made, otherwise you will be led by the garbage-in, garbage-out nature of blindly following the results of a formula.

What Do You Think?

I'm perplexed that the 50% Rule is cited as a rule-of-thumb for consumers. If you have a good defense that addresses the problems I've uncovered, or points out something critical I've missed, I'm all ears. Tell me what you think in the comments section!

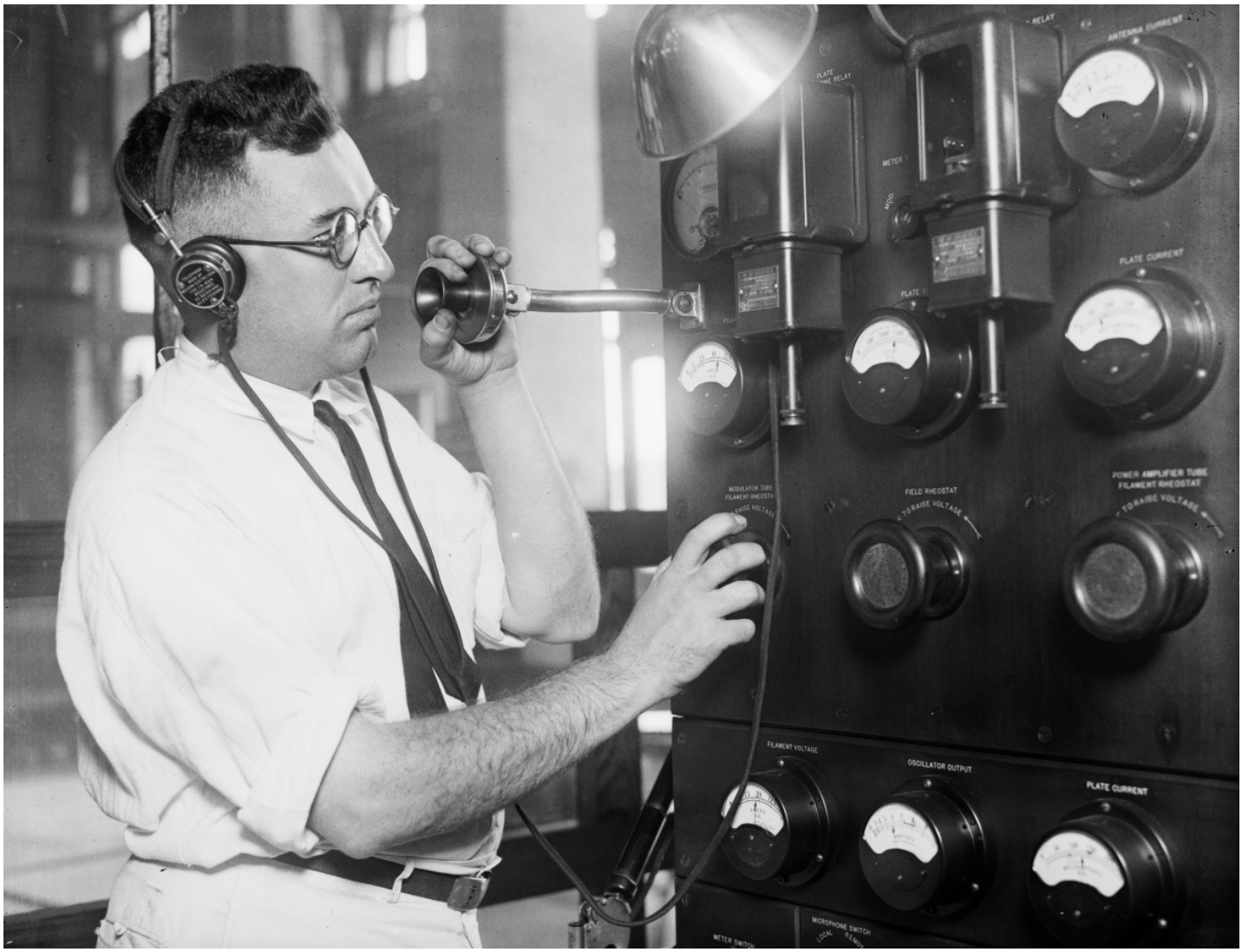
References:

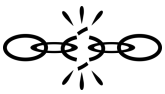
- Header image: Horydczak, Theodor, photographer. *Automobiles. View from front of wrecked automobile*. ca. 1920-ca. 1950. Photograph. Retrieved from the Library of Congress, https://www.loc.gov/item/thc1995012467/PP/.

The 50 Percent Rule: Repair Or Replace, Revisited was originally published April 25, 2014.

Notes:

Talking About Your Problems





> I think we always had more respect for the guy who came in and said, “I think I’ve got a problem,” versus someone who’d say “That piece of crap is blah, blah, blah.” Settle down, tell us what’s it doing. Let’s look at it...
>
> **Dan McCormick**

Once you’ve become a skilled troubleshooter, it’s time to look in the mirror and think about how you interact with the people you call on for help. What’s the best way to present your problems to others? Having been on both sides, sometimes solving issues and sometimes reporting them, I have some thoughts on the matter.

Mixing Fact And Speculation

Since you’ve made the effort to bring your problem to someone else, we can assume that you’re:

- Unsure about the cause of the issue.
- Unsure how to fix it.
- Both.

If you've taken the time to read *The Art Of Troubleshooting,* we can make one more assumption: that you have a keen interest in fixing things. So, let's sound the alarm and warn ourselves against the very thing we dislike when others bring their problems to us: mixing speculation with facts.

Let me explain with a story. Recently, some friends were getting their hot tub repaired (the water in the tub wasn't staying warm). When the technician showed up to look at the problem, he was told that the "heater was broken." A more skilled (or, if we were being cynical, honest) repairman would have responded to this with the right amount of skepticism: "How exactly do you know that the heater is broken?" However, this technician took the provided description as gospel and replaced the heater. It also turned out that this guy really liked to replace heaters, the procedure was squarely in his wheelhouse. Adding to the momentum was that a replacement heater was on his truck, all ready to go. Unfortunately, after replacing the heater, the problem persisted. On a subsequent visit with a different technician, it was discovered that the circuit board that *controlled* the heater was intermittently malfunctioning. The original heater was probably fine. Oops.

Let's go back to the start of these shenanigans and think about what went wrong. While it's true that the words "the heater is broken" were the ones actually spoken, the *meaning* was that the water in the hot tub wouldn't stay hot. Mentally, it's convenient shorthand to associate a particular machine function with a certain component. After all, "heat" comes from the appropriately named "heater." The cause and effect are easily bound together in your mind; if there's no heat, the problem seems obvious: the heater. There's another possible semantic interpretation, some people might use "heater" as a synonym for "the heating system," which includes everything the heater depends on: the thermostat, power source, etc. Unfortunately, people (like technicians) can take things *literally*: when they hear "the heater is broken" they might just go ahead and replace—the heater!

Be careful what you tell them, they might believe you!
(image: Jamiesrabbits / CC BY 2.0)

If You're Going To Speculate, Be Clear About It

The above example is a good illustration of what can unfold when a supposition is accidentally injected into a problem report. However, we don't want to have a blanket prohibition against speculating on the cause. That's no fun! Besides, what if you actually have a helpful piece of information that could possibly save a technician hours of work? It's okay to speculate, but you need to clearly mark what you're doing: "I think the heater *may* be broken because the water is cold. How can we tell if that's the cause?" This is a more cautious framing of the problem: it points out the symptom you're experiencing and what you believe is the source of the issue, while leaving room for other possibilities. It also explicitly states that you're interested in finding the cause. Yep, that would be nice to know.

Symptoms And Goals

Since you're paying someone for their expertise, consider forgoing speculation entirely. If I'm really a fish out of water with a particular machine, I focus on the symptoms and what I want to be able to do. That is, what is the malfunction preventing me from accomplishing? This kind of formulation sticks to what you know and clarifies your end goal for the technician: "The water in the hot tub is cold. I want it 101° F like it was last week, so I can throw a party." Including the overarching goal in your problem description is critical because it really doesn't matter what is broken and what is fixed: if the end result doesn't put the *hot* back in hot tub, you're not going to be satisfied!

Go With The Flow

Reaching out for help sometimes means interacting with a customer service agent in a faraway, exotic land (like Texas). You need to cultivate kindness and patience for these frequently frustrating encounters, where all you want is to talk to someone who understands your problem. But, there are ways you can make these interactions useful, even if the person you're talking to is struggling to help. You may have heard the expression, "Just the facts, ma'am." This was television detective Sgt. Joe Friday's famous admonition to his interviewees, a call to forgo opinions and focus on hard evidence. The problem is that there are an infinite number of facts you could relate to someone about a particular broken machine: its technical specifications, make, model, serial number, when you bought it and from whom, its operational record, repair history, recent changes to its environment, others things that might be wrong with it, etc. Any of these facts *might* be relevant, but then again they might not. Contacting an expert is useful because they know more about which facts are relevant and which are not. You may think you're being helpful by passing along a hidden nugget that will crack the case wide open. However, for every time I've pointed out something useful, I can think of another example where my instincts about the relevant facts were completely wrong. In those cases, my well-intentioned ramblings were actually a distraction to the technician's fault-finding process.

Therefore, if I'm going to put myself in the hands of experts, I like to learn what *they* think are the relevant facts. I do this by letting them guide the investigation process. Even when I'm dealing with someone who I think is reading from a script, I like to give them the benefit of the doubt and let them lead (at first). If I find myself driving the interaction, I fall back to **symptoms and goals:** "the machine is doing X, and is preventing me from doing Y." Of all the facts, these are the most important. Keeping an eye on the end goal also opens the door to alternatives: just by saying your purposes out loud, your mind will start looking for workarounds.

You might be tempted to blow off a scripted response to your problem, but I suggest you go with the flow instead. If you're against scripts, then you're also against things like troubleshooting trees, which are essentially the same thing. Sure, customer service scripts are designed for the "lowest common denominator," but that frequently is *you*. When it comes to machine problems, be wary of thinking that you are a beautiful or unique snowflake. I've often been surprised at the number of times my issue has been resolved by executing the scripted advice during one of these sessions. When asked to verify something you think is obvious, you may be tempted to scream "This is foolish, I want the advanced techniques! I think the problem is related to…um…wait…sorry, it actually wasn't plugged in. Thanks."

Even if canned scripts delivered by a underpaid cubicle dweller fail to deliver a solution, just having a conversation about your problem can lead you to the promised land. There's something magical that happens when you're forced to communicate the ideas in your head. It seems strange, but I've figured out the causes of problems just by explaining them to others. I don't have a peer-reviewed academic study to offer on this point, so I'll just speculate why this happens. To start with, having to explain something to someone else gets you out of your head and shifts your focus externally (to the understanding of the other person). Next, telling the "story" of your breakdown, from beginning to end, exposes the holes in your knowledge. If it's not clear why A → B → C, it'll be even more apparent when you attempt to communicate it. Have you ever started to explain something you felt very sure of, only to realize mid-way through that you don't quite understand it after all? It often takes saying things out loud to trigger these discoveries of what is uncertain. Finally, I mentioned the power of "stupid" questions to facilitate breakthroughs in "A Different Point Of View." Often, when it comes to interacting with Customer Service, if you want those good "stupid" questions, you can have 'em by the truckload!

Before Talking With Customer Service

Even though I've pointed out many positives, I'm not giving a blanket endorsement to always pursuing aid from Customer Service. Getting outside help is a strategy like any other, and therefore must rise to the top versus all the other alternatives. As always, Opportunity Costs should be considered: what else could you be doing with your time? While I've shown that interacting with Customer Service can frequently be useful (even if unintentionally), it still must meet the standard of being the *best* strategy in your universe of possibilities.

Initiating contact with outside help has overhead: in addition to consuming time, some companies will charge you to speak with their support teams. That's why I like to push my investigation to the point just before the onset of severe diminishing returns. Recognizing where this happens is a judgement call. Now, I have a good feel for where this line is: it's usually after all of the "basics" have been tried and I've entertained (and shot down) some more exotic theories. Talking with Customer Service can be a welcome change of pace when you've hit a plateau like this. Just like taking a walk around the block, these interruptions can be a useful distraction.

Finally, I always like to have my ducks in a row before calling for help. A good description of the problem. Check. A series of steps that replicates the problem reliably. Always helpful. The better prepared you are when seeking outside help, the more likely the interaction will lead to a speedy resolution.

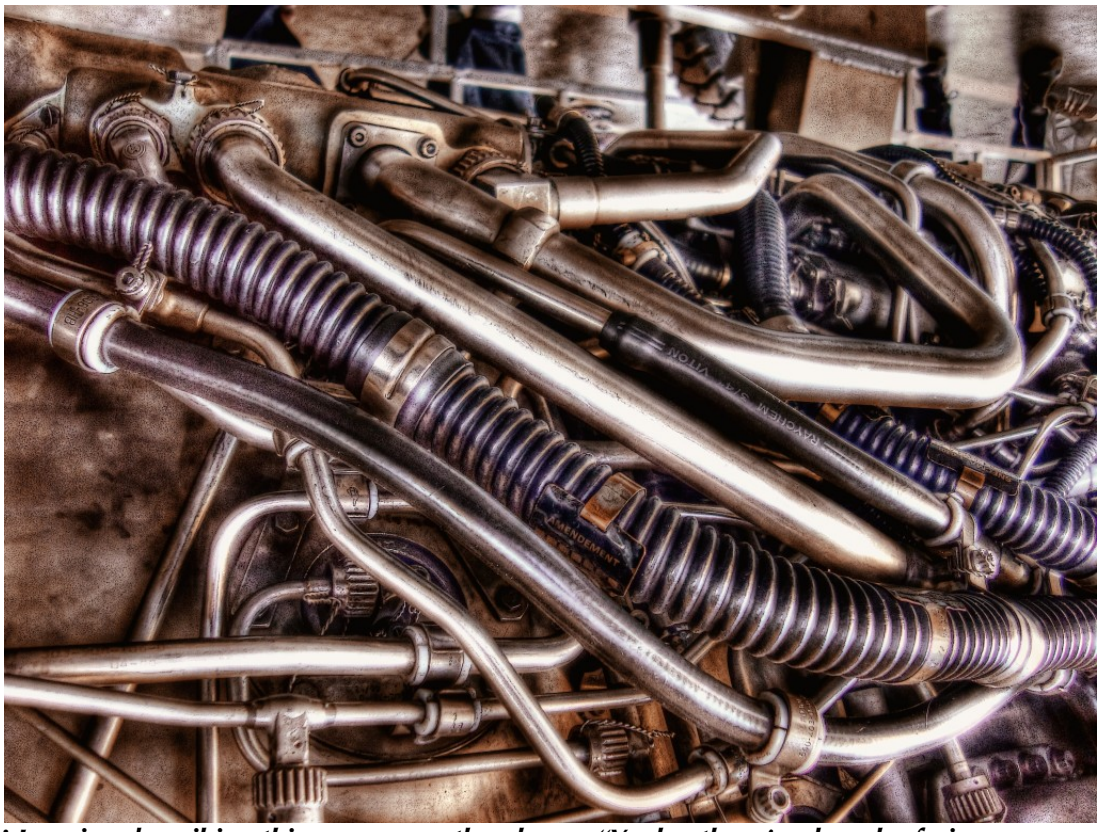


Photo op! Imagine describing this scene over the phone: "Yeah…there's a bunch of pipes…connected to… other pipes."

(image: Victor Camilo / CC BY-ND 2.0)

Picture Perfect

I named this piece "Talking About Your Problems" for a reason: to highlight the pitfalls associated with verbalizing technical troubles. In addition to the ideas above, you should practice the methods presented in "Skillful Questioning" to become a truly proficient language detective. That's great, but seize the opportunity to do an end-run around words, bypassing these problems entirely. Any time there's a decision to be made between *talking* about evidence and *presenting* evidence directly, always choose the latter.

In the world of fiction writing, there's a concept called the "omniscient narrator." This is a point-of-view the author writes from that has access to all the events and dialogue within the story. Jumping from scene to scene or telling the reader what is going on inside a character's head requires this God-like power. When it comes to reporting problems to a fellow troubleshooter, *you* are the narrator. Unlike Charles Dickens or Ernest Hemingway, your narration skills might be lacking, so save them for that brilliant screenplay you're writing. Of course you must speak, but always supplement your words with primary sources when you are able. Show them pictures of the failure, have them read the relevant documents, or take them on-site so they can conduct their own inspection.

Phone-based technical support is particularly vulnerable to the unreliable narrator problem. I've helped a lot of people over the phone, and so I know that a photo (or screenshot, when giving IT help) can be the quick path to a solution. When I was green, I would sometimes think to ask for a photo only after a long and winding conversation that was

going nowhere. Seeing the evidence first-hand was stunning and it was interesting to compare it to what was actually related by the "narrator" during our talk. Often, a key detail was misspoken (or misheard by me) at the outset, taking my questioning nowhere. Or, the person focused most of their description on an irrelevant part of the machine (not that it was their fault, if they knew what was relevant they wouldn't have needed my help). These problems melt away when you can put the actual evidence directly into the hands of someone trying to help you. There's a reason everyone knows the saying "a picture is worth a thousand words."

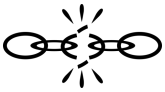
References:

- Header image: Harris & Ewing, photographer. *Telephone Machinery?* United States, 1925. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016894321/.

Talking About Your Problems was originally published June 21, 2013.

Notes:

Starting Over: Rebuilding And Reinstalling



> I love Larry Wall's quote about being lazy. Lazy in the right way. I think that applies to coding in general and troubleshooting for me is often troubleshooting code.
>
> I do a lot of triaging: “How important is this?” You're never going to get everything perfect. Sometimes, it doesn't matter if you didn't name the printer quite right...and sometimes it's time to reinstall everything from scratch so you get a perfect system. But knowing the difference between those two is the Art of Laziness.
>
> **Karl Kuehn**

You’ll recall that the standard troubleshooting exercise has 2 steps:

1. Identifying the problem
2. Executing the fix

This simple problem-solving model is useful because a mature understanding of these two steps leads to the possibility of forgoing either one.

> Less and less do you need to force things,
> until finally you arrive at non-action.
> When nothing is done,

> nothing is left undone.
>
> **Tao Te Ching (verse 48)** [1]

Learning to refrain from harmful or inefficient action is a difficult skill to cultivate, but it separates the amateur from the Troubleshooter. Not every malfunction can be fixed with a wrench, nor with money, and there are conundrums which are best left alone and worked around instead.

When it comes to the value of non-action, I've discussed several strategies that allow you to be ignorant of a malfunction's specific cause, effectively skipping step #1. Many solutions do not require that you understand the *why* behind the failure. Under this banner we include strategies like the restart and "shotgunning" (replacing several components at once). While the restart often works, when it's chosen in lieu of an investigation, the technique illuminates little about the origin of a problem. In effect, it's a substitution that favors the pragmatism of getting back to work over furthering your understanding.

Likewise, fixing something is also optional: always keep in mind that pursuing step #2 is not required. Knowledge gained during the problem discovery phase may show that a repair would be too slow, too costly, or perhaps even impossible. These scenarios provide the rationale for choosing to replace a failed machine instead of fixing it. Or, even better, you discover a workaround or optimization that completely obsoletes the failed component. That is, neither repairing nor replacing, but instead reconfiguring a particular workflow to do without.

With these concepts in mind, let's go all the way down the path of blissful ignorance. Within the context of repair, a strategy that chooses to avoid costly knowledge in favor of expediency is **starting over.** Let's examine the costs and benefits in both the digital and analog realms.

The Price Of Knowledge

I've previously discussed repair as the movement between the current *broken* state back to an ideal *operational* one. However, when making this transition, troubleshooters often rely on the natural advantage conferred by the *nearly-working* state of a recently broken machine. If a car stopped running just yesterday, or you were able to send an email 5 minutes ago (but not now), then it's likely that you are looking for just a single problem to fix.

In my experience, this is the core of troubleshooting: looking for that lone thing that is preventing a machine from working. The nature of the almost working machine leads to a bias for minimalism in your repairs, and also to the key principle of Change Just One Thing At A Time.

While repair may be an attempt to restore a machine back to the manufacturer's ideal, we typically stop work much sooner than that, choosing instead to focus on whether the machine can continue doing useful work. The alternative would be costly madness: to be absolutely sure that a machine conformed to the ideal, you would need to check every single component for correctness.

For example: if your car won't start, and replacing the battery makes it run again, you don't then check every screw, spark plug, fuse, hose, and weld. Although we may desire to know that everything else is in perfect working order, the price of that knowledge is simply too high. As long as a machine is meeting our needs, we're often quite happy to *infer* that the majority of the components are okay.

The Domestic Deleter

The digital world presents a different set of tools for ensuring the correctness of a system, allowing us to achieve a perfection that would be costly and repetitive in the analog world. It also means that starting over can be a viable problem-solving recipe.

To illustrate, let me tell you a story from the trenches of IT. While I was looking for a job after I graduated from college, I did some freelance consulting. I worked with a number of small businesses, which led to referrals to help people out with their home computers. As I was invited into these people's homes in the evening, these jobs were always very interesting. During these house calls, not only did I get a chance to hone my tech chops, but I also got an interesting "slice of life" view into the local community.

Unfortunately, sometimes it's best to start over (Waffle House Index level: Red).
(image: Carol M. Highsmith / Library of Congress)

One client I worked with had that perfect blend of frustration and comedy that made you laugh while shaking your fist at the heavens. She also made me question aspirations I had about turning my little consulting business into a full-time career; after this episode, I kicked my job search into high gear. At first glance, this woman's to-do list for me was nothing special: installing some software, troubleshooting her Internet connection (dial-up!), etc. However, something about her computer was…strange. I hacked away in vain for a while and then sat her down for a little Q&A. We had the following conversation, which I was not prepared for:

> **Me:** "I don't understand what's going on with your computer. It's very unstable and I think files are missing…"
>
> **Client:** "Well, whenever I come across a file and I don't know what it does, I delete it."

> **Me:** "What?! That must happen a lot."
>
> **Client:** "Not really, I'm quite busy and don't have time to look in every folder."
>
> **Me:** "For your computer's sake, that's a relief."

I then calmly explained why her "deleting what I don't understand" strategy was problematic. Convinced by my air-tight logic (and exasperated tone), she promised to change her eradicating ways.

But, what do I do now? What exactly did this lady delete? She had been at it for a while and couldn't remember which folders she touched. The default installation of a modern operating system can include hundreds of thousands of files. Applications can easily contribute an equal number. I knew the work required to figure out precisely which files were missing would be beyond tedious (yes, I checked the "recycling bin," but she was thorough in her efforts). There was only one good solution, and that was to start fresh. So, I convinced her to let me reinstall Windows from scratch.

Reinstalling an operating system may be time-consuming, but it requires little effort: the installer does all the hard work for you. Beyond supplying some basic information like your language, location, and keyboard type, it takes care of the rest, making sure that all those files end up in the right place. I awkwardly watched TV with her husband and the family dog in the living room while waiting for the install to finish.

> **Husband:** "So, you want a beer?"
>
> **Me:** "Oh no…"
>
> **Husband:** (shouting to the next room) "Honey, can he have a beer?!"
>
> **Client:** (shouting back) "No, he's working on my computer!"
>
> **Me:** "That's okay, I'm fine."

Welcome to the glamorous world of IT.

How far do you take deconstruction and choose what to salvage?
(image: Carol M. Highsmith / Library of Congress)

Levels Of Rebuilding

If you're having a problem with your refrigerator, and don't know what's wrong, it would be folly to replace every part. It might fix the problem, but pursuing this strategy would come at a terrible price in terms of time and materials. In the digital domain, however, the costs are different. Replacing everything might be expensive for a refrigerator, but this is exactly what reinstalling software accomplishes. The nature of ones and zeros, which can be copied perfectly and automatically with ease, means you can rebuild something from scratch with just a keystroke.

In the abstract, when is starting over a good option for repair? We need the following conditions to be present:

1. The cost of problem discovery is high.
2. A machine model that is easy to replicate or enforce correctness upon.

You can see that my situation with the woman who liked to delete files fit this mold perfectly. Manually figuring out which files were missing would have been costly, boring, and time-consuming. I wasn't about to move in and become a regular on their couch, even if I did get a say about what we watched on TV and the privilege of having a beer! At the same time, Windows' automated installer ensured that the operating system could be restored perfectly and with little effort, fulfilling the second requirement.

Examples in the analog world are harder to come by: that is, repair situations that favor wholesale rebuilding over diagnosis. A Formula One racing team may completely rebuild an engine after only 500 miles of driving, but that's done in the name of preventative maintenance. Purely as a troubleshooting strategy, rebuilding an entire physical machine from scratch is typically not cost effective because buying a replacement or skillfully isolating a problem using logic will usually win the cost battle. The reason why gets back to the economics of troubleshooting: repair labor is simply not as efficient as factory labor.

What is relevant to both the digital and analog worlds is that you have a decision to make about which level you will choose to rebuild. Using a malfunctioning computer as an example, you could start over at any of these various levels:

- Hardware
- Filesystem
- Operating system
- Applications
- Configurations

In terms of the extent of the rebuild, these are listed in order of most-to-least, with each level being inclusive of the next. If you reformat your disk (filesystem level), then you must also reinstall the operating system, plus applications, which subsequently must be configured, etc. I've done rebuilds at each of these levels, depending on the situation.

To show you that this concept is universal, we could also make the same type of list for rebuilding a house. Imagine a home that was damaged, perhaps in a storm or fire. You would also face the same dilemma about which level to begin rebuilding:

- Footings
- Foundation
- Framing
- Plumbing/electrical/HVAC
- Finish work (drywall, trim, etc.)

Just like in the computer example, going from top to bottom, each of these levels is inclusive of the ones below. Rebuilding from the footings would imply that everything else would also need to be restored as well: the foundation, framing, plumbing/electrical/HVAC, etc.

Because the scope of the rebuild expands as you go up through these levels, the cost-effective way to proceed is to see if rebuilding at the smallest level will meet your needs. In our computer example, that would be to consider the list in reverse order: Configurations → Applications → Operating System → Filesystem → Hardware. Indeed, experience confirms this tack: refreshing configuration files are often a quick fix to application level problems!

Nothing in your way: a greenfield project has no prior constraints. Except reality, of course…
(image: Christian Lendl / CC BY 2.0)

Clean-Sheet Innovation And Rebuilding…Trust

There's a fine line between engineering and troubleshooting, so a rebuild can also be an invitation to rethink a machine's design. Especially if you're working on a custom-built system, a rebuild is an opportunity to make improvements that have been discovered after a machine has been in use. This happened all the time in my software career: a decision to remake a faulty or bottlenecked component opened up the door to big breakthroughs. Rebuilding is a great way to free yourself from the constraints of prior decisions, which were made in a different context that likely has changed.

Finally, there's one more aspect of rebuilding that transcends optimization. In my tenure as CTO, I managed many incidents affecting security: from virus infections on employee computers, all the way to a malicious hacker intrusion that nearly bankrupted the company. In these cases, rebuilding went beyond an advantageous troubleshooting strategy that leveraged the automated correctness of software. Instead, it was a vital bridge that got us back to a system we could **trust**.

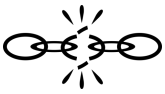
References:

- Header image: Harry Dona, photographer. Retrieved from Unsplash, https://unsplash.com/photos/qz1DQ7sKxZE.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 48.

Starting Over: Rebuilding And Reinstalling was originally published October 19, 2014.

Notes:

Border Lines



> I'm so dependent on people to explain what's going on, to get to the bottom of a problem.
>
> **Jeremy Sheetz**

In the garage of my apartment building there's a green tag hanging from the phone wiring box. It's one of those things that I see every day while driving in and out of my parking space, part of the myriad stimuli that enters my awareness and immediately exits again. When taking the trash out the other day, it caught my attention again, and so I decided to finally see what it was. I flipped it over and saw big letters that read:

"MPOE"

For a moment, I wistfully stared off into the distance and channeled Obi-Wan Kenobi: "MPOE. Now, that's a name I've not heard in a long time… A long time." Walking back upstairs, I had flashbacks to some of the epic telecom battles I've fought over the years. The more I thought about it, the more I realized that the MPOE represents a critical concept for the Troubleshooter.

Assigning responsibility: on which side of this dividing line does the problem lie?
(image: Wikijimmy / Wikimedia Commons)

The Handoff

If you aren't familiar, MPOE stands for "minimum point of entry." For a telecom company, it's the end of the line, the official point where their network terminates and a customer's begins. This "line in the sand" is critical because it creates a clean and predictable way to assign responsibility when a problem arises.

If you've ever had a telecom technician visit your home or business, they likely want to start their investigation outside, at the MPOE. Testing to see if the network works up to this demarcation point is a play straight out of "follow the chain." Inserting a probe where a line enters a customer's premises attempts to answer the question: "Is the problem upstream or downstream from this point?" Answering this question leads to some very useful information.

If the network is not working at the MPOE, the implication is obvious: the problem lies somewhere within the telecom company's domain. This is the tactic of isolation in action! Note too that testing at the demarcation point also allows proper responsibility to be assigned: the customer isn't expected to either find or pay for problems that originate within a provider's system.

Conversely, if everything up to and including the MPOE is in working order, then the conclusion is also straightforward: the problem must be somewhere within the client's infrastructure. If you're a customer, it would be bizarre to be given a hard hat and asked to get in a cherry picker to help troubleshoot AT&T's network. Of course, the reverse is also true: unless you pay them, AT&T shouldn't have to chase down problems with the wiring inside your home or business. That's your responsibility.

The MPOE is an interesting entity because this junction is much more than technical: that box hanging on the outside

of your house is a boundary marker with social, legal, and business-related dimensions. The decision to troubleshoot is good for your personal development precisely because it forces you to understand and engage whatever and whomever lies on the other side of these dividing lines.

Some borders are wide open…
(image: Swedish National Heritage Board)

Border Crossings

Borders have always fascinated me. When I was young, we did a fair amount of driving around the good ol' USA on family road trips. I distinctly remember being confused when crossing between states. A sign saying "Welcome To The State Of X…" would pass by, but the view would remain largely the same. I guess I was expecting the landscape to change drastically, so much that the dividing line would be *visually* apparent. Sometimes, I'd miss the sign and only realize I was in a different state hours later.

Maybe my disappointment also stemmed from those cartoon maps that they make for kids, the ones where every state has an icon representing what it's known for: a potato on Idaho, a car on Michigan, etc. Is it any wonder I was disappointed after crossing these borders, only to find that Texas wasn't awash in a sea of barbecue sauce or that Wisconsin wasn't made of cheese?

I also remember visiting Nogales, Arizona and Nogales, Sonora when I was growing up. These conjoined cities straddle the USA/Mexico border. To my naive mind, this transition was mind-blowing: we had walked just a few minutes and suddenly everyone was speaking a different language!

As a broad concept, recognizing where entities meet and transition is crucial to navigating the world. Because interesting things often happen at these dividing lines, the implications are rich for the Troubleshooter. If you fix things long enough, you will eventually have to negotiate the following types of boundaries:

- **Contractual:** this is where you can see where the problem lies, but the *responsibility* for fixing it belongs to someone with whom you have a standing agreement. The example of the MPOE neatly illustrates this division: you pay the phone company every month with the expectation that they will provide a working service up to this boundary marker. Just like they will happily terminate your account for non-payment, you too must demand that they uphold their end of the deal. In that sense, you don't ever "fix" the phone company's technical problems, except by enforcing the letter of your pact.
- **Business Relationship:** this is often the flip side of the contractual dimension, where the maintenance of goodwill demands you fix something that is clearly *not* your responsibility. This is an incursion into your sovereign territory,

but one that needs to be tolerated. We used to get on the phone all the time to help our clients ingest the data we produced, often troubleshooting their systems along the way. This wasn't necessarily contractually mandated, but everything is on the table for the Big Client.

- **Organizational:** this is the case where a problem is internal to your organization, but fixing it requires resources outside your local sphere of influence. Instead of turning a wrench, these repairs require the tools of advocacy, seeking out those who have the power to help you and convincing them that they should.
- **Legal:** just because you can fix it, doesn't mean the law allows you to do so. Some repairs can bring up interesting legal questions, like those involving reverse engineering. You may have signed an agreement that prevents tinkering, as with many closed source software products. The bottom line is that if you don't own it, fixing it may require permission.
- **Social:** people can get territorial about…well, pretty much anything (see also: Milton's red stapler from *Office Space*). If you're cavalierly fixing problems in someone else's domain, there can be social repercussions.

…and others are vigorously defended.
(image: U.S. Army / Wikimedia Commons)

Border Skirmishes

When I was CTO of Discovery Mining, all these boundary types came together during my epic struggle to procure a high-speed data line connecting our office in the Presidio of San Francisco to one of our colocation facilities in Bayview (about 6 miles away as the crow flies). Getting this point-to-point circuit installed and tested was a series of maddening Kafkaesque vignettes that took me through a full range of emotions: optimism, confusion, anger, despair, numbed indifference, and finally anti-climatic triumph.

From start to finish, it took me 11 months to coordinate the installation of this line. **11 months.** I thought about it every day during that period. This lengthy outcome was a severe violation of my expectations, given that I had seen the snazzy marketing page on the telecom provider's website promoting the product. They made it seem like this was just something you ordered and then it arrived shortly thereafter, like getting a pizza delivered. Unfortunately, "layer 2 metropolitan gigabit ethernet connection" was not on Domino's menu, or I would have gladly ordered it from them instead.

The cast of characters in my saga included a major telecom firm, a federal agency, a state regulatory commission, the

building owner of the colocation facility, another company that we were sub-leasing space from, our Internet service provider (ISP), and us. Did I miss anyone? Within this group was a tangled web of business, legal, and technical relationships. In retrospect, half the battle was just figuring out who had jurisdiction to make a particular decision. Some territory was aggressively controlled while ironically, in other situations, it was nearly impossible to figure out who was in charge and could move my project along!

Witnessing the little skirmishes between these various entities as they asserted control over their territories was eye-opening. For example, take the meet-me-room, where our circuit was to be handed off from the telecom firm to our ISP. At the time, they were embroiled in a dispute with the colocation building owner over cross-connect fees and they absolutely refused to pay to connect our new line. It was a standoff that lasted for several weeks and neither side would budge on their demands. Also, getting this circuit somehow triggered a review of the contract by the California Public Utilities Commission. Weeks lost there too.

Although this line required a fairly simple *physical* connection (strands of fiber optic cable from point A to B), mostly already existing, the route unfortunately snaked through all these various fiefdoms. Some of these regimes were sympathetic allies and others were hostile territory. Not all travel looks like a photo shoot from an issue of Condé Nast.

During this frustrating process, there were also social boundaries that I could and could not cross. Since I was paying them, I could justify laying into our ISP for chronic mismanagement of the project and lack of communication about what was happening. So, I did.

On the other hand, getting fiber pulled to our office building was dependent on the whim of inaccessible bureaucrats with whom I had little influence. It was a mighty struggle getting them to answer *any* form of communication (phone messages, emails, smoke signals, etc.), with weeks sometimes lapsing between responses. With those people, it was cheerful and patient persistence at every turn, as I could never afford to be publicly angry with them.

Even after the circuit was turned up, we spent a few more months troubleshooting problems discovered by actually using it! There was one particularly embarrassing rollback that I had to stomach; it occurred after a failed switchover rained down chaos on our operations. That was the beginning of my education in networking arcana, with a deep-dive into the meaning of the three letters M-T-U.

Flipping the switch to fully utilize our new line for actual business was a day I'll long remember. It occurred **399 days** after my first enquiry. Who would have thought that ensuring a clear and stable path along a tiny sliver of glass, nearly weightless, would have been so difficult?

Bon Voyage! Remember Your Passport.

In life and business, we often rely on systems that are beyond our direct control. To get help, repair will involve trips to neighboring territories. Leaving your immediate locus of control requires human skills because your influence diminishes as you cross these border lines.

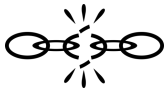
References:

- Header image: Jones, Suzi. *Fence with Smooth Wire*. 1978. July. Photograph. Retrieved from the Library of Congress, https://www.loc.gov/item/ncr000222/.

Border Lines was originally published November 22, 2014.

Notes:

If You Have To Force It, Something Is Probably Wrong



> The lessons you learn best are the ones that hurt the most.
>
> **Jamie Karrick**

I like to wander around the motorcycle club I belong to, observing and sometimes helping the other members with their projects. A second set of eyes or hands, along with a "dumb" question or two, often helps to get someone going again when they are stuck. During these fix-it sessions, the repair principles I've written about are brought to life and new ones are always bubbling up to the surface.

Take a recent session, where a fellow member had brought in his Yamaha. Out of the corner of my eye, I saw him carefully remove a substantial portion of the front fairing. You can usually tell when someone has hit a roadblock, either by the swearing, or the confused look on their face as they stare blankly at the parts strewn about them. I could see this guy was struggling with whatever he was doing, and so I meandered over to see if I could help.

"When in doubt, get the hammer out" is great advice—except when it's not.
(image: Esther Bubley / Library of Congress)

He explained that he was trying to get into the assembly that held the two headlights, so he could replace a burned-out bulb. The clips that held the plastic enclosure were difficult to pry open, so we tried successively larger screwdrivers and corresponding amounts of force. However, the clamshell assembly would not yield to our efforts.

Whenever I'm assisting on someone else's bike, I'm a bit more careful than if it were my own. If there is a risk of breaking something, from a wrench turned too far or the misplaced strike of a hammer, I figure that's their decision to make. They must deal with the consequences and I can just walk away, so they need to be in the driver's seat at those critical moments. It seemed like we were getting to that point, so I paused and said "This is hard…Yamaha really made it difficult to change a stupid lightbulb. Are you sure this is how to get in there?" The owner voiced his agreement and took a step back to reconsider. If we stayed on our current course of action, the next logical step would be to apply forces that would break the plastic housing.

I stepped away to get a drink, and when I returned a few minutes later he had solved the problem. The owner of this Yamaha had mistakenly thought that the lightbulbs were to be replaced from the front. However, they were supposed to be accessed from the rear. The smart engineers at Yamaha had even provided an easy-to-remove cover for this purpose (as if they anticipated this very problem!). With this pivotal discovery, just a few minutes later the lightbulb had been replaced. Now, all that was left was to rebuild the front part of the bike. Frustrated at the time wasted, he lamented: "I *knew* it shouldn't be that difficult!"

Forcing It

> Rushing into action, you fail.
> Trying to grasp things, you lose them.
> Forcing a project to completion,
> you ruin what was almost ripe.
>
> **Tao Te Ching (verse 64)** [1]

As I reflected on this particular fix-it drama, my mind flashed images of the many times that I had forced something, along with the mixed results. In life, many rewarding things we do are hard, taking copious amounts of effort. To say "don't do anything that is difficult," would be to cut out a path that has led humanity to many glorious triumphs. However, repair has a special relationship to the application of brute force. You can't rule out its use, but it should be marshaled with caution. I also began to think about what *should* be hard about troubleshooting, and how that differs from neighboring fields like engineering and invention.

First, we need to think about the nature of repair and its position at the end of the chain of invention → engineering → manufacturing. As a general rule, the amount of wasted and forced effort decreases as you go down this chain; also, *what* is difficult about each stage is dramatically different. Think about Edison's search for a suitable lightbulb filament: before trying Japanese bamboo, his team laboriously rejected over 1600 candidates! Edison was speaking from deep experience when he talked about the "99% perspiration" part of creation. As inventors desire to prove their ideas by building working prototypes (and fixing them when they break), they experience a microcosm of the hardships endured by engineers, manufacturers, and troubleshooters. Propelling yourself from a moment of inspiration to a working example requires resolution of *all* the problems along this entire chain.

Engineers have their problems too, but at least the burden of initially proving that a particular idea will be feasible is not among them. Instead, they can *build* upon the discoveries of scientists and inventors by making certain assumptions: that an airfoil will create lift, that copper wires will carry electric current, etc. You'd think that would make their jobs easy, but engineers are asked to deploy these existing ideas in new contexts. Will this airfoil provide *enough* lift to get this particular airplane off the ground? Will this copper wire be *sufficient* to carry a specific amount of current from the breaker panel to the outlet? This frustrating intersection between the known and unknown is what makes engineering hard.

Manufacturers and builders take the fruits of inventors (via engineers) and attempt to replicate their designs economically. They aim to take abstract designs and put them in the hands of your everyday consumer. Discovering and translating Machine Models into useful product concepts may be outside their purview, but of course manufacturing at any scale is rife with serious challenges. Just because some egghead can sketch a design on a napkin or click around fancy CAD software, doesn't mean it's going to be easy to efficiently replicate that model, over and over. Manufacturers must solve the problem of marshaling resources, both natural and human, in the right quantities, in the right way, and at the right time. A schematic says nothing about how to properly outfit a factory, train and retain workers, manage a supply chain; nor does it address legal matters, finances, and the myriad other details needed to consistently churn out widgets by the gazillions.

What's So Hard About Troubleshooting?

After all this, you could say that troubleshooting is the easy part. Of course, it isn't. Even so, it calms me to take a moment and understand all that came before the fateful moment I picked up a screwdriver and decided to give repair a shot. A flash of brilliance or a happy accident started it all, then the idea was vetted through the perspiration of scientists and inventors. Next, an engineer came along and translated the concept for a specific purpose. Finally, a manufacturer or builder made it real.

Now, it's up to you to make it work again. Troubleshooters interact with the Machine Model in a restorative way and have one enduring advantage: fixing anything implies that **the system functioned at one point in the past.** Compared to bootstrapping a working *something* from just an idea, this is a huge head start. Repair, at its most efficient, aims to interact with *only* the broken parts of the machine.

All this means that effective troubleshooting is more often an internal struggle than a grand show of force. Once, I remember describing a car problem I was having to my Grandfather, an auto mechanic. He closed his eyes, tilted his

head back, and began to think out loud: "ignition…accelerator…fuel pump…turns the…" Grandpa definitely was expending effort, but not the kind that involves rolling up your shirtsleeves and grunting: this intense investigation was all happening in his mind. The hard parts of repair are figuring out what's wrong, understanding the machine's design (at least the parts that are integral to the problem), and then choosing from among a wide spectrum of fixes. Here, brains triumph over brawn.

Swing away, but don't say I didn't warn you…
(image: Alfred Harrell / Library of Congress)

Enough Talk, Pass Me That Hammer

Sometimes the path is meant to be difficult. While reading this, I'm sure the seasoned veterans among you thought of times when elbow grease was necessary. You can't definitively say that the application of great effort or force is bad. The lesson here is that an aware Troubleshooter pauses and reconsiders when something gets hard. The nature of repair is treading well-worn conceptual and physical paths, so the application of brute force or sustained exertion should be surprising. While you can take back an errant thought, an angry swing of the hammer can leave lasting scars.

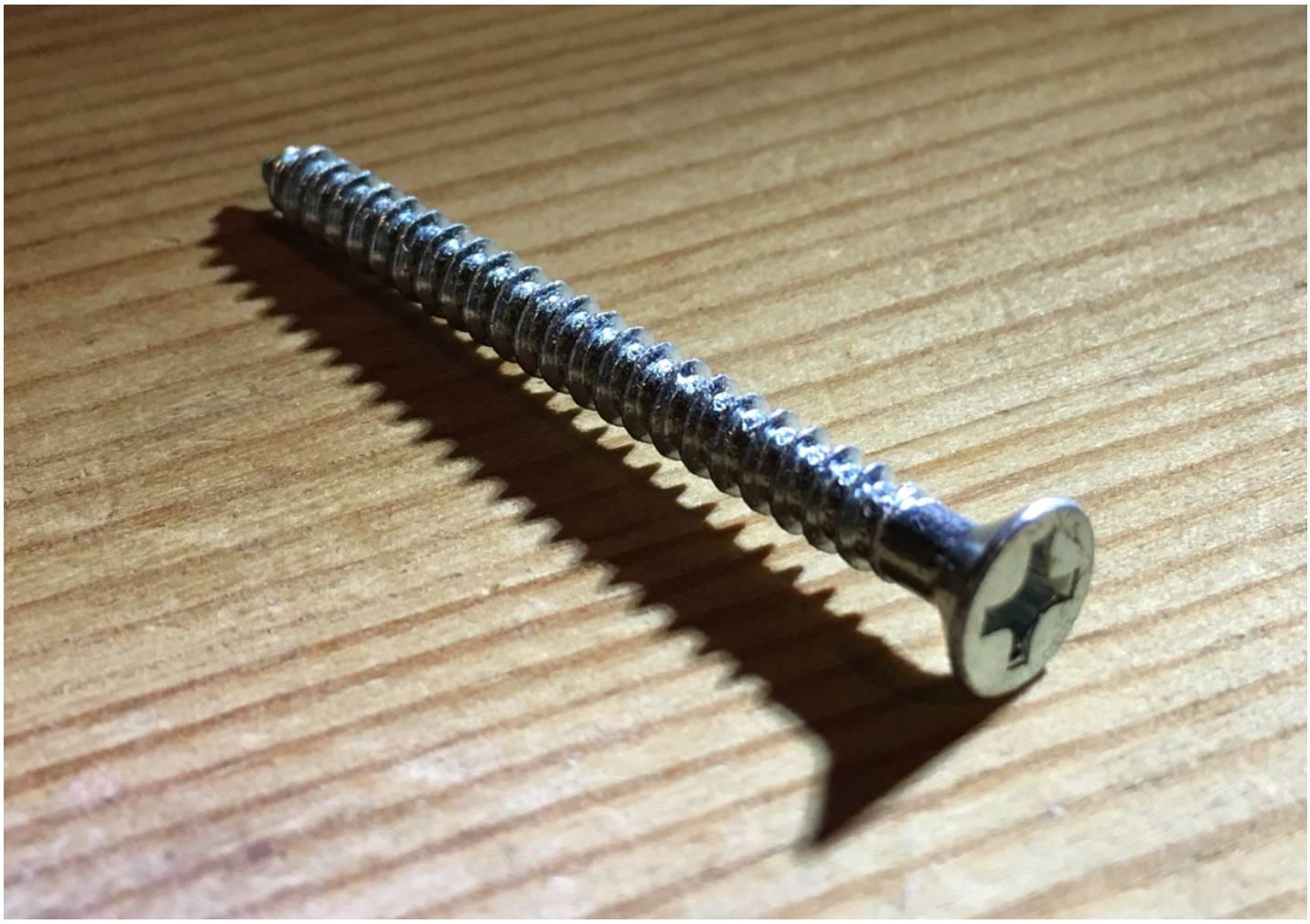
References:

- Header image: Lee, R., photographer. *Migrant boy who is somewhat of a mechanic checking up the lighting wires of their improvised truck which will carry them to California. Near Muskogee, Oklahoma*. United States, Muskogee County, Oklahoma. 1939. July. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017783819/.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 64.

If You Have To Force It, Something Is Probably Wrong was originally published May 21, 2015.

Notes:

What We Bring With Us: “I Want One Of These”



> Some people who like to replace parts are good guessers, and I never liked to do that. Some had the theory: “If I guessed the first time and I'm right, I fixed it,” instead of diagnosing the problem. I wanted to know what was wrong before I replaced a part. I was not good at spending other people's money.
>
> **Dan McCormick**

If “one good test is worth a thousand expert opinions” then it’s also true that “one broken part is worth a thousand detailed diagrams.” When it comes to seeking replacement parts, have what you need *in hand*. That’s not a fancy metaphor, I mean to physically put the broken whatever in your paws and bring it with you to the store or junkyard.

This simple act cuts through the potential confusion about what exactly is needed and will prevent repeat trips. Most stores will have tens, if not hundreds, of options for replacement. If you go looking for a lightbulb at a typical big-box store, you can expect a whole aisle of choices. You can easily tell a lightbulb apart from a lawnmower, but can you distinguish the lightbulb you need from the 346 other lightbulbs on display? What was the wattage? What do the threads look like? What color temperature do I need? Hmm…I see there are bulbs here with different heights. Will this taller one fit in my fixture?

When I need a replacement for something, I usually march down to the hardware store with the broken one in my pocket. I’ve saved a lot of time by simply showing the busted thing to the clerk and saying “I need one of these.”

When your choices look similar, knowing generally what you need isn't very helpful ("A bolt please…"). Being able to physically compare your options can save a lot of time.
(image: Lee Russell / Library of Congress)

Of course, not every broken thing can fit in your pocket. For those cases, a good substitute can be to snap a few pictures, or to write down the part and model numbers. However, these identifiers can be discontinued or change; also, this strategy assumes someone can translate this information into what you need. There might be multiple companies, apart from the original manufacturer, making "compatible" replacement parts. Hopefully, there will be a mapping between these worlds (OEM part #XXX ⇔ aftermarket part #YYY). However, there's nothing quite like the ability to put broken and new, side-by-side, spotting any differences between the two *before* you make the long drive home.

Bonus round: have you ever gone to the store and forgotten the *one thing* that you went there for? I have done this (all those free samples in the deli department can be very distracting…mmm, cheese). Let's just say that it's harder to forget your trip's purpose with a broken part weighing down your pocket!

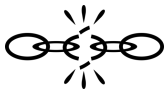
References:

- Header image: Maxham, Jason, photographer. *A lone screw*. December 9, 2016.

What We Bring With Us: "I Want One Of These" was originally published December 11, 2016.

Notes:

What Else Could I Be Doing?



> I am enough of the artist to draw freely upon my imagination. Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.
>
> **Albert Einstein**

Troubleshooting seems to involve a lot of **doing**: preparing your workspace, taking things apart, looking for malfunctioning components, ordering parts, the turning of wrenches, etc. You take positive action while fixing something: you choose a plan, execute it, and then assess the results. Following a particular path in this way, from beginning to end, may make it seem like repair is primarily a linear, physical process. However, the organizing concept for a troubleshooting project exists solely in your mind: by selecting a mental model to guide your repairs, you've decided how a machine *should work* and therefore the standard by which you'll be judging the outcome.

But before you've chosen a conceptual guide for your actions, before a single screw is loosened, you've made a much more important judgement. By selecting a specific repair path, you're implicitly saying that it's the best among *all your available options*. For example, you might perceive those lesser alternatives to be: replacing the failed machine with a new one, quitting your job (if you're fixing something for work), checking your email, watching YouTube, going to lunch, or staring out the window. Whatever other pathways you've envisioned, attempting a particular repair is

asserting, “I think this is better than A, B, or C.”

We often make choices because of momentum, or a sense of obligation, or because we don’t know any better. We are also limited by what we think is plausible: if you can’t conceive of something as a possibility, you’re unlikely to pursue it. This is why repair is actually a problem for the imagination. When troubleshooting, I always try to keep the end goal in mind. Counterintuitively, being true to the underlying purpose of *why* you’re fixing something is a constraint that will free your creativity. You’ll begin to see all kinds of possibilities, some of which won’t involve fixing the broken thing at all! Early in the troubleshooting process, the leverage from a clever redirection can be **astounding**—a good idea can save countless hours of work.

Your ability to optimize reality at the speed of thought is a precious gift. Cultivating it requires a balance of internal quiescence and the clever reuse of external stimuli. Your mind must be still enough so that your flashes of brilliance aren’t drowned out by distracting chatter. Engagement with the world provides the building blocks for these inspirations: be an artist that mixes lessons from your own doings, good ideas from other people, books you’re reading, and careful observations of the world. When a brilliant rearrangement short circuits a bad plan, instantly cutting an easy path to victory, there will be plenty of time for a nap.

> All action involves the employment of scarce means to attain the most valued ends. Man has the choice of using the scarce means for various alternative ends, and the ends that he chooses are the ones he values most highly. The less urgent wants are those that remain unsatisfied.
>
> **Murray Rothbard** [1]

References:

- Header image: Locke, Edwin, photographer. *Truckman napping at Amity Hall, Pennsylvania*. Amity Hall Pennsylvania Perry County, 1937. Aug. Photograph. Retrieved from the Library of Congress, https://www.loc.gov/item/fsa1998023597/PP/.
- [1] Rothbard, Murray. *Man, Economy & State.*

What Else Could I Be Doing? was originally published May 3, 2017.

Notes:

Part 3: Virtues

> My mindset is: never be afraid to try and fix anything. Odds are, if it can be fixed, you can fix it!
>
> **Mike McCormick**

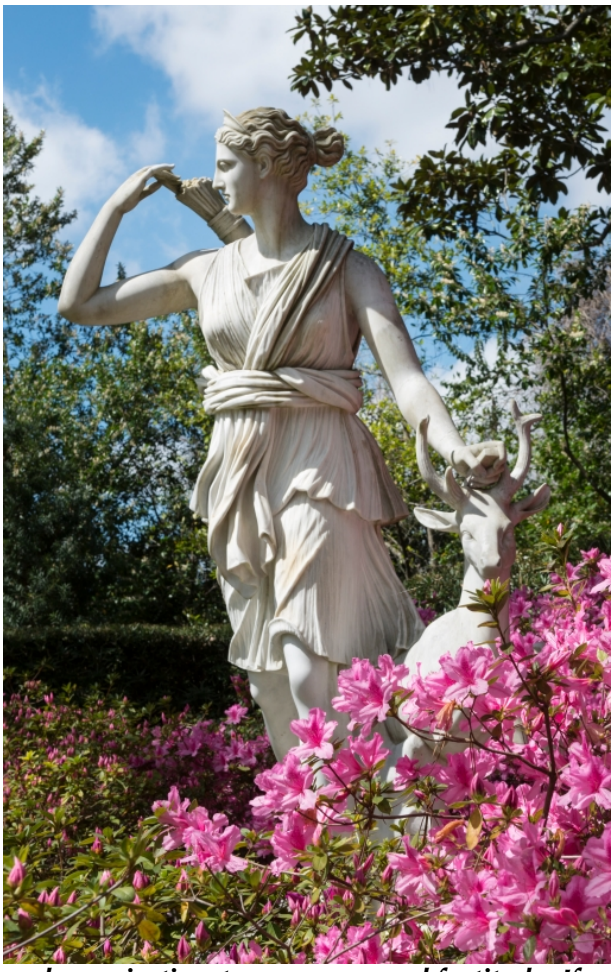
Why Talk About Virtue?

It may seem quaint to introduce a concept like "virtue," especially in a book about fixing things. It's probably even more confusing because troubleshooting is associated with machines, supposedly a world apart from humans. But, it's essential to describe the attitude and character of a great troubleshooter, should you want to become one.

The cultivation of these traits are an equal partner to learning the strategies. If that makes it sound like this is a self-improvement program, then so be it. If you want to become a better troubleshooter, you will need to improve yourself. Don't worry, I'll tell you how—it doesn't involve attending a clothing-optional "personal change workshop" with a charismatic guru named Ramu.

If you thought the mastery of troubleshooting was only about acquiring deep technical knowledge of a particular system, you're mistaken. Coupling your expertise with the proper mindset and behaviors will allow you to leverage the familiarity you do possess, but will also take you well beyond the limits of your knowledge. Virtuous skills, like the ability to listen closely to someone reporting a problem, will increase your chances of fixing the failure even if you don't know much about the particulars of a broken system. That's because they:

1. Direct your focus externally to the situation and facts at hand and put you closer to the actual problem.
2. Draw upon internal resources that open up a new world of possibilities for finding a solution. This is useful for problems that have never been encountered before (i.e., there's no book you can look in that will have the answer).



The Classical Virtues were: prudence, justice, temperance, and fortitude. If you want extra credit, you can practice those too.

(image: Carol M. Highsmith / Library of Congress)

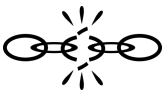
Some of the virtues may appear contradictory, like being creative versus being organized. If you practiced one to the exclusion of all others, you would be correct. Most virtues are like this: they are best exercised in moderation. Being hardworking is typically a good thing, until you work so hard that your spouse leaves and you have a heart attack because you've neglected your health. Likewise, being fun-loving is an attractive quality, but pursued single-mindedly might leave you bankrupt and living in a van…down by the river!

The goal of the virtues is to give you a choice in how you frame your actions while troubleshooting. Ideally, you'll bounce between the virtues as the context requires. While brainstorming solutions and alternative workarounds, switching into a creative and playful mode will be ideal. When it comes time to collect and analyze data, systematic and organized is where you'll want to be mentally. Achieve balance between the virtues and you will be a formidable troubleshooter.

References:

- Header image: Highsmith, C. M., photographer. (2010) *Column details located within the Pension Building, 401 F St., NW, Washington, D.C.* Washington D.C., United States, 2010. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2010641753/.

Skepticism



> It's good to have a certain amount of doubt about what is being reported. Of course, you can't be a jerk about it... Hear what they have to say and then investigate yourself.
>
> **Austin Quade**

When it comes to troubleshooting, skepticism is a virtue. That's because so many troubleshooting projects begin with someone else's account of the situation. Be nice about it, but never take a problem report as gospel truth. I can't tell you the number of times I've been sent down the rabbit hole by taking someone's description of a failure at face value. A person comes to you and says, "X is broke": the truck won't go, the Interwebs are down, the refrigerator is dead, etc. State something is broken and your mind will leap to consider ways it can be fixed. However, in my experience, people are much more likely to mean: "**I can't do Y**" (as in, "I can't haul the dirt…," or "I can't send this email…") when they assert that something is broken.

I stumbled on skepticism as a virtue of troubleshooting the 207 th time someone came to me with a problem, along with associated speculations on the cause, and sent me on a wild goose chase. Taking their report at face value led to a long exercise of debunking: not only of the *actual* problem, but ultimately the person's report itself. At the end, I thought "You lied to me!" Fool me once, shame on you, fool me 207 times…well, you won't fool me again!

There must be a lot of good troubleshooters here.
(image: LeeAnne Adams / CC BY 2.0)

You should expect most problem reporters to have the best of intentions, but not necessarily the knowledge or expertise to relate even the most basic facts of a breakdown (much less the underlying cause). After all, if they were that knowledgeable and wise, they might have just gone ahead and fixed it by themselves! You'll find all kinds of wild guesses about causes embedded in user reports. Parsing those messages by asking the right questions is an art in and of itself, as you've seen in "Skillful Questioning".

Until you master that, train yourself to say these words: **"show me."** Even better, **"What exactly are you unable to accomplish?**"

With regards to **"show me,"** there's nothing like actually observing a person attempting to do something they *think* they can't do. Watching is frequently a powerful antidote to any speculative words that may have been uttered during the reporting phase. Unfortunately, it's also an entree to the "intermittent" class of troubleshooting problems: if you've ever had the IT guy come over to your desk only to have the problem disappear, you understand that unexpected things can happen when you're being observed. Watching a person perform a task can also be a moment to suggest a workaround that may placate them until a real solution can be found. You may notice that their workflow or use of said machine is somehow suboptimal, in which case you might be actually improving their process:

You: "This copier is definitely broken, but did you know that you can use the one next to your office, rather than this one twelve floors down in the basement?"
Them: "You're amazing."

The question, **"What exactly are you unable to accomplish?"** brings the problem back to the land of facts (versus the assumptions that frequently accompany bold assertions like "X is broke!") and what is *actually* vexing the reporter of said problem. No one uses a machine for its own sake (exception: my motorcycle), so that's why "the printer is broke" can easily be turned into "I can't print my monthly accounting report from the computer in the conference room." Which, in turn, may lead to the discovery that the printer is fine and that an automated updater installed a buggy printer driver that is causing Microsoft Excel to crash when printing. That's way more specific and definitely not as dramatic as "the printer is broke!"—but it's a problem that can actually be solved or worked around.

As I introduced them, I said that the Virtues of the Troubleshooter are to be practiced in moderation. When it comes to being skeptical, I'm not telling you to disbelieve everything you hear. Likewise, this isn't license to mercilessly question every fact offered by someone reporting a problem. The point of skepticism is to keep open the possibility of a disconnect between words and reality, between someone's *description* of a problem and the problem itself. The deeper

implication of skepticism is to realize that you have a choice in what will become germane to your investigation and that it's your job as a troubleshooter to determine what is relevant and what is true.

References:

- Header image: *"Mechanic at work"*. Aarron Norcott, photographer. Retrieved from Unsplash, https://unsplash.com/photos/gD1HU_XU5aE.

Skepticism was originally published September 20, 2011.

Notes:

Listen Up

> I'm guilty of this as I like to chatter—I jump in and start talking... If you basically bite your tongue and shut up, most patients will give you all of the information that you need in a much more efficient amount of time, versus trying to interject and redirect them.
>
> **Ken Fechner**

With the virtue of listening, realizing that something so human is central to something *seemingly* so technical is a microcosm of my personal journey as a troubleshooter. That the cold, *hard* world of machine problems could be undone by a *soft* skill like listening is not a new idea:

> The soft overcomes the hard;
> the gentle overcomes the rigid.
> Everyone knows this is true,

> but few can put it into practice.
>
> **Tao Te Ching (verse 78)** [1]

Let listening be a proxy for using *all* of your senses to take in information about the situation at hand. Yes, even your sense of smell will come in handy while troubleshooting—you know this if you've ever smelled a burnt out power supply or gas leak. You need to be open to all of the sensory information coming your way when investigating a problem. I chose to group all this under "listening" because of the special importance of words: as noted previously, this is because so many troubleshooting exercises begin with someone else's account of the situation. As opposed to "hearing", which to me connotes the raw sensory experience of sound, "listening" implies that some thought is applied on top of what is coming in from your ears. Each of the senses can be thought of in a similar manner: the sensory data must be combined with focused effort to be useful.

Listen, even if they don't have much to say.
(image: Library of Congress)

You may think it contradictory to have both skepticism and listening as Virtues of the Troubleshooter. What use would a skeptical person have for listening? However, it's not a contradiction because any doubt you have must be anchored to reality. Feel free to be skeptical of someone's account, but make sure you're being skeptical of what the person **actually** is saying. Try to cultivate an "informed skepticism" to make sure you're resisting against something real. Put another way: it would be a failure to counter a reporter's fantasy of a problem by making one up yourself and attributing it to them! If you want to tilt at windmills alone, there's no surer way than to be a troubleshooter who has stopped listening. It may be funny when Cervantes wrote about it, but tragic when observed in real life.

When it comes to listening, there's another NLP concept that is extremely helpful: "uptime." Uptime is "tuning the senses to the outside world"[2] and it is extremely useful for certain phases of troubleshooting. When interviewing, it means being completely present in the interaction. When investigating on-site, it means being focused externally. Doing this requires energy (especially if you're introverted), but it's worth it. Being inside your head at these crucial moments, you will miss so many important details. I discovered the value of uptime by the astounding number of times the key to solving a particular problem was either staring me in the face or explicitly referenced by a person reporting a problem. And, conversely, the number of times I missed it by being distracted. Now, I pay attention with laser focus.

I interviewed a very experienced auto mechanic who has 30+ years in the troubleshooting trenches. He laid out two reasons why he is such a strong believer in listening: first, the person closest to a problem will always know better than

you what is "normal." Second, listening allows you to size up someone's personality. Dan told me the story of a car that, from his perspective, was perfectly fine. However, he listened to the client anyway:

> **Jason Maxham:** What weight do you put on a client's description of the problem?
>
> **Dan McCormick:** A *lot*. They know their car better than I do. If it's doing something different now than what it did a month ago or a year ago, something has changed. I might drive it and say, "I don't notice anything wrong." I've run into that situation: for example, this guy and I went for a test ride and I said "To me it feels fine…" and he said "No, something is different." So, we decided we were going to dig into it and I started tearing the car apart and I found two wires that were crossed. At that point, the guy had left and I called him up and said, "Would you mind coming back and going for a ride?" I switched the two connectors around and he said "That's it, that's the way it used to run!" In my mind, I didn't feel like there was anything wrong, but this guy really knew his vehicle.

Putting It All Together: Being Skeptical And Listening Carefully

Listening while guiding someone to give a useful account is a delicate balance of letting them speak, while also leading the person to tell you what you need. If you do all the talking you won't be getting much closer to a solution, but neither should you let someone go off on an endless tangent. This is another reason why you must be in uptime during the interview process: you must maintain situational awareness to break in when the interviewee goes off-road. If you don't maintain control, prepare to hear about life, the universe, and everything else. I think this is because people think: "Finally, someone is listening to my problems…so I'll just keep going!" It's funny how inquiring about a broken printer can quickly morph into a free-range discussion about management's lack of vision, the absence of meaningful choices in the break-room vending machines, and the "girl that got away." Some people don't get a chance to vent very often and will relish your bent ear, while others simply love spinning a good yarn. You must be "up above it" (guiding the conversation by cutting short unnecessary threads) and "down in it" (actively listening and soaking up the critical details being thrown your way).

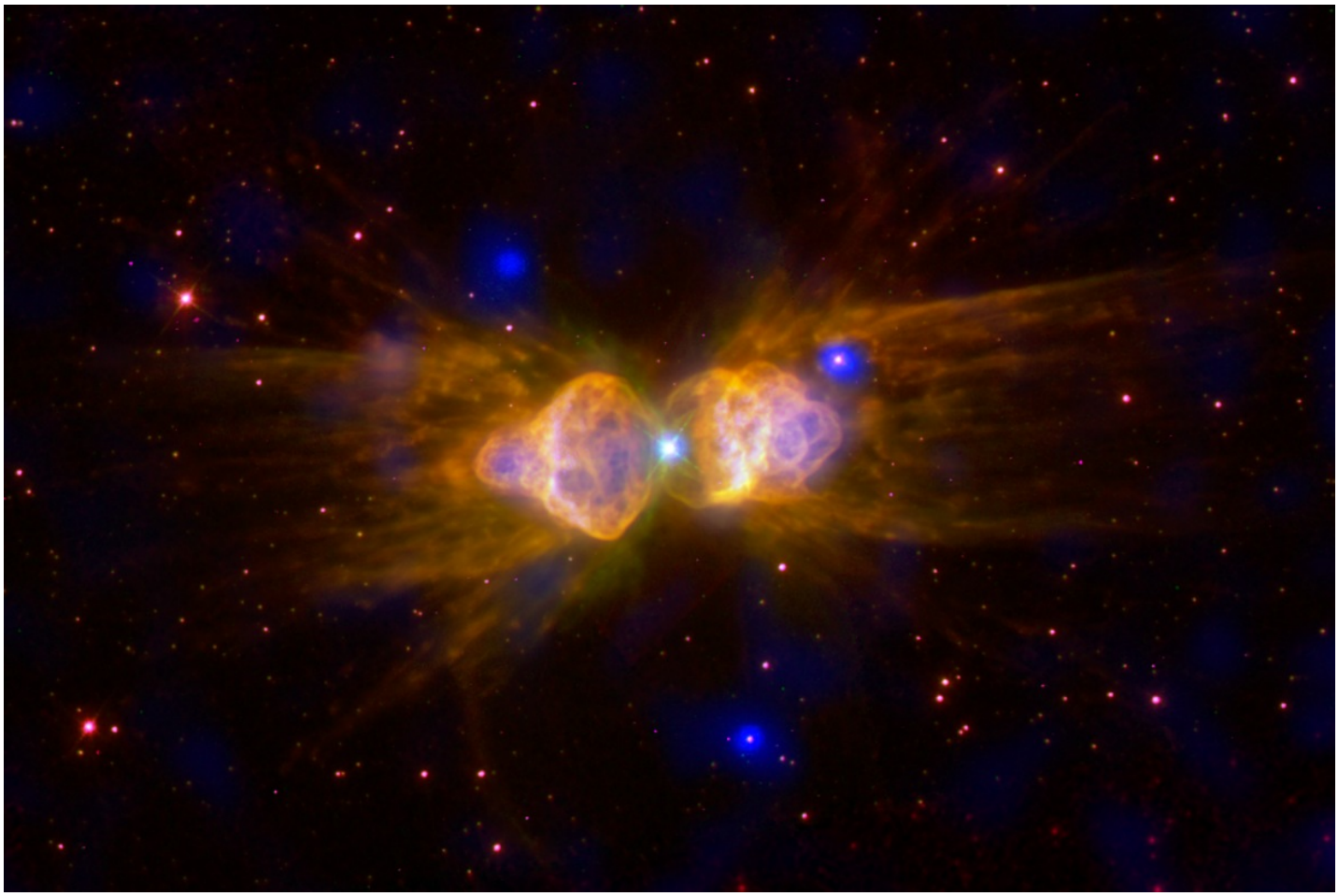
References:

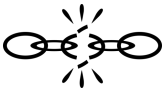
- Header image: Bain News Service, P. *Listening to Records.* ca. 1920-1925. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2014715085/.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins: 1999), verse 78.
- [2] Joseph O'Connor and John Seymour, *Introducing NLP* (London: Thorsons/HarperCollins, 1990), pg. 111.

Listen Up was originally published November 2, 2011.

Notes:

Curiosity





> You're going out into the unknown. If it was known, it wouldn't be a problem, it wouldn't need troubleshooting.
>
> **Karl Kuehn**

Troubleshooting favors the curious. A burning desire to learn and ask questions is a natural complement to fixing things. Don't mistake the pressure to fix a problem for curiosity: this is separate from job responsibilities and the frustrated energy created when a breakdown blocks your way. Foster your sense of curiosity as a virtue in its own right, especially apart from your work. Asking "Why?" and following the answer wherever it leads, will spill over to your troubleshooting. Probing deeper won't seem like a chore, because it'll be in your nature to want to go beyond the surface level. When you do, you'll find that the world is an interesting place when you peel back its layers. My philosophy of "Cleaning Up" stems from this inquisitive mindset: for the curious, the problem and immediate solution are just the beginning.

Curiosity Made The Cat Into—A Great Troubleshooter!

Were you that kid that liked to take things apart just to see how they worked? If so, you might not have always been able to put it back together again. Curiosity can get you in trouble—and that can be a good thing—assuming you live to tell the tale! I remember messing around with the family computer when I was growing up. What does this file do? What would happen if I edited it or moved it? The family computer was the most expensive thing we owned (besides our house and cars). It was spoken about in hushed voices and wrecking it would be a Bad Thing. But wreck it I did and most of the time I was able to put it back together again.

Curiosity will lead you to situations that are over your head. I mucked around, made mistakes, and had to recover. This made me stronger. I'm not saying you have to flirt with disaster to be a competent troubleshooter. While the super inquisitive make great fixers, I just want your average troubleshooter to be a little more curious.

Let your curiosity take you beyond the horizon, to that place just out of view…
(image: NASA Goddard Space Flight Center / CC BY 2.0)

Exercise That Muscle

Don't consider yourself naturally curious? That's okay, I think it's something that can be cultivated. Try this fun exercise:

The first week: every day, choose something in your non-work life and decide to learn just a little more about it. Take a business, organization, or government entity that you know or interact with and look them up online (this can be fascinating, especially if they are prone to saucy scandals!). You probably come into contact with lots of people who do things for you: butchers, taxi drivers, auto mechanics, doctors, store clerks, baristas, etc. Why not ask them how business is going or about an aspect of their job? There are lots of things we rely upon in our modern lives, but it's amazing how we take them for granted. Do you know how electricity works or how an internal combustion engine operates? Who invented the Internet? What's in a Twinkie? If you're truly stuck and need inspiration for a topic, dive into a random Wikipedia article.

The second week: every day, choose something from your work life and use that as a basis to dig a little deeper. The tools you use, processes you manage, the people you work with, your organization, your industry: these are all fertile ground for asking questions. Who makes your favorite tools and what else do they offer that might be useful? Who invented the key technologies in your industry? What's the background of your co-workers? Who stocks the vending machines in the cafeteria? What does your company's latest annual report and balance sheet say about the future of your job?

I guarantee that, at least once during these 2 weeks of curiosity, your mind will be expanded with an amazing fact, a fascinating but hidden connection, or a supremely useful nugget o' knowledge that will make your life better.

After that: Keep a running list of things that you want to learn more about. I keep a list like this on my phone and I add to it whenever I come across something that deserves follow-up. The upside? I'm never bored as there's always something interesting on the list to learn more about.

Fake It Till You Make It (But Don't Go Too Far)

If you don't consider yourself a curious person, just try pretending you are one when troubleshooting. Think: what would a curious person do. I can just see the "WWCPD?" bumper stickers, coffee mugs, and t-shirts being printed in mass quantities—I'll be glad to take the credit for that when it happens. Especially look for moments where inquiry has stopped and a "first-level" response or solution is being accepted. The curious person isn't satisfied with the surface, they're eager to plumb the depths.

When can curiosity get in the way? I mentioned that the Virtues of the Troubleshooter need to be balanced amongst each other and practiced in moderation. As surprising as it might seem, curiosity should probably take a back seat to the other virtues when actually troubleshooting. Curiosity really shines as a virtue before and after fixing something. Before, it's great for accumulating knowledge about the world in preparation for troubleshooting (especially if you're curious about your tools, systems, organization, and industry). After everything's back up and running, curiosity is great for pushing to fully understand what happened and make long-term improvements (i.e., in the "Cleaning Up" phase).

While troubleshooting, however, curiosity must be tempered by the other virtues. The knowledge gained from being curious has a cost in terms of time and resources. As noted before, sometimes it simply doesn't pay to know why a failure occurred. The impact of downtime should lead the troubleshooting process: if lives or livelihoods are on the line, the focus should be on getting back to normal. Save your wide-eyed curiosity for when the pressure is off. Lastly, the troubleshooter should always keep an eye on safety. Curiosity may have killed the cat, but don't let it harm you!

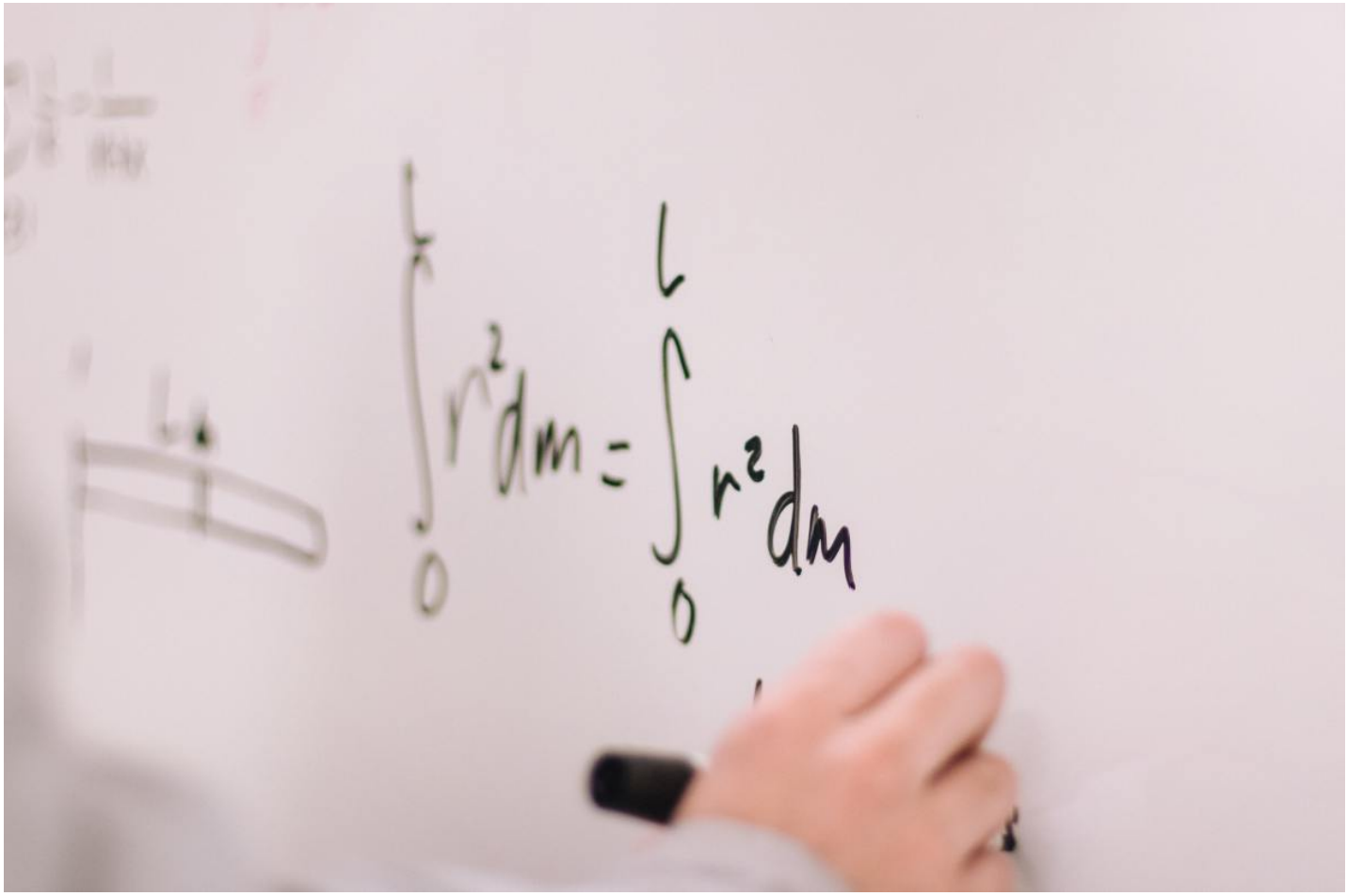
References:

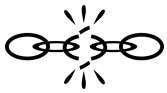
- Header image: *"Mz 3, BD+30-3639, Hen 3-1475, and NGC: 7027 – Planetary Nebulas – Fast Winds from Dying Stars"*. NASA's Chandra X-ray Observatory. Smithsonian Institution.

Curiosity was originally published November 15, 2012.

Notes:

Out Of Your Vulcan Mind





> It's important for you as a troubleshooter to remain calm. Because if you start freaking out, it will make them freak out. And, a lot of times, they are already freaking out.
>
> **Austin Quade**

Up to this point, the Virtues have focused on social skills like listening and right-brained strengths like creativity. Now it's time to give a nod to what the left-brain brings to the table. I want the information in *The Art Of Troubleshooting* to be extremely practical, so I've highlighted some very simple troubleshooting methods like the reboot and seeing if it's plugged in. However, I don't want you to get the false impression that the answers are always that easy – some issues will take everything you've got (and then some) to figure out.

Clearly, you're interested in reaching a level of mastery that will allow you to tackle the hardest troubleshooting problems. That will require a higher level of commitment, along with four interconnected and Spock-like virtues: being organized, systematic, detail-oriented, and logical.

Put your inner Spock on the problem.
(image: Wikimedia Commons)

Organized

Difficult problems will require you to create your own structure around the troubleshooting process. “Winging it” won’t be an option for a months-long investigation that involves not just finding and fixing a problem, but also preparing a detailed report to be read by your peers.

Organization might be required in many dimensions, depending on the size of the problem you’re trying to solve. You might have to manage a team of troubleshooters (people), conduct interviews (gather information), work with outside service technicians (manage company resources), update the wider organization on your progress (communicate), alongside the core tasks of finding and fixing the issue. Each of these areas benefits from careful planning and coordination. Being organized also includes marshaling the information you will analyze and disseminate: from taking notes to collecting data (logging, graphing, etc.). Ably executing these varied roles will require good organizational skills.

Systematic

Troubleshooting can be a real grind: there will be times when you'll need to tediously examine a long list of possibilities, one by boring one. The Strategies are designed to reduce the time spent mucking around, but the shortcuts they provide won't always bear fruit. When that happens, you'll need to step into a methodical mindset to go the full distance.

Especially as the complexity of a problem increases, the number of variables will grow beyond your mind's ability to track them. Research suggests that your mind's working memory is only between 4-7 items[1]. Troubleshooting simple machines may not tax this limit. For everything else, you'll need to get out a notebook and pencil, or fire up a spreadsheet program.

Systematic also means being thorough, following through all the way on a given troubleshooting path to obtain a useful end result. With some strategies, there will be a large number of possibilities to test, along with meticulously resetting your test environment every time to get valid results. Getting to the end of a process like this can be tedious, but some data won't be useful for decision-making until you've mapped the entire set of possibilities.

Detail-oriented

When troubleshooting, the devil is in the details. Or perhaps leprechauns live in there. Or pixies. Or that gremlin William Shatner saw on the wing in that *Twilight Zone* episode. Whatever's plaguing your machines, you'll need to get involved with the nitty-gritty. Modern machines can have hundreds or thousands of parts. The strategies presented in this work try to increase your odds of finding the errant element without having to resort to a piece-by-piece inspection. However, the ability to track a project on a micro-level will be useful for those complex problems that will sprawl out and stay a long while, like a bad houseguest.

Logical

> The first principle is that you must not fool yourself—and you are the easiest person to fool.
>
> **Richard Feynman** [2]

Our pointy-eared role model knew that no amount of feeling would make a situation other than it is. Always remember the difference between *good sounding reasons* and *sound good reasoning*! Spock had an unwavering commitment to reality, there was no fooling him: he would follow the facts wherever they led. Breakdowns can have an emotional component, especially when there are angry clients or inconvenienced co-workers knocking at your door. It might be tempting to respond in kind with passionate fire, but it's better to channel Spock instead. Turn to detached reason, focus on the facts, and use the tool best suited for the job: logic.

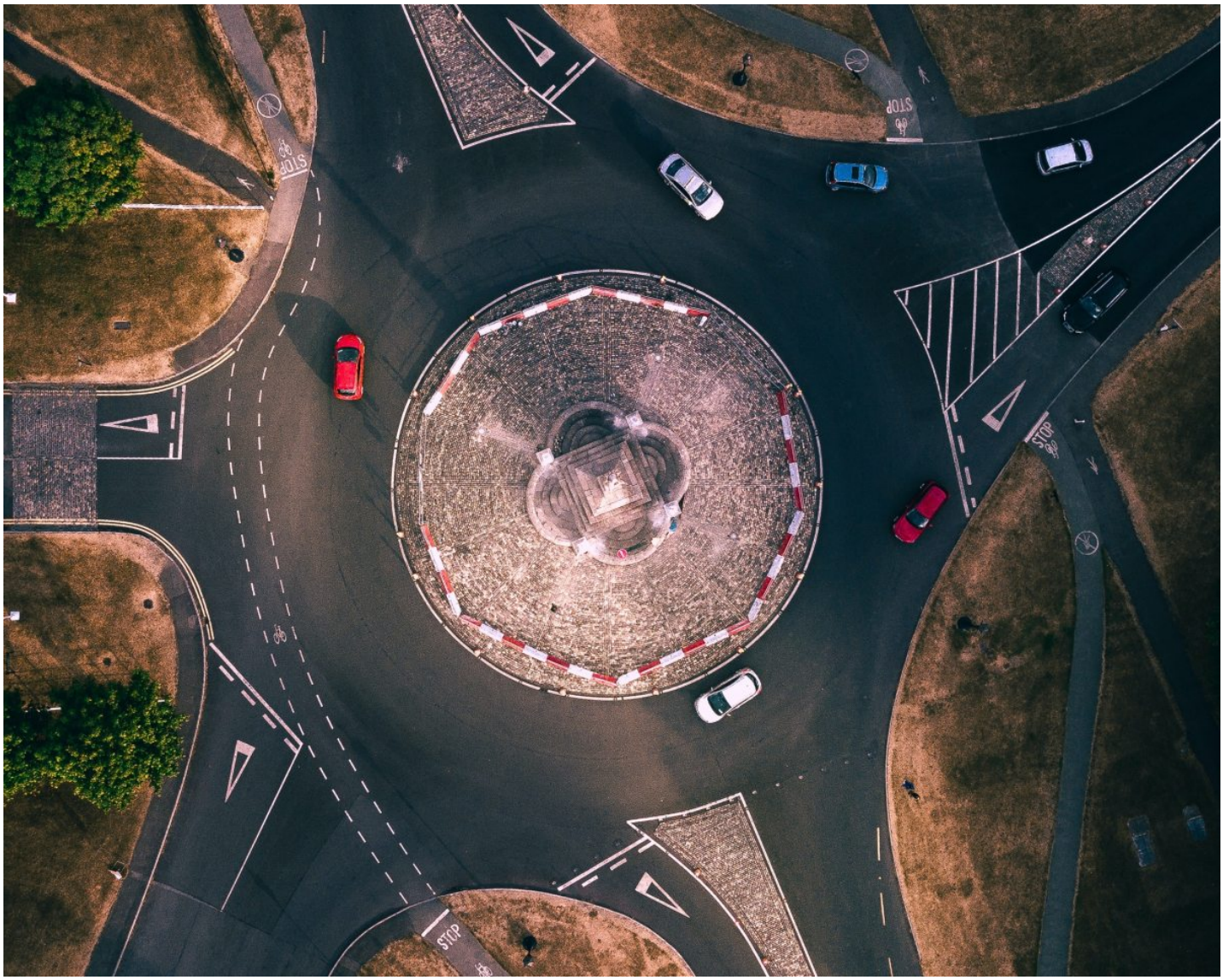
References:

- Header image: Jeswin Thomas, photographer. Retrieved from Unsplash, https://unsplash.com/photos/hecib2an4T4.
- [1] Nelson Cowan, "The magical number 4 in short-term memory: A reconsideration of mental storage capacity," *Behavioral and Brain Sciences*, February, 2001.
- [2] Richard P. Feynman, *Surely You're Joking, Mr. Feynman!* (New York: W.W. Norton & Company, 1997), pg. 343.

Out Of Your Vulcan Mind was originally published March 7, 2013.

Notes:

Creativity



> You have to have an improvisational mindset.
>
> **Alex Chaffee**

Troubleshooting is about making something work again; most of the time, no one will care *how* you make it work again! People use machines to accomplish their goals, not for their own sake. They are a *means* to an end: if the purpose a machine served can be met some other way, no one will mourn a breakdown.

Recognizing this critical distinction between *machines* and the *goals* they advance leads to the following insight: because the outcome is the most important thing, any reasonable route that gets you there should be considered. That includes totally bypassing a broken system! When you're freed to take any number of paths to the promised land, *creativity* becomes a potent virtue of the troubleshooter.

Using your creativity to make a situation better is the difference between someone who merely "fixes things" and a *bona fide* Troubleshooter. Repair, swap, replace, reroute, cannibalize, triage, find a workaround, or do nothing:

creative problem-solvers open themselves to the whole universe of possibilities.

Choosing Between Mrs. Right And Mrs. Right Now

When a machine breaks, the need it served will still persist–that's the impetus behind every repair. However, the existence of this unfulfilled need is why you must troubleshoot the *whole situation*. In addition to searching for a fix, you should also look for ways to provide temporary relief to those affected by a machine failure. You may need to jury-rig something to get that last load delivered or that important email sent, until a long-term solution can be found.

Balancing the needs of the present with the future is its own art because providing temporary relief can conflict with efforts to find a long-term solution. There's usually not just a single fix to consider, but rather a whole spectrum ranging from the temporary to the optimal (and often expensive). The finite resources needed to make a repair (people, tools, spare parts, etc.) may only allow you to pursue one avenue at a time. If you intend to quickly put a machine back into service with a temporary fix, clearly it won't be available for a more extensive repair that would require downtime. Another danger is that temporary hacks can become *de facto*, permanent fixes if you're not vigilant and push to make long-term repairs a priority.

You could write an entire book about these competing considerations—in fact, *this* book—so we won't go over what has already been covered. The takeaway is that creativity is the supreme virtue when managing situations where a malfunction has left someone stranded. If you unleash that creative genius inside, you can often find a way to provide both temporary relief and long-term reliability.

So many paths to take…
(image: Susan Yin / Unsplash)

The Right Question

So, you need to be creative…but is it possible to learn to be more creative? I certainly can't teach you in the short space of this piece, and I'm not even sure where I would begin. Many engineer-types draw creative inspiration from other aspects of their lives, from artistic hobbies like music, photography, painting, dancing, improv, theater, etc. These activities can cross-pollinate to your problem-solving, so consider those avenues. Others find that it's hard to access their creative side because of a cluttered or "noisy" mind. In order to give yourself the space to actually be creative, you may need to first gain control of your runaway thoughts. On that front, check out the new-fangled inventions of exercise and meditation.

The above are all worthy pursuits that can enrich your life and boost your creativity. However, I don't want you to think you need to embark on some radical self-improvement program just to inject a little more imagination into your

problem-solving. It's likely you already have plenty of clever ideas and just need to remind yourself of the distinction between machines and the purposes they fulfill. Lastly, asking this question before you start troubleshooting will get your creative juices flowing:

> **"What exactly does this machine *do* for us?"**

This query will focus your mind on the end goal—the need a machine was serving.

Also, be sure to ask those affected by a breakdown how they would manage if a machine took a long time to be repaired: the "victims" will often have the best ideas for workarounds!

"Do you mind if I try out some new techniques while we're falling to our deaths and you're strapped to my belly?"
(image: Morgan Sherwood / CC BY 2.0)

Exercise That Muscle

In those high-pressure situations where creativity would be useful, you might not be able to generate good ideas because of the stress involved. Or, maybe the troubleshooting process in your field is rigidly proscribed because of legal, safety, or quality reasons. Some professions (police, firefighters, pilots, doctors, etc.) have well-defined procedures and require extensive training because stress impairs cognitive functions. I'm not sure I want my brain surgeon or skydiving instructor to "get creative" without first testing their whacky new theories in a safe and controlled environment (or on themselves). An idea might look good at first glance, but after critical examination it may fall apart. You'd like to know this *before* it really counts.

High-stakes troubleshooting will still benefit from your creative input, but in a more thoughtful, *after-the-fact* type of manner. Root cause analysis (RCA), when done post-crisis, is a great way to thoroughly review an incident and generate ideas for future improvements. At a conference table, far removed in space and time from the chaos of a meltdown, you can feel free to be as creative as you want. Plus, your imaginative new ideas will benefit from the peer review of such a setting. There's no substitute for taking your time: some refinements will need the rigor of the scientific method (making a hypothesis, designing an experiment, collecting data, etc.) to know whether they are better than your current practices. Finally, the learning that happens as a result of your RCA process will lead to the kind of deep proficiency that makes spontaneous creativity possible. If you've ever seen a true expert come up with amazing things on-the-spot, it's usually a result of all the experience and learning that came before.

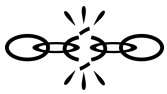
References:

- Header image: Lucas Miguel, photographer. Retrieved from Unsplash, https://unsplash.com/photos/SdMuKn6KKaA.

Creativity was originally published April 12, 2013.

Notes:

Be Present



> You need to be open and utilize all the senses you have. Not only all five senses, but also your sixth sense.
>
> **Rich Kral**

When I was learning how to ride a motorcycle, I decided to take an introductory "safety and skills" class. I'd heard about how dangerous riding was, so it seemed like a good idea. Point of disclosure: taking the class didn't prevent me from crashing my first motorcycle, leaving a big scar on my knee. You were right, Mom.

When you're first learning how to ride a motorcycle, turning can be quite mysterious. First, there's the concept of countersteering, which is very much counter-intuitive. That's right, you push the handlebars in the *opposite* direction of the turn. On top of that mind-blowing detail, coordinating the turn with my head, arms, and legs proved to be quite a challenge for me. Feeling out of control, I'd move my head (and field of vision) all over the place, including long glances at the ground in front of me (so I could see where I was going to crash!). I was always veering off past the orange cones set up to mark the course boundary, or coming dangerously close to my fellow students. Seeing my struggles, one of the instructors pulled me aside and said "Forget everything we've taught you and just do one thing: look *through* the turn at where you want to go."

I tried out his advice on the next turn and—*SHAZAM*! That one little tip cleaned everything up. The motorcycle magically went where it should; my body and head aligned while leaning the bike just the perfect amount to complete

the turn. I did it a few more times and realized that, by simply looking in the right place, I didn't really have to think about turning at all. What was so complicated before was now reduced to a single thing. Looking through to the end of the turn automatically coordinated all those other actions I was struggling with—and I didn't even have to be consciously aware of them.

You could just do it, or…create a situation that requires a lot of hand washing. It's your call.
(image: s.h.u.t.t.e.r.b.u.g / CC BY 2.0)

Since then, I've been on the lookout for other "unifying actions:": solitary things you can do that automatically bring with them a whole host of other benefits. A "buy one get one free" sale for your efforts, if you will. When it comes to troubleshooting, I can think of no better example of a "unifying action" than being present while you solve problems. If there's something that will kill your troubleshooting abilities, it's being in your head at those critical moments where you should be externally focused instead. That's because the initial phases of troubleshooting are all about collecting information about the situation at hand.

You can start whacking away, trying random strategies, but you'll be so much more effective if you accept what is freely given and then take the obvious next step. What is freely given? All those relevant facts that are staring you in the face if you were just in tune with the situation. The key details, which will help you select the correct strategy, are out there in the world around you, not floating around inside your head. The other virtues may come into play later on, but being present reigns supreme at the *beginning* of any troubleshooting exercise. There's just too much you can miss by mentally being elsewhere. Let me give you some examples of how being present brings with it a whole host of other benefits, for free. Consider:

- Troubleshooting environments are often dangerous, with exposed electrical circuits, ladders, power tools, heavy machinery, parts strewn across the floor, sharp edges of broken components, etc. In these kinds of places, you *must* be externally focused and aware in order to be safe.
- Listening to other people describe their problems requires your attention: important details might need to be teased out with smart follow-up questions. On the opposite end, your critical filter must also be engaged to prevent being led down a rabbit hole. Both of these considerations require paying close attention to what you are receiving from the people around you.
- The environment will give you clues as to what is wrong: funny noises, odd smells, error messages, indicator lights, the state of nearby machines, etc. Tuning into this contextual information will often make the difference between a timely repair and a painful slog.

I could go on and on, but instead I'll simply say that **every strategy in *The Art Of Troubleshooting* will benefit from**

the virtue of being present. Conversely, everything you do will be that much harder if you're off on another planet.

Later on, there will be an appropriate time to go inside your head and build castles in the sky. Save that for when you're interpreting the information you've collected and during the final phase ("Cleaning Up"), where you learn from a breakdown. Those processes benefit from being inwardly focused and decoupling from the outside world—your ability to concentrate and think will be improved from the isolation.

But, when I'm actually troubleshooting, I'm *with* you. I'm right here.

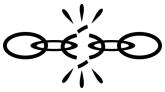
References:

- Header image: Hine, L. W., photographer. *"Alex", a fourteen-year old working boy in St. Etienne, was found intently studying the playground exhibit at the Children's Welfare Exhibit at St. Etienne. Exposition of the ARC. July 1918.* Saint-Etienne, France, July 16, 1918. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017681813/.

Be Present was originally published April 17, 2013.

Notes:

Setting Boundaries



> As the years went by, the cars got more expensive and the people got crabbier. There was no patience and they would get on your back. I'd say, "Wait a minute. I didn't make this car. I didn't sell you this car. I'm just trying to help you."
>
> **Gerald Quade**

I vividly remember one of the first times I stepped back from the abyss. While I was looking for a full-time job after I graduated from college, I did some IT consulting for local businesses in my hometown. One of my clients was a successful family business that had outsourced most of their IT work to a consulting firm. I filled the gaps between visits from their main consultants with anything that needed to be repaired, installed, or upgraded: PCs, printers, Internet access, software, etc. These were basic tasks I felt very comfortable performing, and low-risk.

One day, I was asked to install a piece of software that ran on the office's shared application server. There was a new text editor program they wanted to run on all the terminals attached to their networked computer system. Would I take a look at it? Okay…sure. I was brought to the server closet and saw the two computers at the heart of their operation. Looking back across the bustling office to the lobby, I saw all of the employees and customers that depended on this system working correctly. A lightning bolt of warning struck me: **"Don't mess with this, you're out of your league!"** Up to this point, I had never worked on a networked application server, much less in such a high-risk situation. Let's face it, I was as green as could be. I don't know what triggered that candid moment of self-awareness. Normally, I'm a "get in there, tear it up, and see what happens" kinda guy. Maybe it was the worst-case scenario flashing before my eyes: a vision of me crashing the system and the business closing indefinitely until a high-priced guru could be flown in to mend the situation. I politely declined the work, citing my inexperience.

Belle Of The Ball

As your troubleshooting skills grow, you may find yourself becoming quite popular. Don't get too excited, we're not talking "Homecoming Queen" popular, but rather the "Can you help me with this problem?" kind. Sorry, but you can still buy the tiara if you want. While it's great to be wanted, you'll have to set boundaries to protect yourself. It's not only about guarding your time, you also need to shield those whom you intend to help. When I declined that opportunity to work on the application server, it protected me *and* the people I was trying to help. A blunder would have ruined my reputation (and weekend), along with their ability to conduct business. Biting off more than you can chew is annoying to you *and* the person who asked for your assistance. The guiding principle is "do no harm." If you're there to help, make sure that actually happens.

Flattery Gets You…In Over Your Head

"You're so smart, I just know you can fix this!" Everyone likes being complimented, but it's also a recipe for getting in over your head. I've seen flattery feedback loops end in big failures. The problem is that if you assert you can fix something, most people will believe you. It takes two to tango: a person with a problem will typically give you all the rope needed to hang yourself. Once you take control of the situation and start mucking around, it's rare for a customer to say, "Are you sure you should be doing that?" You might be the first, last, and only line of defense against hubris. When you pack your gear, throw some restraint into your toolbox.

If you're not careful, it's easy to get sucked in…
(image: Jonathan Bean / Unsplash)

Leave Yourself An Out

Whenever I go hiking, I like to remind myself that every step I take away from the trailhead will require a step back when I return. In a sense, every step outward is a *commitment*, a promise to take a step in the opposite direction. You can't hike for 4 hours one way and then be angry when it takes 4 hours to return! Hiking a trail is a good example of a debt that slowly builds over time, one that must eventually be repaid.

Likewise, large repair projects have implicit obligations that can be accumulated over time—or even in a single moment! Taking a particular repair path requires you to make a commitment, but often that commitment is made without careful consideration of the long-term consequences. Certain actions are difficult to reverse: many machines are easy to take apart but *very* difficult to put back together (even if you've taken notes or pictures). Removing a suspicious part may destroy it; long, complicated procedures that don't bear fruit may require an equal amount of time to revert. Back to our virtue of maintaining boundaries: you must be aware of what's happening and not cross over these lines without a deliberate plan.

So, before you reach the "point of no return," stop and ask some important questions:

- Will repairing this system require downtime? If so, who will be affected?
- How long will this repair take? What if it's not completed on time?
- What are the risks of attempting this repair? Can it be reversed and what are the steps to get back to where I started?
- What if I am unable to complete the repair?

Answering these questions is a great invitation to forming a **contingency plan**, an "out." The "when" of a repair is often negotiable: even if a system is running slowly or intermittently, it may still be doing useful work, allowing a repair to be postponed until a more convenient time. Of course, if the broken system is already down, there's no issue about interrupting work—it's already happening!

Even when your hand is forced, there are still important decisions to be made because the question of risk is present in any major repair. I always favor swapping if there is a good possibility of damaging a machine while troubleshooting: it's always better to attempt those kinds of fixes in a low-pressure environment.

Great Expectations

> Blessed is he who expects nothing, for he shall never be disappointed.
>
> **Alexander Pope**

Perhaps the most important boundary you can set is someone's expectations. I've learned the hard way the value of being modest when promises to help are being made. Besides, it's always best to let your actions speak for themselves. If you find yourself prone to verbal boosterism, at least tap the brakes when it comes to two crucial subjects: **time and money**. These are always the most requested pieces of information about a repair, but speaking too optimistically about either risks major-league disappointment. It's much better to say "I don't know, I'll have to take a look." Which happens to be the truth: how can you really know how long a repair is going to take and how much it's going to cost without a thorough assessment of the situation? Underestimating the time of a repair has led to some of my worst, "in over my head" incidents. If you're hacking away on a Sunday night, remember that Monday morning will eventually come.

While I've solved a lot of tough problems, I feel that some of my most impressive fixes (at least from the perspective of those I was helping) were simply the result of initially being circumspect about what I could accomplish. There's nothing like starting off with, "Geez…I'm going to have to take a look first…I'm not sure if I can help you." And then 5 minutes later finding the solution. Standing ovation.

The emotional arc of this kind of repair drama is very satisfying for those you are helping: doubt → optimism → elation. Compare this with the trajectory of expectations violated: certainty → doubt → anger. The funny thing is that two troubleshooting exercises can be identical except for the expectations created: one results in a happy customer, the other in an angry one.

Granted, you can go overboard with sandbagging people's expectations. It'll take some trial and error to find the right balance between being confident and being guarded, the right tone appropriate for the people you're trying to help. If you're too much like Eeyore, the undisputed king of low expectations, you'll never be asked to do anything! People will rightly expect that you project a baseline level of confidence that communicates you are competent. While it takes effort to make this calibration, the overall goal of deftly managing people's assumptions is worthy of your time: it's always better to exceed expectations than to be perceived as coming up short. It's just human nature.

References:

- Header image: *"Gaithersburg tennis court"*. Chris Chondrogiannis, photographer. Retrieved from Unsplash, https://unsplash.com/photos/oN_rL__KiiU.

Setting Boundaries was originally published May 15, 2013.

Notes:

Part 4: Cleaning Up

> The easiest problem to troubleshoot is the one that never happened in the first place.
>
> **Alex Chaffee**

Why You Should Go Beyond Troubleshooting And "Clean Up"

Troubleshooting is a reactive response to a failure. The cause and solution may be unknown, but the strategies are designed to provide the quickest path to a resolution. If all you did was use the strategies and thereby gain reactionary skills, it would be an improvement to your life. However, as you grow, your gaze will turn to the proactive side of troubleshooting. That's the focus of the "cleaning up" material presented here.

When the job is done, it's time to wash up.
(image: Jack Delano / Library of Congress)

My concept of troubleshooting continues after the crisis has past and whatever was broken is fixed. The good dentist doesn't want to find cavities in his patients' mouths; the good firefighter would prefer to never see a house engulfed in flames. Likewise, the good troubleshooter would rather not have a system failure lead to a crisis (if we're talking about what he or she would rather do, it would likely involve a beachside view and a fruity drink with a little umbrella). Yes, the virtues and strategies can help you be the hero when needed. But better yet is to never need to be a hero at all.

To achieve this ideal, you will be vigilant about learning from failures (both human and machine) and feeding that information back into your processes and procedures. You will use the moral authority from the aftermath of a crisis to make needed changes. You will collect data so that you understand what is happening at every level of your systems and infrastructure. You will probe deeply into breakdowns to understand the root cause. You will be anticipating failures, freeing spare resources and creating procedures to focus on being prepared for meltdowns. In short, you will transcend mere troubleshooting.

The Master Troubleshooter would rather be here. Heroism is for the unprepared.
(image: Carol M. Highsmith / Library of Congress)

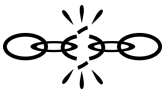
The argument for the reactive side of troubleshooting is self-evident: something is broke and needs to be fixed. The proactive side requires greater advocacy because its benefits aren't as easily seen and slower to appear. These rewards require delayed gratification: making investments that may be long to bear fruit, taking action based on incomplete information, putting things in order before they exist, not taking action, considering alternatives. Is all this starting to sound a little philosophical?

Know the reactive, but keep to the proactive.

References:

- Header image: Detroit Publishing Co, P. *Washing Down Decks.* [Between 1900 and 1905] [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016811292/.

Is This Normal? An Ode To Data Collection



> For optimization, you need to gather data first.
>
> **Alex Chaffee**

You Don't Need Specific Knowledge

Over time, you begin to start understanding what is needed for efficient troubleshooting to happen on a consistent basis. In my interviews with great troubleshooters, all have mentioned deep knowledge of the system they're working on as a key weapon in their arsenal. However, I hope to show you that deep experience with a particular system **is not necessary** to start and guide the troubleshooting process. I believe the right strategies, coupled with the right mindset and behaviors, are equal to experiential troubleshooting skills. I'm not knocking experience, it's a powerful tool and I want you to tap into it whenever you can. I'm just challenging the notion that it's the only entry to effective troubleshooting. Also, I've seen the belief that experience is needed as an excuse to do nothing. "We don't know anything about this machine, so let's wait until Jim takes a look at it…," only to have Jim arrive and have no better ideas than the strategies presented here.

Specific system knowledge is like a totally sweet Camaro in the mating game: it's not necessary, but it doesn't hurt either.

(image: Don O'Brien / CC BY 2.0)

But, It's Great If You Have It

While specific system knowledge isn't necessary to begin troubleshooting, it can be a huge bonus to the virtues and strategies. A big part of "specific knowledge" is being able to answer the question: **"What is normal?"** You may think you know what is normal, but unless you're collecting and analyzing data about your systems on a regular basis, you don't have a clue. Trust me.

You take the cover off a machine and see something that looks out of place, or notice some strange output, or hear a weird noise, or smell something unsavory coming from a machine. The next question that will invariably pass your lips will be: "Is this normal?" This question comes up all the time, and the answer is usually "Umm…I don't know." While "Is it plugged in?" may be troubleshooting's most famous question, **"Is this normal?"** is the one most likely to be uttered by an actual troubleshooter while on the job.

Not knowing the normal operating ranges within your systems will make it difficult to determine if a particular fix has worked. If the goal of troubleshooting is getting back to normal, and you don't know what that means, you'll merely be hoping that you've solved the issue. While it might appear to be "working," your system could be operating in an entirely new range. Your fix may just be a temporary reprieve before another catastrophic failure. So, without further ado, let's enter the world of data collection.

Gauges are all about the current state.
(image: Branden Williams / CC BY 2.0)

Gauges

Gauges will tell you "what's happening now" and consequently are a great way to begin any data collection regime. Gauges don't have a memory and therefore don't allow for comparison to the past, but deploying them is a good first effort.

For some systems, knowing the current state may suffice if the operating parameters (i.e., the *desired state*) are relatively well known and unchanging. The manufacturer may have published guidelines that can be a sufficient proxy to "what is normal." Or, "normal" may be well described in your own documentation or known among your operators.

Hopefully, when problems arise, they will be visible on your gauges. Imagine a valve installed to control the flow to a water delivery system:

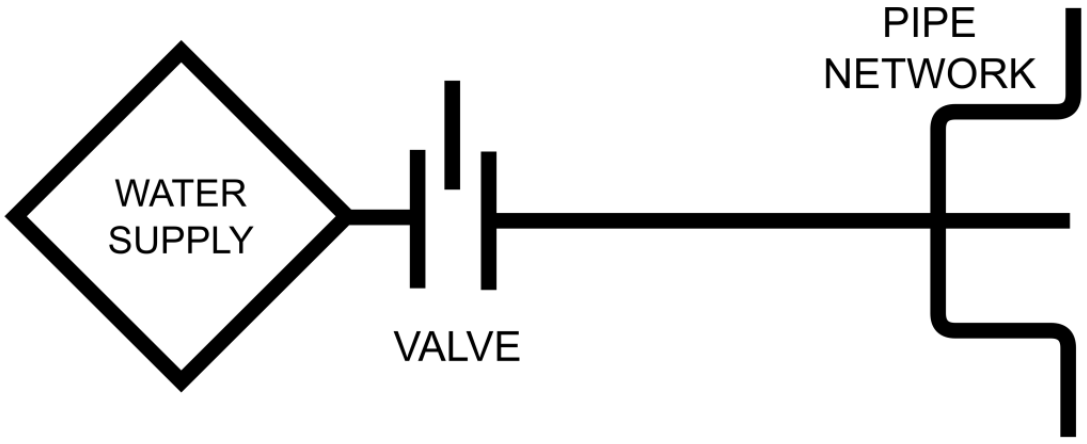


Diagram: a valve controlling flow to a network of pipes.
(image: © Jason Maxham)

Let's say that pipes downstream from the valve keep bursting and, after an investigation, the cause is suspected to be that the the valve was improperly set and not restricting flow adequately. There are no markings on the valve, so it's really difficult to know what flow level has been selected. So, you put a gauge on the output side of the valve in an attempt to better understand the situation:

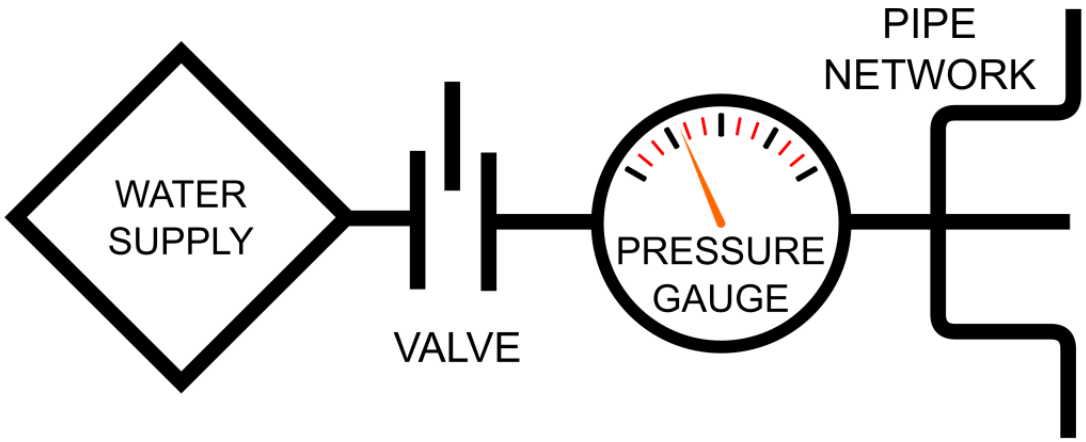


Diagram: valve with a gauge to monitor pressure.
(image: © Jason Maxham)

Before, the valve setting would have been our focus: it was the only thing we could control. Previously, we had no information about the pressure flowing through the system, except of course when it was too much and our pipes were bursting! Now, our attention can shift to the reading on the pressure gauge, with the valve merely being a means of selecting the desired pressure level. The addition of the gauge leads to improved awareness of certain scenarios that would have been difficult to detect in its absence:

1. If the gauge reads higher or lower than desired, the valve is not set properly and requires adjustment.
2. If adjustment of the valve has no impact on the pressure reading, we can hypothesize that the valve is stuck or broken.

Within this simple system, troubleshooting can now be done efficiently with just this single gauge. Knowing the current state (i.e., the pressure reading) will point the way to the problem.

Gauges Don't Have A Memory

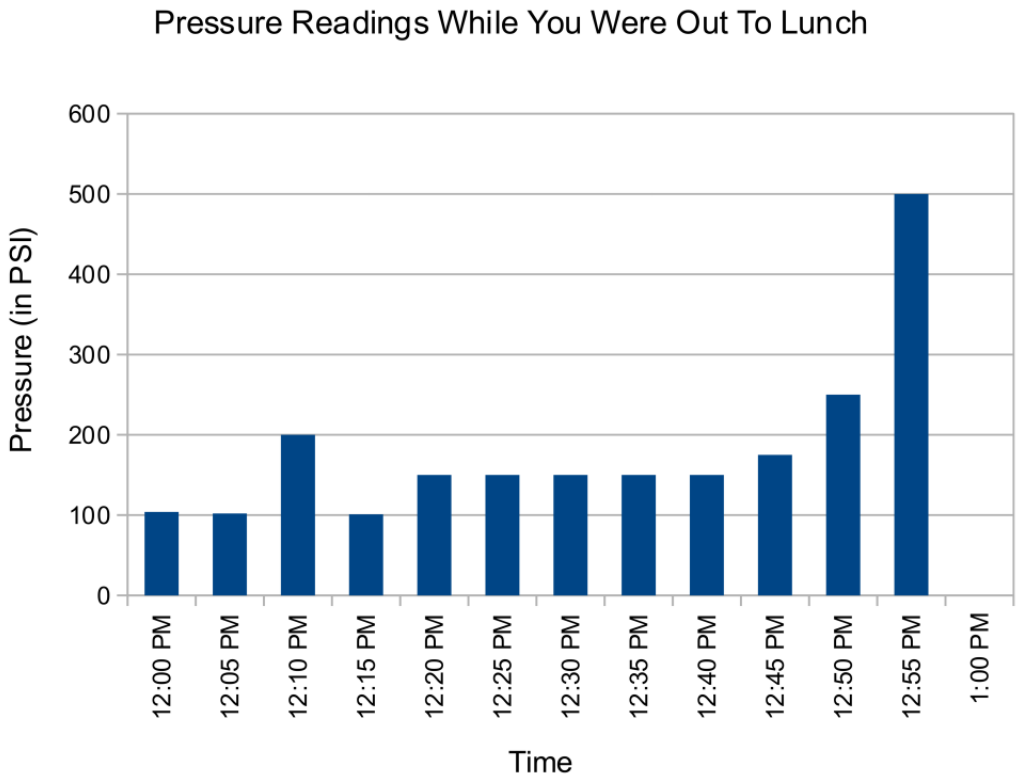
What gauges will miss (unless you're watching them 24/7) are intermittent problems and erratic behavior in the run-up to a failure. Imagine a boiler system with a pressure gauge that your boss has asked you to monitor. You look at the gauge hourly throughout the morning. As your omniscient narrator, I was keeping track of the readings, even though you weren't:

Time of day	Pressure reading
08:00 AM	104 psi
09:00 AM	101 psi
10:00 AM	102 psi
11:00 AM	102 psi
12:00 PM	104 psi
01:00 PM	0 psi

When you get back from lunch, you hear wide-eyed tales from your co-workers of a loud explosion in the boiler room! How could this have happened? After all, you were diligently monitoring the pressure gauge all morning…

The problem is that many failures will start within (or near) the normal operating range, until breaking out. Again, unless you had been present at the time of the failure (in this case, it's probably good you weren't!), you wouldn't have seen the exponential rise in pressure before the explosion. During the time of your monitoring, the pressure readings were in a tight group between 101-104 psi. There was a slight uptick at noon to 104 psi, but from these data points alone it would be difficult to tell if that was a harbinger of trouble (plus we saw a reading of 104 psi at 8am). So, you arrived back from lunch to an exploded boiler…and unfortunately a gauge that now reads zero (if it can be found among the wreckage).

In the aftermath of an incident, knowing the current state isn't very useful. What the gauge reads *now* isn't nearly as useful as what it was reading between noon and the time of the explosion. If you had an automated data collector with graphing capabilities, you would have seen this:



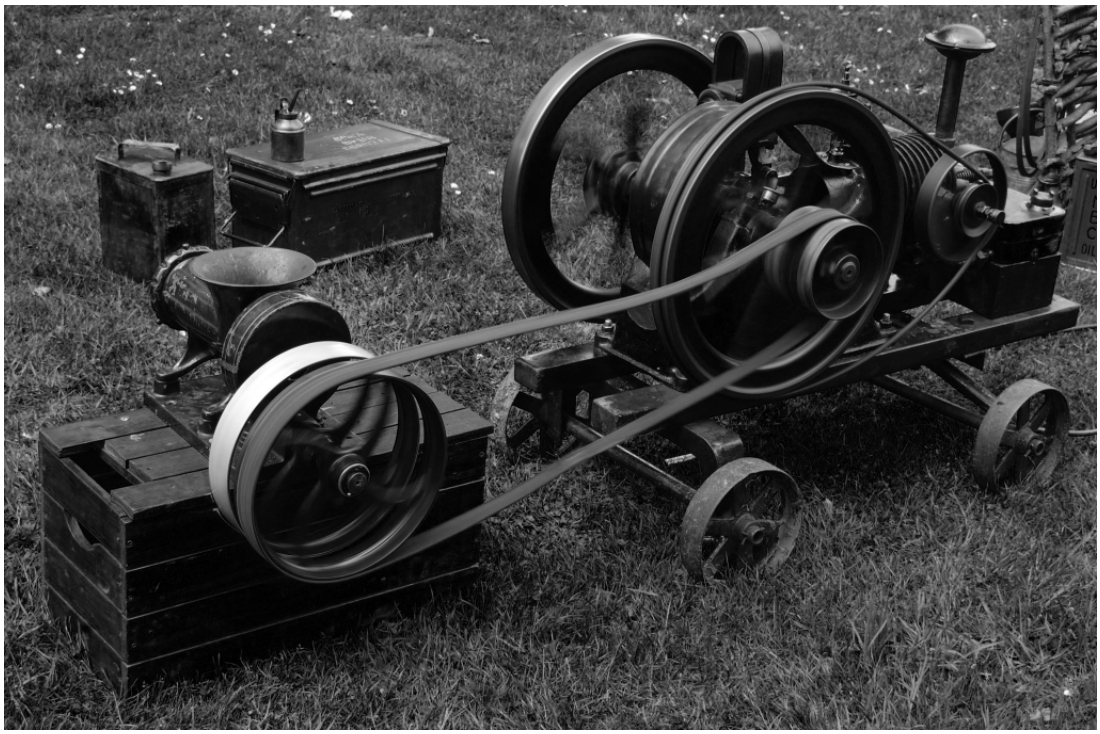
Graph: pressure readings during the noon hour.
(image: © Jason Maxham)

This is so much more useful than just knowing the current state! You can see the pressure building in the run up to the explosion and begin to form a theory about its cause. Here we can see a spike to 200 psi around 12:10pm and then the pressure briefly returns to a normal level. Perhaps a safety release valve kicked in and brought the pressure back down? After that, the pressure hovers around 150 psi, until going exponential. If there was a safety release valve, it clearly failed this second time around. Looking at the graph, it's easy to pinpoint the time of the explosion as occurring between 12:55pm and 1:00pm.

By the way, I have seen this particular failure pattern so many times in so many different contexts. A system will be operating normally, then experience some kind of shock. After the shock, it will operate in a new, above-or-below-average range for a short period of time, before experiencing a complete meltdown.

So Many Things To Track

Gathering and analyzing data has an opportunity cost, so you'll want to be judicious about how you allocate the time you devote to a data collection project. Also, in even the simplest of systems, the number of different things that can potentially be monitored and tracked is infinite. To prove this to you, take a look at this belt-driven contraption:



Even within this simple system, there are an infinite number of things that could be monitored.
(image: Shaun Wallin / CC BY 2.0)

Let's say you wish to monitor and collect data on how fast that longest belt is spinning. To do this, you'll measure the length that the belt travels past your monitoring point within a given time period. Depending on the problems you're trying to spot, just this parameter alone could result in an infinite number of possible data collection schemes. Perhaps the belt occasionally slips and stops moving momentarily (and you'd like to know when and how often), but that's only something you'll notice if you look at very small time slices (like sub-second).

Different sampling rates (i.e., the period between making subsequent observations) can lead to an endless number of possible data collection schemes. You could observe how much belt passes an observation point over the course of a second, 1/10 second, 1/100 second, 1/1000 second, etc. to infinitely small time periods. Smaller time slices might not necessarily be *useful*, but each sampling rate would give you slightly different information. And this is just a lone belt on a simple whoozy-whatzit. Imagine the data collection possibilities in an automobile factory or an oil refinery. Infinity times infinity!

Start Small

Given that the amount of data you could collect on a machine is infinite, you're going to need to find a way to prioritize and make what you do collect manageable. One very efficient way is to let your breakdowns dictate what gets watched. Often when troubleshooting, you'll find yourself saying "I wish I knew what was happening to this *before* the failure…" Those moments are great indicators of what to monitor in the future. There is little wasted effort with a scheme like this: you're focusing on things that *actually* have tripped you up in the past!

I think you can do yourself one better by proactively surveying your systems for things to monitor so you'll be ready when the next crisis occurs. If you have even a passing familiarity with a system, you can usually make an educated guess about what you'd like to know about in the event of a breakdown (run through some imaginary scenarios in your head or review your recent incidents for ideas). From there, let real problems guide the additional parameters you add to your data collection regimen.

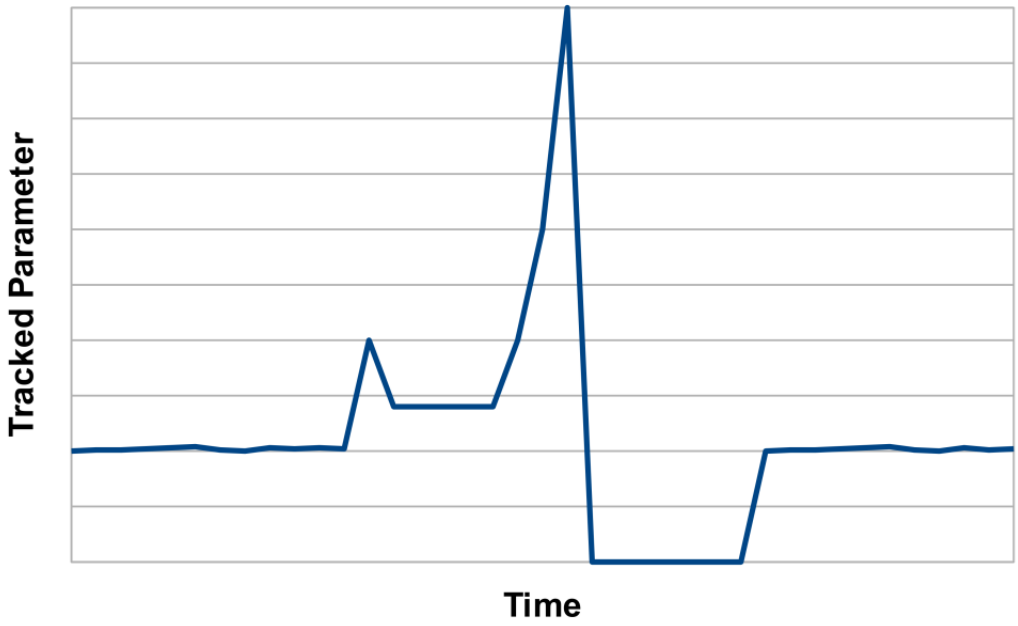
Baselines Verify Fixes

For complicated systems, it can be difficult to know if a fix has worked without a good data collection system in place. I've worked on problems where I was sure I had nailed a fix only to have things unravel later. Over time, I began to notice a huge difference between systems where we had solid data coverage and those where we were "flying blind." The data advantage operated on two levels:

1. Good data leads the way to the discovery of the cause.
2. Good data shows you if a fix is actually working.

Both of these benefits immensely increase your confidence in your repairs. It all comes back to the theme of this section: if you can't say what is "normal" you won't really know if the repaired system is functioning "normally."

When your efforts finally pay off, you'll get graphs that look like this:



Graph: a data collection victory.
(image: © Jason Maxham)

Notice how everything, from regular operation before the incident, to the onset of trouble, to the meltdown, to the recovery after the fix, is so *visually* apparent in this graph. It's abundantly clear what "normal" was before the breakdown, and it's just as clear that you have indeed returned to that normalcy after the repair. When this happens for the first time, expect to get a little choked up. Beautiful.

War Correspondent

During crises, you should be collecting a very specific type of data: what fixes you've tried and when. I've been called in to troubleshoot some very hectic situations and gone immediately into "let's try this, then let's try this, and then how about this" mode. The energy of the crisis fuels an endless stream of ideas and avenues to pursue. However, if the problem persists and you don't find a quick fix, you may find that you begin to accidentally repeat yourself. You may catch yourself in a moment of deja vu and ask, "Didn't we already try this?" To avoid this looping, I suggest you start a simple troubleshooting log at the beginning of any crisis. If you want to call it "My very secret and very special diary of my most precious fix-it feelings," that's your prerogative. Oh, and this idea isn't only for crises: any long-term troubleshooting project benefits from such a record of happenings.

A related concept is the maintenance log, which is a list of changes made to a machine, including when they were made and by who. You might have seen these pasted in the interior of a service panel or dangling from a clipboard nearby. Along with other maintenance records, these are a very valuable source of data that are great for answering the question: "What's changed?"

Start Now

The past will be murky if you weren't keeping track of it, so orient yourself to a bright future and commit to start collecting data. Yesterday may remain shrouded in mystery, but at least you'll have data to make comparisons for the next incident.

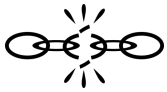
References:

- Header image: Leffler, W. K., photographer. *NATIONAL OCEANOGRAPHIC AND SCIENCE CENTER – seismograph report of L.A. earthquake.* 1971. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017646241/.

Is This Normal? An Ode To Data Collection was originally published April 11, 2012.

Notes:

Zen And The Art Of Routine Maintenance



> Routine maintenance is always a good thing to do. It's a pain...but it needs to be done.
>
> **Austin Quade**

I've heard that being a doctor can sometimes be a downer because all day long you are seeing people at their worst. No one randomly schedules an appointment with their physician to enthusiastically reveal how good they're feeling!

Likewise, if you're called on to troubleshoot, it's because something has gone wrong. Your job or business may be on the line. Troubleshooting can be fun, especially if you bring a sense of curiosity and discovery to the enterprise. However, the baseline experience usually involves someone who has been inconvenienced by a machine failure. We only bother to fix things that matter to someone.

All this drama can be an invitation to heroism, but it's best to not get hooked on the excitement. If you find yourself in crisis mode all the time, you're either an adrenaline junkie or haven't done enough preventative maintenance. You can look at expert troubleshooting skills like having a black belt in Kung Fu: it's nice to know that you'll be able to kick some butt when the time comes. However, if you find yourself in drunken bar brawls all the time, you've probably

tuned out when your sifu emphasized self-control and peaceful ways to avoid confrontation.

As a skilled troubleshooter, it's great when you can step in and be the hero. Better still is to prevent trouble from ever starting.
(image: Andrew Magill / CC BY 2.0)

In his introduction to *The Art of War*, Thomas Cleary relates this parable relevant to our topic:

> According to an old story, a lord of ancient China once asked his physician, a member of a family of healers, which of them was the most skilled in the art. The physician, whose reputation was such that his name became synonymous with medical science in China, replied,
>
> "My eldest brother sees the spirit of sickness and removes it before it takes shape, so his name does not get out of the house.
>
> "My elder brother cures sickness when it is still extremely minute, so his name does not get out of the neighborhood.
>
> "As for me, I puncture veins, prescribe potions, and massage skin, so from time to time my name gets out and is heard among the lords."
>
> **Thomas Cleary, The Art Of War** [1]

When it comes to troubleshooting, you want to be like the eldest brother. Routine maintenance is your opportunity to prevent trouble before it has a chance to take shape.

Maintenance Windows

A maintenance window is a prearranged time when preventative maintenance is done. Because a machine is idled, work will be slowed for the duration of the window. You wouldn't agree to this without some upside: the benefit

gained is that you will reduce costly *unscheduled* downtime. As clients come first, don't expect that you'll be able to have your maintenance windows at convenient times (for you), like on a Monday morning from 9am-11am. If you can convince management of the benefits, expect something more like a Sunday night when the business is closed. Again, tradeoffs. Would you rather give up a Sunday night once a month, or see your co-workers in their pajamas because the team had to be called in on a Tuesday at 3am for a meltdown? Unless you have co-workers who look good in their pajamas, the choice is clear. For some companies, the cost of unscheduled downtime can be in the millions of dollars per hour. All it will take is a couple of high-profile incidents to sell your "radical" notion of using maintenance windows as a preventative measure.

Any modern production environment will have more opportunities for maintenance than time will allow. You'll need to prioritize…
(image: © Jason Maxham)

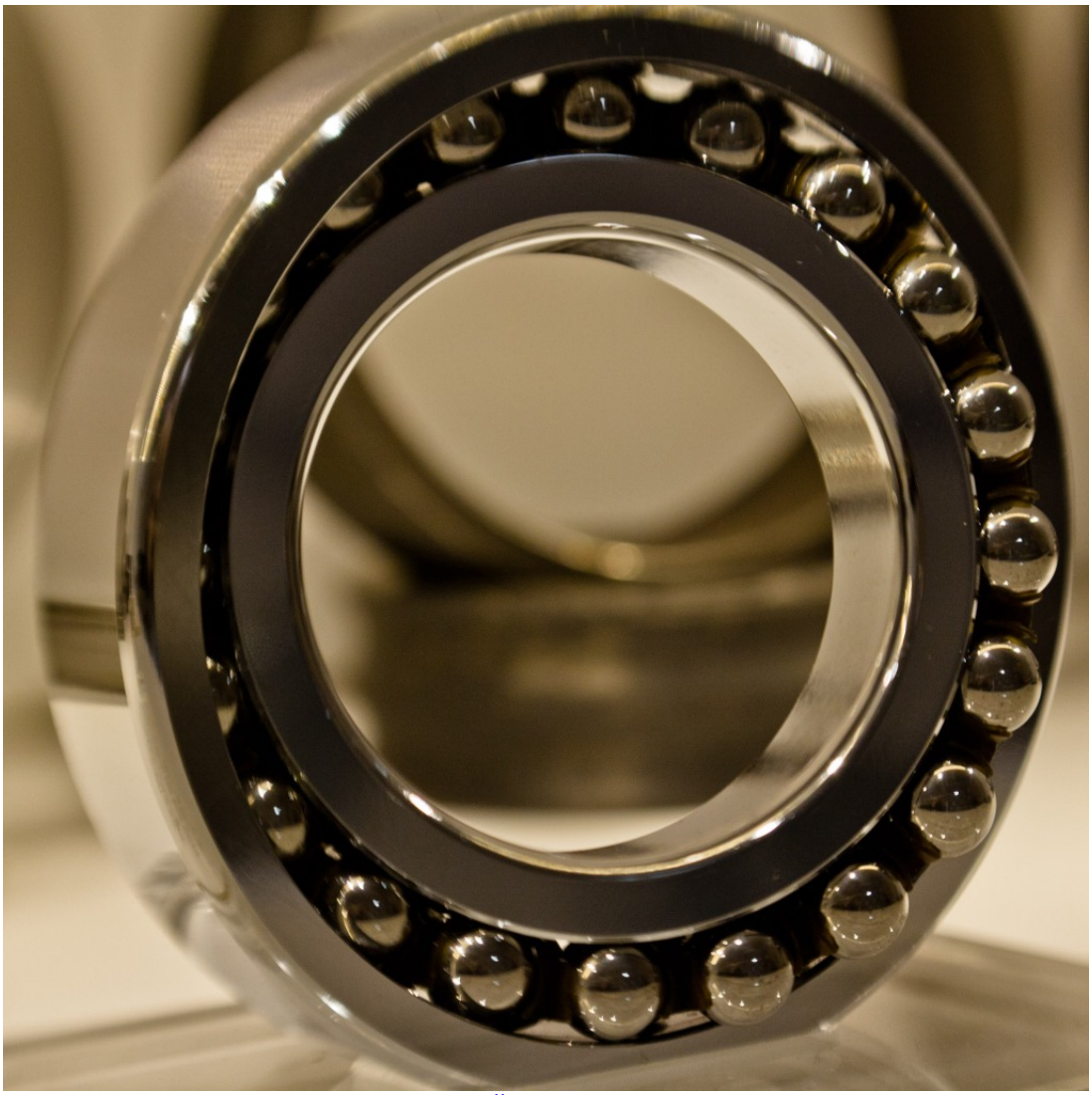
Planning A Maintenance Window

There is an art to planning and executing a maintenance window. Typically, the time you have to work with is finite and fixed on the calendar. If you are fastidious about collecting all of the possible maintenance tasks you could pursue, you may find it exceeds your allotted window duration many times over. That's okay, that will simply force you to decide what's most important. Over time, you'll get an intuitive sense of what to prioritize by this formula:

Chance of Failure × Cost of the Failure = Expected Cost of the Failure

Let's work through an example of choosing between some possibilities for a maintenance window. You are employed at Widget Inc. and, after some costly downtime, you have negotiated a monthly 1-hour maintenance window from management. On the top of your list are the following two items:

1. Replace the worn ball bearings on the factory's conveyor belt:



(image: Pelle Sten / CC BY 2.0)

2. Upgrade the software that controls your computer-controlled routers (in manufacturing, a router is a tool used to hollow out a piece of hard material, like wood or metal):

(image: Fagor Automation / CC BY-ND 2.0)

Given that we'll only have 1 hour to do maintenance, how do we choose? We'll start with a basic analysis, using the formula above. For the bearings on the conveyor belt, we'll note that there's only one conveyor belt in the whole factory. When it's down, the company is effectively shut down. To bootstrap a comparison, we'll roughly calculate the cost by spreading the company's annual revenue over each hour, using that to estimate the cost of an incident. If an unscheduled bearing change in an emergency situation will take about 6 hours (¼ day) and the company makes $100 million dollars a year, we can get a sense of how much an incident will cost:

(.25 days ÷ 365 days) × $100,000,000 annual revenue = $68,493.15

The automated routers are a different story: the company has several of these machines and if one malfunctions, the aggregate speed of production is reduced by 1/n (where n is the number of machines). There are 10 machines and so if 1 breaks down, the slowdown will be:

1 ÷ 10 = 10% reduction in line speed

Let's say it takes one day to repair a router in the case of an unplanned outage. Now, we can estimate the cost of downtime:

(1 day ÷ 365 days) × (1/10) × $100,000,000 annual revenue = $27,397.26 per incident

At this point, if only one maintenance item could be accomplished, you'd be biased towards servicing the bearings ($68,493 > $27,397). By the way, the cost of the 1-hour maintenance window itself can be calculated as well:

((1 ÷ 24) ÷ 365 days) × $100,000,000 annual revenue = $11,415.53 per 1-hour maintenance window

That's a pretty expensive maintenance window! It would imply that incidents with an expected cost less than the cost of the window shouldn't be given time for maintenance. However, there are many other things to consider when calculating the "cost" of downtime. Idling your workforce, missing deadlines, and many more factors will result in expenses both tangible and intangible (e.g., loss of goodwill with a key client).

These cost of downtime calculations are a good first start, but you may have noticed one critical piece of information missing: the relative probability of each kind of failure. You may be drawn to changing the bearings, given that the cost of a bearing failure on the assembly line is 2.5 times more expensive than a router failure ($68,493 ÷$27,397 = 2.5). But, if a router's **rate of failure** was 2.5 times that of the bearings, they would extract an equal cost. One more thing to think about is the relative number of each item within your the total infrastructure. Let's say that there are 20 bearing

units in the assembly line (and 10 routers, as previously noted). Our calculations need to take into account that there are twice as many bearings that could fail.

We've come a long way in our analysis, but our equations are still missing some information:

- **Expected cost of a bearing failure = 20 y × $68,493**
- **Expected cost of all routers failures = 10 z × $27,397**

Where y and z represent the rate of failures of the bearings and the routers, respectively. Even at this point, we can do some quick math to guide our decision. What would the failure rates of bearings and routers have to be for their total cost to be equal?

$10z \times \$27,397 = 20y \times \$68,493$

$z \times \$273,970 = y \times \$1,369,860$

$z = (\$1,369,860 \div \$273,970) \times y$

$\boldsymbol{z = 5y}$

For these two scenarios to be equally costly, a router would need to be 5 times *more* likely to fail than a set of bearings. Based on your experience with bearings and routers, this is one more way to evaluate your options without historical failure data.

Known Unknowns

When the chances of various failure scenarios are unknown, like in our example above, choosing maintenance tasks will a judgement call. If you are just starting to track your breakdowns in a systematic way, you'll need to estimate and then revise later as the evidence accumulates (see "Is This Normal?" for ideas and inspiration on data collection). Even if you have been collecting data, there will be some failures that have never happened, meaning you'll be trying to prevent the unknown. If that seems like an amorphous thing to advocate for, backing you is the collective experience of millions of technicians who know that an "ounce of prevention is worth a pound of cure." It's better to deal with a machine on your terms, not in the middle of something important like trying to meet a deadline. If you forego regular maintenance, you're basically letting the machine decide how it will inconvenience you. Personally, I'd rather be in control of the timing. Where to start? If you've been around your organization for any length of time, you'll probably have an intuitive sense of where to devote your maintenance resources. Otherwise, do an inventory and rank your systems according to cost of downtime.

The Right Frequency

If you've ever changed the oil in your vehicle, congratulations, you've conducted a maintenance window. You set aside time (maybe when you passed an oil change shop and looked at the "Next Oil Change" sticker in shock) and your car was unusable while the oil was being changed. You weighed the inconvenience of spending time drinking bad coffee and reading old magazines in the waiting room of a service station against the cost of an engine failure.

Speaking of changing your oil, that's a great example of where you have to decide how closely you adhere to a manufacturer's guidelines. Although every mechanic I've talked with endorses frequent oil changes, they are quick to admit that your owner's manual may recommend a frequency that is excessive. You might notice the difference if you drove your car for 2 million miles, but consumers (buying new) only keep their cars for an average of 6 years[2].

In my experience, I've seen recommended maintenance intervals range from opportunistic revenue generators to woefully inadequate (or non-existent). On the opportunistic side, my printer is always telling me to change the ink cartridges. I'm not surprised, given that the liquid inside is worth more than gold. Parts and service can be an important revenue stream for a company, so don't be shocked when encountering aggressive replacement schedules. Given the economic incentives at play, be sure to deploy an appropriate amount of skepticism and double check the need against your own failure data. Also, keep in mind that routine maintenance doesn't have to expensive, I would do a quick visual inspection of our office infrastructure on a weekly basis that took no more than 10 minutes.

On the other side of the coin, you may find a manufacturer recommends a maintenance schedule that allows an

unacceptable number of failures. Maybe their recommendations are geared towards the "average" customer, while your usage profile is more strenuous. I've also seen systems that don't come with any maintenance advice at all. If you are dealing with a custom-made machine that you cobbled together, obviously the ideal maintenance schedule is something unknown that will need to be discovered. By you!

Go Ahead And Let It Fail

If treated it as a learning opportunity, you don't need to fear failure. This applies to life as well as routine maintenance. When Discovery Mining first started out, one of the most annoying things to maintain was the printer. The darn thing was always breaking down at the most inopportune times—usually right before an important sales meeting. Of course, when that happened, it was a "drop everything" emergency that required immediate attention.

I've talked before about "listening to machines" but this printer was a real chatterbox when it came to error messages. Just like being trapped in a boring conversation at a party, you eventually tuned it out. It wanted everything to be replaced, on a constant basis. At first I was responsive, but there didn't seem to be a tight correlation between replacing parts and preventing breakdowns. I began to suspect the rampant "replace X, it will fail soon" error messages were more of a revenue-enhancing ploy by the printer company.

My solution was to get a second printer. This put an end to the crises: if one of the printers was malfunctioning, you could always use the other one. I also started to note (but not act on) the various error messages, instead letting each printer go until it couldn't print any more. Doing this, I began to get a better sense of what was required to keep these printers up and running. That is, which preventative maintenance items were worth my time. The buffer provided by the second printer gave me the breathing room to learn what was really going on.

Adding a layer of redundancy is a general method to enhance (or even replace) your routine maintenance program. If the cost of failures is low and swapping is easy, then consider the very efficient "do nothing" protocol (patent pending). If you think about it, this is the scheme most people apply to their TV remote control. You don't test the batteries in your TV remote every day (if you do, get help now!). When the batteries die, you simply replace them. Except for the pain of having to leave the couch, a dead battery in this context is no big deal. Another bonus is maximizing the use of your resources: you get the absolute most out of something by using it until the end.

Put The Routine In Your Maintenance

I'll leave this discussion by noting that periodic maintenance requires discipline. If you've taken the time to study a system and determine that maintenance will head off trouble, but then fail to do it… This is the most tragic kind of failure, when you *know* but fail to act. Make it easy on yourself, set up automatic reminders. I also favored vendors that would were good at keeping to a maintenance schedule, independent of me badgering them. Find people like that, and you'll vacation with confidence!

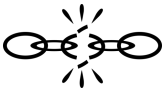
References:

- Header image: Detroit Publishing Co. *Oiling up before the start.* ca. 1904. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016803601/.
- [1] Sun Tzu, translated by Thomas Cleary, *The Art of War* (Boston: Shambhala, 2005), pg. xi.
- [2] Jerry Hirsch, "Americans keep their new cars for almost six years," *Los Angeles Times,* February 21, 2012.

Zen And The Art Of Routine Maintenance was originally published May 1, 2012.

Notes:

Storm's A-comin'



> I'm not surprised that I'm often surprised.
>
> **Alex Chaffee**

Carnies get nervous when an accident happens, because they say that "Bad luck happens in threes." Even if you've killed someone whose first name is Darth, there's always another Sith Lord running around. When you first see an ant crawling across your kitchen floor, you know there are thousands more, lurking unseen. The darkening of the horizon as storm clouds appear foreshadows rain. The small tip of an iceberg peeking above the water hints at a much larger threat below the surface.

These leading edge moments of discovery are first encounters with things yet to come. When troubleshooting, we also have these initial contacts with circumstances that may foretell of larger disasters. As my troubleshooting skills grew, I became increasingly skeptical of "one-off" breakdowns. There was usually more to the story. If you're curious and tuned into what's going on, you may be able to mitigate an impending disaster.

This section isn't just for professional troubleshooters who work in high-risk industries (like nuclear or petroleum), because machine-related "disasters" can be lurking anywhere in your life. If the brakes failed on your car or if your house started on fire because of a missed warning sign, it might not be featured on the front page of *The New York Times*, but it would be still be a big deal—to you!

As you can see, there's always much more below the water.
(image: NOAA's National Ocean Service / CC BY 2.0)

Look For The Signs

After something has been successfully repaired, it's a natural instinct to take it easy. The excitement is over, so you grab a cold one, pat yourself on the back, and put your feet up on the desk. But, you may want to reconsider that habit. That's because large disasters often start out with small, "normal" failures. Here's some insight from the Deepwater Horizon accident, the largest marine oil spill in the history of the petroleum industry:

> About seven hours before the Gulf of Mexico oil well blowout of 2010, a group of four company VIPs helicoptered onto the drilling rig in question, the Deepwater Horizon. They had come on a "management visibility tour" and were actively touring the rig when disaster struck.
>
> There were several indications in the hours before the blowout that the well was not under control, in fact that it was "flowing", that is, that oil and gas were forcing their way upwards from several kilometers below the sea floor. These indicators were all either missed or misinterpreted by the rig staff. The touring VIPs, two from BP and two from the rig owner, Transocean, had all worked as drilling engineers or rig managers in the past and had a detailed knowledge of drilling operations. Had they focused their attention on what was happening with the well, they would almost certainly have recognized the warning signs for what they were, and called a halt to operations. But their attention was focused elsewhere, and an opportunity to avert disaster was lost.
>
> **Andrew Hopkins, *"Management Walk-Arounds: Lessons from the Gulf of Mexico Oil Well Blowout"*** [1]

Maybe you're thinking, "Wait a second, how can this be practical advice? There are so many little things that go wrong in the course of a day. Not all of them lead to deadly disasters!"

You're right, not everything that goes wrong is the harbinger of a catastrophe. If you're in charge of an operation that carries the risk of a major incident, the likes of which would show up on the evening news (I'm talking about sugar mills, oil rigs, nuclear power plants, etc.), you need to be engaged with the field of process safety. Any industry with significant risks of a major human or environmental disaster should have safety as a core part of their culture, woven into every process. That's beyond the scope of this work, so I invite you to do your own reading in this area.

Most disasters have humble origins.
(image: Jonathan Perera / CC BY 2.0)

Instead, I want to make the connection with you in the role of a troubleshooter. Because catastrophes frequently begin with small breakdowns, they're exactly the kind of things that front-line troubleshooters will be first on the scene to investigate. There are certain kinds of failures which should prompt you to dig deeper, as explained in the book *Recognizing Catastrophic Incident Warning Signs:*

> There are many types of warning signs [for catastrophic incidents], including the following:
> · Early indicators of failure that provide opportunities to take appropriate action. Process equipment that is not functioning properly may be prone to failure. Organizations sometimes ignore these on the basis that they will address the issues later (or if the problem escalates).
> · Suggestions that a major incident may be imminent. An example might include a piece of process equipment reaching its end-of-cycle or retirement limit.
> · Indicators that are less obvious and require detailed analysis. For this reason, a practical follow-up option is to conduct an audit to help ensure that programs and systems are managed effectively.
> · Seemingly insignificant issues that, when combined with other warning signs, suggest a breakdown of management systems.
> · Actual incidents with measurable consequences. If we ignore these, they can increase in frequency or magnitude and contribute to a catastrophic incident.
>
> **Center for Chemical Process Safety, "Recognizing Catastrophic Incident Warning Signs in the Process Industries"** [2]

I'll add a few more items to this list of warning signs:

- ***New* failure conditions:** whenever something breaks down in a way that you haven't encountered before, watch out.
- **Malfunctions that span multiple systems:** problems that strike in multiple places simultaneously are a big red flag for a coming catastrophe.
- **Weird stuff:** intermittent problems, issues that seem to magically resolve themselves (i.e., you investigate and everything seems to be fine), failures that are difficult to quantify because of conflicting or incomplete reports, etc.

Dodging Icebergs

That little bit of ice that sticks above the water is the phone call from a field technician about an unexplained interruption at a customer site. It's a couple of identical bug reports from your best clients. It's the "check engine" light coming on in your car. It's that "something weird" you've never seen before in the plant.

Perhaps a "major incident" in your line of work might not be the end of the world, but it might inconvenience customers, put you out of business, or cause injury. Smaller organizations may have the same operational risks as large ones, but few or no staff dedicated to "process safety." In these cases, those called on to troubleshoot may be the only ones with the visibility to take that extra step and flag a situation for further review, before it gets out of hand. Awareness is key: I just want to plant the seed in your mind that you can ask, "Is this part of something much bigger?"

Stay Awake A Little Longer And Run The Numbers

Whenever you see something that matches the "warning signs" list above, it's time to take action. When I was in charge, I always liked to reserve extra time after troubleshooting for a brief audit of our infrastructure: a quick once over to make sure everything was in order.

Easy-to-access historical records and automated monitoring are extremely helpful in this regard and can help put a suspicious failure in context. If you're only working with external symptoms and have no way to know if the rest of your infrastructure is okay, you're flying blind. A malfunction could be a one-off, or it could be a bellwether of terrible things to come. If you have good data, you have a shot at telling the difference.

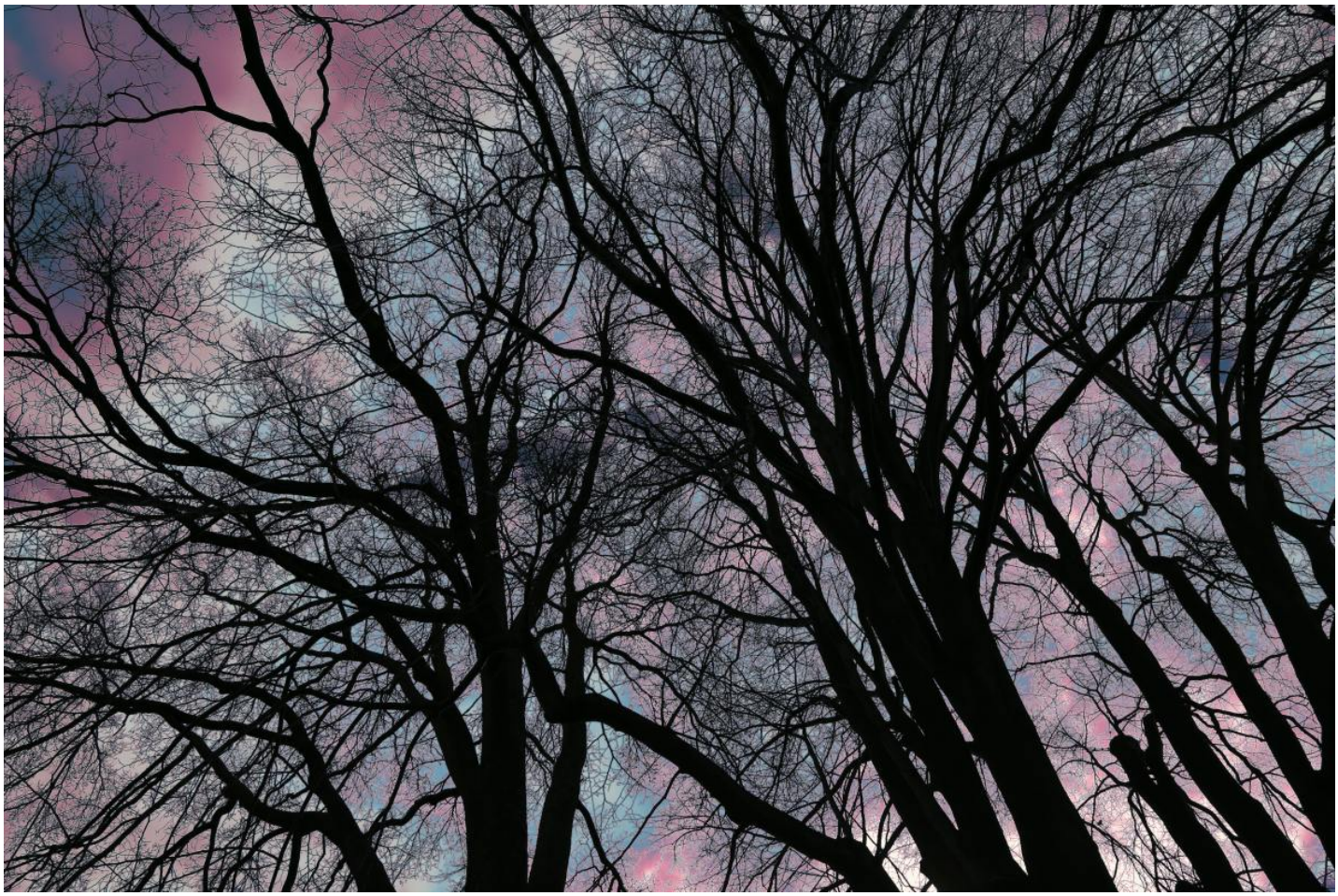
References:

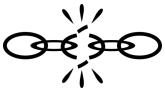
- Header image: Frans Ruiter, photographer. Retrieved from Unsplash, https://unsplash.com/photos/M04dMrtjm3U.
- [1] Andrew Hopkins, "Management Walk-Arounds: Lessons from the Gulf of Mexico Oil Well Blowout", February, 2011, pg. 3.
- [2] Center for Chemical Process Safety (CCPS), *Recognizing Catastrophic Incident Warning Signs in the Process Industries*. (Hoboken: John Wiley & Sons, 2011), pg. 2.

Storm's A-comin' was originally published February 18, 2013.

Notes:

Troubleshooting Trees



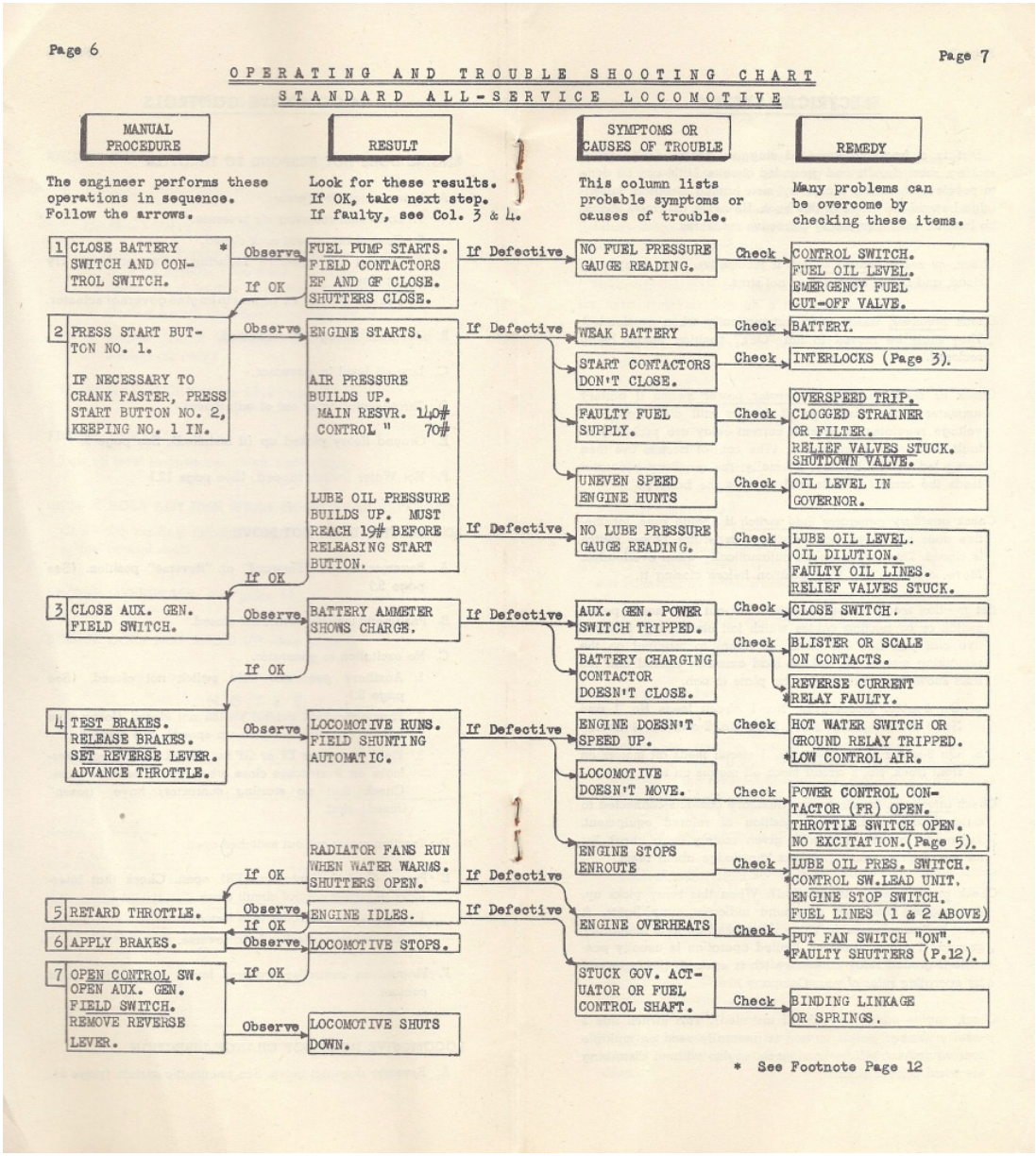


> Good documentation will tell you what to do and what you *shouldn't* do.
>
> **Austin Quade**

One thing I really hate is making the same mistake twice. In my tenure as a CTO, I always wanted to move forward, never making the same error again. I tried to clear the way for our team to make *new* mistakes.

Some failures happen over and over again. As you gain experience with a machine, you'll begin to see all the different ways it can break down. As discussed in "Same Symptom, Different Causes," there will be times where a machine will *appear* to be broken like a previous time, but actually have a completely different underlying cause (and associated fix). Figuring this out can be costly, so let me introduce a way of preserving and communicating your hard-won troubleshooting knowledge: a **troubleshooting tree**.

A troubleshooting tree is a formal description of the troubleshooting process for a particular problem. The tree walks you through a rehearsed fix-it routine, with branches along the way where you stop, gather information, and make choices about which way to proceed. Let's look at an example:



Page 6

Page 7

OPERATING AND TROUBLE SHOOTING CHART

STANDARD ALL-SERVICE LOCOMOTIVE

MANUAL PROCEDURE		RESULT		SYMPTOMS OR CAUSES OF TROUBLE		REMEDY
The engineer performs these operations in sequence. Follow the arrows.		Look for these results. If OK, take next step. If faulty, see Col. 3 & 4.		This column lists probable symptoms or causes of trouble.		Many problems can be overcome by checking these items.
1 CLOSE BATTERY SWITCH AND CONTROL SWITCH. *	Observe	FUEL PUMP STARTS. FIELD CONTACTORS EF AND GF CLOSE. SHUTTERS CLOSE.	If Defective	NO FUEL PRESSURE GAUGE READING.	Check	CONTROL SWITCH. FUEL OIL LEVEL. EMERGENCY FUEL CUT-OFF VALVE.
	If OK					
2 PRESS START BUTTON NO. 1. IF NECESSARY TO CRANK FASTER, PRESS START BUTTON NO. 2, KEEPING NO. 1 IN.	Observe	ENGINE STARTS. AIR PRESSURE BUILDS UP. MAIN RESVR. 140# CONTROL " 70# LUBE OIL PRESSURE BUILDS UP. MUST REACH 19# BEFORE RELEASING START BUTTON.	If Defective	WEAK BATTERY	Check	BATTERY.
				START CONTACTORS DON'T CLOSE.	Check	INTERLOCKS (Page 3).
				FAULTY FUEL SUPPLY.	Check	OVERSPEED TRIP. CLOGGED STRAINER OR FILTER. RELIEF VALVES STUCK. SHUTDOWN VALVE.
				UNEVEN SPEED ENGINE HUNTS	Check	OIL LEVEL IN GOVERNOR.
			If Defective	NO LUBE PRESSURE GAUGE READING.	Check	LUBE OIL LEVEL. OIL DILUTION. FAULTY OIL LINES. RELIEF VALVES STUCK.
	If OK					
3 CLOSE AUX. GEN. FIELD SWITCH.	Observe	BATTERY AMMETER SHOWS CHARGE.	If Defective	AUX. GEN. POWER SWITCH TRIPPED.	Check	CLOSE SWITCH.
				BATTERY CHARGING CONTACTOR DOESN'T CLOSE.	Check	BLISTER OR SCALE ON CONTACTS.
						REVERSE CURRENT *RELAY FAULTY.
	If OK					
4 TEST BRAKES. RELEASE BRAKES. SET REVERSE LEVER. ADVANCE THROTTLE.	Observe	LOCOMOTIVE RUNS. FIELD SHUNTING AUTOMATIC.	If Defective	ENGINE DOESN'T SPEED UP.	Check	HOT WATER SWITCH OR GROUND RELAY TRIPPED. *LOW CONTROL AIR.
				LOCOMOTIVE DOESN'T MOVE.	Check	POWER CONTROL CONTACTOR (FR) OPEN. THROTTLE SWITCH OPEN. NO EXCITATION. (Page 5).
				ENGINE STOPS ENROUTE	Check	LUBE OIL PRES. SWITCH. *CONTROL SW. LEAD UNIT. ENGINE STOP SWITCH. FUEL LINES (1 & 2 ABOVE)
	If OK	RADIATOR FANS RUN WHEN WATER WARMS. SHUTTERS OPEN.	If Defective			
5 RETARD THROTTLE.	Observe	ENGINE IDLES.	If Defective	ENGINE OVERHEATS	Check	PUT FAN SWITCH "ON". *FAULTY SHUTTERS (P.12).
	If OK					
6 APPLY BRAKES.	Observe	LOCOMOTIVE STOPS.				
	If OK					
7 OPEN CONTROL SW. OPEN AUX. GEN. FIELD SWITCH. REMOVE REVERSE LEVER.	Observe	LOCOMOTIVE SHUTS DOWN.		STUCK GOV. ACTUATOR OR FUEL CONTROL SHAFT.	Check	BINDING LINKAGE OR SPRINGS.

* See Footnote Page 12

A troubleshooting tree for a train locomotive.

(source: *Baldwin-Lima-Hamilton All-Service Locomotive Trouble Shooting Handbook, February 1, 1952*)

In this excerpt, you can see standard operating procedures, symptoms of trouble, and the associated possibilities for remedying them. For example, if the locomotive's engine won't start ("Engine Starting → If Defective"), the tree has four branches leading to different solutions:

1. Weak battery → Check → Battery
2. Start contactors don't close → Check → Interlocks
3. Faulty fuel supply → Check → Overspeed trip; Clogged strainer or filter; Relief valves stuck; Shutdown valve.
4. Uneven speed engine hunts → Check → Oil level in governor.

Looking over this locomotive troubleshooting tree, you can see some symptoms have been identified as the result of a single cause (e.g., "Start contactors don't close → Check → Interlocks"). Others, like the engine not starting, can have multiple causes, which need to be serially checked until the problem is remedied. Some of the fixes in the tree are very quick (e.g., flipping a switch), while others refer to detailed procedures elsewhere in the guide that might take several pages to describe.

For the operator, information like this is solid gold. The time savings from a guide like this can be immense: even just one branch on this tree might have taken days of trial and error to figure out!

The seed of a troubleshooting tree is a desire to help someone else make it home in time for dinner.
(image: eflon / CC BY 2.0)

Planting The Tree

When planting a new tree in your backyard, there are certain things you must make sure are present: the right soil, sunlight, water, and climate. Likewise, there are things you should make available to the reader in order for a troubleshooting tree to be useful:

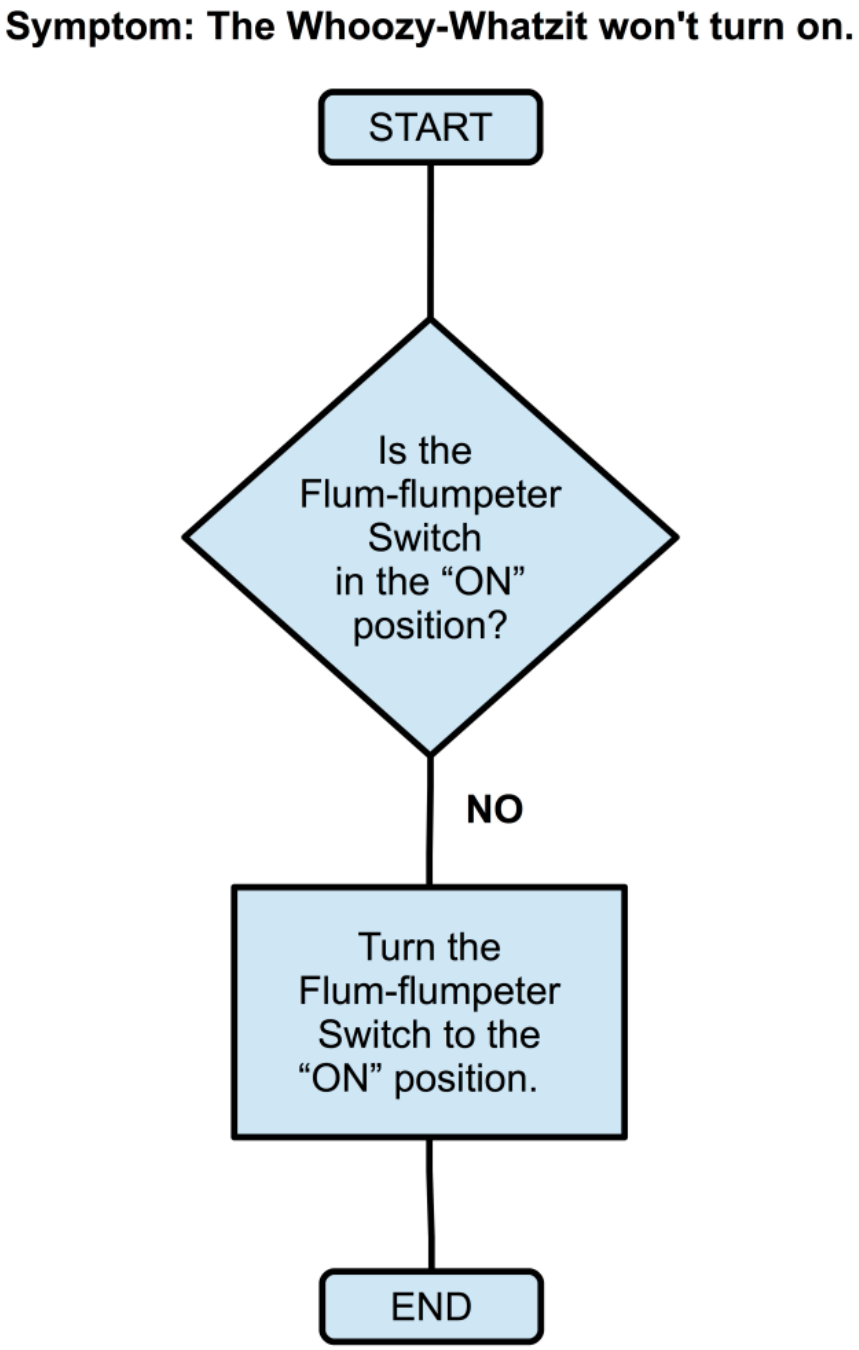
1. Necessary operating conditions, useful tools, easily missed assumptions, and setup requirements.
2. The path of investigation, with all the likely possibilities for each decision node explained.

In any kind of documentation you write, you must specify the conditions under which your methods are valid. Should a test be performed with a warm or cold engine? Is a certain software version required? Are there specific tools or resources that must be on hand to perform the operation as described? S-p-e-l-l it out, because if you don't you can be sure that someone, somewhere will misunderstand what's needed. Also, don't count on "normal" conditions to exist. After all, the fact that the reader of your documentation is troubleshooting is a strong indication that the conditions present are *not* normal!

Finally, when it comes to any kind of test, remember to include the *full range* of possibilities in your decision nodes (unless, of course, it violates the Laws of Physics or the possibility is otherwise covered in your prerequisites). If you've asked the troubleshooter to take a reading from a meter that reads from 0-100, you shouldn't just list decision arrows for the ranges of "81-90" and "91-100". What if the meter is reading "26"?

A Seedling

After you've described the prerequisites, you can start to create the troubleshooting tree itself. Lead the reader from start to finish, passing through the points needed to make the diagnosis and then finally to remedial actions:

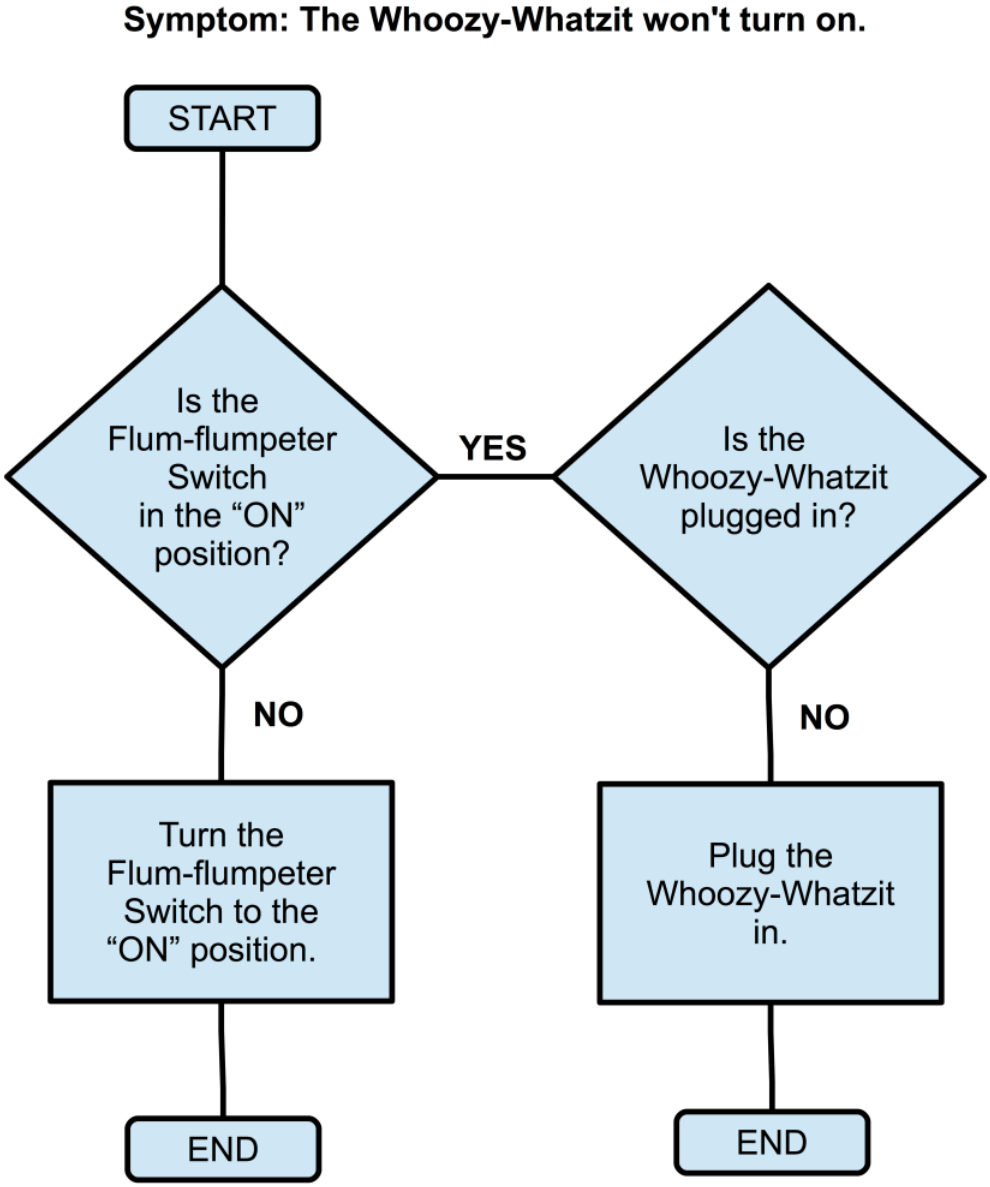


A basic troubleshooting tree with a single symptom, cause, and remedy.
(image: © Jason Maxham)

The "boxes and lines" layout is optional, you could represent the same information using words in a list. However, the visual structure of connected boxes makes it easier to understand and follow a particular troubleshooting path.

Healthy Growth

Troubleshooting trees, like real trees, are living and evolving organisms. Whenever a new cause is discovered, be sure to add a branch to your troubleshooting tree. Let's return to our fictional Whoozy-Whatzit example above. After deploying the troubleshooting tree to technicians, it's determined that the proposed remedy (i.e., turning the Flum-flumpeter Switch to "ON") doesn't always fix the problem. In fact, a *new* cause for the same symptom is discovered. When that happens, you need to grow your troubleshooting tree:



When a new cause is discovered, expand your troubleshooting tree by adding a branch.
(image: © Jason Maxham)

Ideally, you'd have a recommended course of action for every possibility (i.e., for every "yes" branch there would be a "no" branch and vice-versa). In this example, you can see that the reader is left hanging if the Flum-flumpeter Switch is "ON" and the Whoozy-Whatzit is in fact plugged in: there's a missing branch for "yes" off the "Is the Whoozy-Whatzit plugged in?" decision node. What then?

Don't feel like you have to cover every possibility to create effective documentation. While every branch you could take might be *theoretically* possible, there's a variety of reasons why you might choose to not include it:

- The probability of the failure is low (compared to other failures) or has never been observed.
- A desire to keep the documentation brief: maybe you only get a page or two for troubleshooting information in the back of a product manual and so you can only cover the most common failures types.
- The cost of researching a failure condition is prohibitive or would take too long.
- The information is geared towards a certain user group who are not likely to encounter the failure condition. For example, maybe you have a product that is installed in airplanes and boats and so you have separate troubleshooting trees for mariners and aviators.

From An Acorn To A Mighty Oak

If you are vigilant about updating it, over time a troubleshooting tree will represent the sum of all the knowledge you

have for a particular set of symptoms and causes. Back to the point made at the beginning: always be moving forward with your troubleshooting knowledge. If you've already solved the problem and haven't documented it, you're blowing it!

Creating and maintaining documentation is a necessary step for growing from "small" to "large." Freeing information in this way allows an entire team to support your systems. Around Discovery Mining, we used to joke about our "bus number." This is a morbid (and hilarious!) way to think about redundancy: how many people on your team could get plowed under by local public transit, and still life would go on? If the answer is nobody, you have a "bus number" of zero. Of course, people step away from their responsibilities for less dramatic reasons. They get sick, retire, receive a promotion, go on vacation, or leave to take a job somewhere else. Speaking of teams, documentation also allows you to introduce specialization by breaking your team into those that find the problems and those that fix them. Better yet, you can push your documentation all the way out to your customers and have them solve their own problems!

Even if you're troubleshooting alone with no team to support you, documentation like troubleshooting trees can be extremely useful. If you have to maintain a lot of systems, memorizing all the possible symptoms and remedies might be impossible. Also, documentation is a calming influence in a crisis. High-pressure situations favor the simplicity of following something like a troubleshooting tree, versus having to figure it out, again.

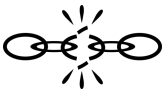
References:

- Header image: *"Black branches silhouetted against a sunset sky"*. Nareeta Martin, photographer. Retrieved from Unsplash, https://unsplash.com/photos/tFMdw0jCBPA.

Troubleshooting Trees was originally published February 26, 2013.

Notes:

Is It Really Fixed?

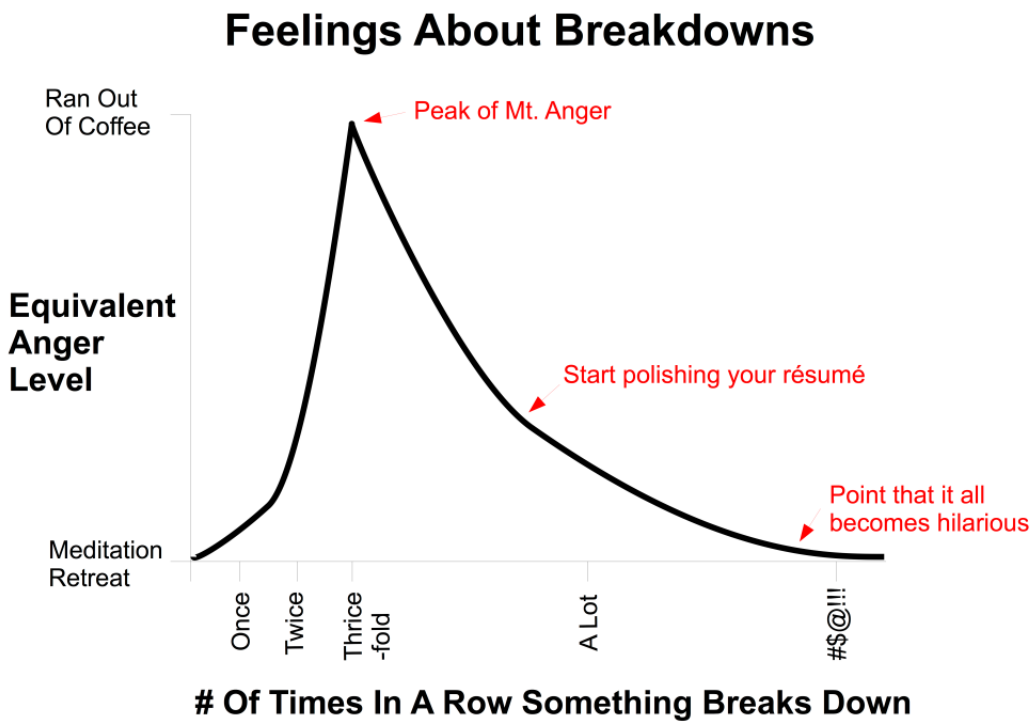


> Usually, if you repair or replace something and you haven't quite put your finger on the cause, it will fail again.
>
> **Rich Kral**

When I'm asked to help troubleshoot, I feel a sense of responsibility towards the future. After I'm done, I want it to be fixed. Forever. I suppose this stems from a desire to be competent and the pride I take in my work. Mostly though, it's that I don't ever want to be called back to fix something again. There's something humiliating about triumphantly telling someone "Well, it looks like my job here is done!," only to have them call you later…and tell you that the problem has reappeared. Going back that second time (or third, or fourth) to fix something again is humbling. Trust me, because I know from firsthand experience.

Even if there's no one around to witness your humiliation, the reality is that a botched repair can come back to haunt you. Especially if you're responsible for the smooth operation of a system (i.e., you are the one to be paged in the middle of the night when it fails), I recommend that you truly open your eyes and do everything you can to make sure the problem is fixed *for good*.

Here's what I've observed about how people react to repeated breakdowns:



Graph: people really lose it after the third failure.
(image: © Jason Maxham)

I have personally observed the arc of anger in the graph above: the first time something fails, most people will consider it a fluke. The second time will elicit concern. However, the third time is when people get angry and decide that someone needs to be held accountable: expect a visit from the CEO.

In general, the more a failure happens, the more that people will become desensitized to its effects. It's exhausting to maintain Peak Anger for very long. Set your expectations for people's reactions and keep your eyes on the prize: the long-term fix. You've got more time than you think between summiting Mt. Anger and the point where you should "start polishing your résumé."

Of course, it's preferable to not even start up the trail to Mt. Anger, so the following are strategies to prevent the embarrassing prospect (and related job insecurity) of fixing something only to have it break again.

Pull The Plug, Accelerate The Inevitable

In "Defaults and Reboots," I discussed the risk of turning a machine off: sometimes a system won't come back after a cold restart. While I mentioned that as a warning before, in this case we'll use it to our advantage to learn more about a machine. I call it the "Pull The Plug Test" and it's as simple as it sounds: after you make your repair, turn the machine off and then turn it back on again. In some cases, I would literally pull the plug on a device and see what happened when I plugged it back in again. You'll have to use your judgement on how far to take this: in some cases a hard reset like disconnecting a power cord might damage a machine. A gentler, but still effective, method is to shut down a machine using the steps recommended by the manufacturer. Either way, "pulling the plug" can be very illuminating: if a machine fails to work after being reset, it's better that you discover this on your own terms.

The Pull The Plug Test asks the question: "Will this system continue to operate correctly after some kind of restart?" Power failures aren't the only thing responsible for restarts, there are many others like scheduled maintenance, refilling supplies (e.g., restocking paper in a copier), human error, etc. An even more likely cause is the natural rhythm of your workplace: things are routinely turned on and off at the beginning of the day, when going to lunch, at the end of a shift, for the weekend, etc.

In those cases where a machine is on the edge of death, turning it on and off again may be the catalyst that finally

sends it to the graveyard. These end-of-life failures are often *independent* of the fix you just performed. That is, it was going to die anyway. Remember, the goal is to learn whether a machine you have finished repairing is fully functional and can be put back in service. If it's just a single power failure away from the scrap heap, you'll want to know. The point isn't to cause destruction, but to get out in front of a future problem. Put another way, would you rather see what is going to happen after a cold restart in a controlled manner, or at 3am on a random Tuesday?

Before executing the Pull The Plug Test, think a move ahead and consider the implications of turning off a machine. Complicated systems may have special restart sequences, so make sure this expertise is available before you get trigger happy with the on/off switch. Very high-risk restarts should be left to a maintenance window.

Spot Configuration Problems

For fixes that involve manipulating a machine's configuration, the Pull the Plug Test will expose whether your fix will be overridden by a restart. Take the example of a network router that you were called in to look at because data wasn't getting from point A to B. You easily spotted the problem, made the necessary configuration changes, verified that the data was flowing and then went on with your life. Two weeks later someone bumps the power cable, the device is reset, and the problem comes back! You go through the same troubleshooting cycle, come to the same conclusion as to the cause, and make the same change. What happened? Well, some devices will allow you to make configuration changes that are only valid for a limited period of time: while your login session is active, while the device is powered on, or until some automated process restores a default set of rules. In other words, you made a change to fix the problem, but you didn't make it *permanent*.

If the system in question cannot be made to automatically use a particular configuration upon restart, then you'll need to employ low-tech methods. For example, you can use tape and a marker to show where dials and levers need to be set. Or, have a laminated guide dangling next to the controls that says "TO RESTORE FUNCTIONALITY AFTER A POWER FAILURE, FOLLOW THESE STEPS…"

Before you grab the checkered flag and take a victory lap, make sure you've really fixed it…
(image: Jimmy_Joe / CC BY 2.0)

Other Paths To Permanence

A few more ways to prevent false fix-it victories:

- **Meet the standard:** making sure your repairs conform to a known standard increase their chances of sticking. Are you referring to a repair manual, design schematic or blueprints…or are you just winging it?
- **Ask the machine:** many machines have built-in diagnostics that'll tell you when they're sick and when they're healthy. Does the machine's self-reported state confirm the effectiveness of your repair?
- **Guilty until proven innocent:** be careful with the reuse of parts, especially if you use the shotgun method of

replacement. Bad components will lie in wait, ready to sabotage a future repair. When I worked on networks, sometimes we would replace a cable that we suspected was faulty. Later on, that same suspicious cable might end up back in our parts bin with all the other good cables. The funny thing about defective parts is that they can look exactly like working parts! I'll state the obvious: replacing a faulty part with another faulty part is a sure way to have a problem recur. Burned by the mixing of good and bad cables, I started cutting suspicious ones in half. That absolutely prevented their reuse. Later on, I got a cable tester and we would vet *any* cable, used or new, before using it as a replacement. Remember, "new" is not the same as "functional." That's why you'll see many manuals use the phrase "known working parts" when taking about replacements.

- **Testing and watching:** of course you'll confirm all your fixes by doing some rudimentary testing. It's even better to watch the person who reported the issue use a machine for real work after a repair. If possible, have them go through their *entire workflow,* top to bottom.
- **Automated detection and alerting:** the ultimate solution. Setting up a system to be continuously monitored and then to be notified automatically if there is a regression is the best way to be sure it's operating normally post-repair. The digital world really shines in this regard, but data collection interfaces can bring this level of certainty to analog processes as well.

Eat Your Dinner, Not Your Words

Perhaps the most important thing you can do is sprinkle in a dash of humility when discussing the results of your troubleshooting. Even if I'm sure I've nailed a fix, I prefer to let the results speak for themselves. If you still want that satisfying recognition of victory, be patient: like revenge, it's best served cold. Wait a few weeks, then ask how things are going with that repair you did. When they say everything is fine (and have paid the bill for the work), then go ahead and take your much-deserved victory lap.

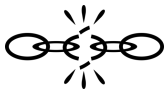
References:

- Header image: Palmer, A. T., photographer. *"Bog-downs" and bottlenecks in defense plant operation are out for the duration. Maintenance crews are eternally at it to keep everything in good working order.* United States, Ohio, Cuyahoga County, Cleveland. 1941. Dec. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017691079/.

Is It Really Fixed? was originally published February 28, 2013.

Notes:

Down To The Roots



> There are a thousand hacking at the branches of evil to one who is striking at the root.
>
> **Henry David Thoreau**

Machines don't exist in nature. You've never turned on the Discovery Channel to see a program called "Stalking the Wild Internal Combustion Engine in North Africa." Therefore, the root cause of *all* system failures originates with decisions made by a *human being* (or a group of people). Let that sink in. While I sympathize with the feelings of many an anti-social engineer, who would like nothing more than to build beautiful machines while being left alone, avoiding the human context will seriously impair your ability to be an effective troubleshooter.

Aftermath

The crisis is over. You fixed it, the client is happy, the production line is rolling again, the money spigot is flowing. You're done, right? Wrong. There's another step you should take that will take you from good to great: taking what you've learned from the incident and feeding it back into your organization. If you want things to improve and prevent failures from recurring, you need to make learning from your malfunctions a part of your process. Don't let it happen by accident, this "last mile" needs to be a priority and given adequate resources.

Get down here and find out what's really going on…
(image: Anna Levinzon / CC BY 2.0)

Let me introduce the field of Root Cause Analysis (RCA). Much ink has been spilled on this topic; you could fill an entire library with books about RCA. Beyond that, there are many management systems that integrate some aspects of RCA as part of a larger regime (GE's famous Six Sigma, for example). We won't get into all that, because this chapter is simply about making you aware of the need for something like RCA, rather than advocating for a particular system. It's up to you to choose and experiment with a particular one. Don't worry, an MBA isn't required nor will you have to spend months in boring seminars to benefit from RCA. Later on, I'll show you a very simple and powerful version of

RCA that may be all you ever need.

The Basics

First, let's go over the basics that should be included in an effective root cause analysis process. Learning from a failure requires these essential steps:

1. **Precisely defining the problem and explaining why it needs to be solved.** People need to understand why they're being asked to take time away from their responsibilities to participate. Defining the problem is its own challenge: I've seen this part of the process take considerable time. Resolving these disagreements is worth it, because they highlight the differing perspectives people bring to the process (e.g., customer service vs. engineering vs. legal). A good way to define a problem is in terms of its **external impact:** this deadline was missed, this client was inconvenienced, this person was injured, etc. Focusing on the *impact* of a problem adds emotional weight and provides the obvious reason for pursuing improvements. You'll need this energy because the recommendations stemming from a RCA investigation can have a large upfront cost. In other words, it's going to be *more work* for somebody.
2. **Gathering information: data, perspectives, assembling a timeline, quantifying the impact.** What happened? When? Where? How much? These details are crucial to identifying causes and will make sure you're fixing the right thing. I've had investigations end prematurely because of missing information: the only recommendation you can make in these type of cases is to start collecting data so you'll be prepared next time. Finally, quantifying the effects of an incident can be a powerful force to drive change: people not directly involved can be amazingly ignorant of the effects of a failure. You'll need to show them "how much" to make them care.
3. **Tracing the problem back to the root(s).** Not superficial proximate causes. Not "the car ran out of gas." If you're doing it right, the problem will always lead back to…you guessed it: people, processes, or policies. There can be no other answer. We already know that all machines will eventually break down; when that happens, they're just doing what's expected. Someone designed that machine, someone chose to deploy it for a given purpose, someone was responsible (or not) for maintaining it and on and on. The root cause of a failure is *never* within the machine itself.
4. **Generating possible solutions to prevent the failure from happening in the future.** I like to cast a wide net in this phase: the more options the better.
5. **Identifying your resources, then choosing a path.** The options you've generated will have varying costs and benefits, so you'll need to decide amongst them. Or, advocate for additional resources if you feel the problem is important enough.
6. **Follow-up: implement, measure, review, respond, evangelize.** This is the phase where many RCA projects stumble and die. Not only do you have to actually implement the recommendations coming out of the process (that is, do some *work*), you should also agree on a way to *measure* the effectiveness of your efforts. The data you collect for this purpose needs to be evaluated and responded to: be ready to change course if reality doesn't match your theory. Finally, spreading the good news is very important for ongoing participation by your co-workers. If you can *show* everyone that your RCA process is making their lives better, you won't have to bribe them with cookies to attend your meetings (although that helps too).

Enlist Allies

Having the backing of your organization is important before starting an RCA program. I guess you could go rogue and dish out RCA-based vigilante justice on your own: a geeky Batman armed with only a spreadsheet and the question "Why?" The problem is that your ability to implement what you learn from any RCA process will require resources. Therefore, be sure to enlist the help of someone with decision-making power. Your co-workers need to believe that This Is Important. RCA represents change, and people fear change. They will have to attend a meeting and you know that everyone loves attending meetings (although I personally think the 5 Whys process is fun). Finally, the recommendations will likely make more work for someone (at least in the short term). "Why are we doing this?," will inevitably be asked by someone. It'll either be done in the spirit of "My time is precious and so why should I give it to you?" or "You're cutting into my FaceBook time and so why should I give it to you?" Whether you work with people who take pride in their work or unapologetic slackers, you should have an answer that will appeal to the best in them (for the rest, you'll have to convince them the old-fashioned way: with carrots and sticks).

Where To Start

If you've got a reluctant manager, offer to start with just a single issue and see how it goes. There's no need to commit

to a full-blown RCA program before seeing the value with your own eyes. When deciding what issue to tackle first, consider starting small as well. As stated previously, the best topics are those that resulted in an impact that's easy to point to: something that harmed your customers, employees, the bottom line, or your reputation. If you're just starting to think critically about your operations, it's better to focus on bad things that have actually happened, versus hypothetical risks because:

1. If it happened once, it's likely to happen again. A problem that has already occurred has nominated itself from amongst all the other probabilities. Go with the flow and take this gift.
2. Nothing focuses the mind and rallies the troops like a threat to your organization. It also bestows upon you the moral high ground to make change.

Assemble The Players

There's always the question of how many people to involve in your RCA meetings. I'd aim for a "medium" amount of participation because there are two counteracting forces at work:

- **Why you want to include as many people as possible:** you'll need to have enough perspectives and information in the room so the problem can be understood and learning can happen. If you're missing too many key people, you'll be left with questions that can't be answered. You'll have frustrating moments where you'll ask a question and the room will be silent: "So–and–so knows, but he's not here…" If it's really important, you'll have to pause the meeting or suspend it until the right people are present.
- **Why you want to include as few people as possible:** among the "resources" you'll have to manage during any RCA meeting are your participants' interest and caring. If you have too many people in the room, you'll dilute this precious human commodity. There's a natural limit to the size of an effective meeting. As the number of people grows, the logistics of managing the attention span of a large group takes its toll. A smaller, energized team that is empowered to make improvements is preferable to a stadium-sized crowd.

So, you see the right answer is somewhere in between. A big enough group to know, and a small enough group to care.

Define The Problem

It may sound trivial, but sometimes defining the problem will be the biggest challenge for a RCA meeting.

Scope: problems defined too narrowly and not in the context of larger meaning to your organization will lack the psychological punch to get action. "Drive shaft with serial number #2319310 failed at 3:45pm in Chilling Unit #5" may be accurate, but won't inspire action. Back to the point above about defining the problem in terms of its impact to your organization.

Problems defined too broadly may prevent the team from taking any action at all. Things like "our clients are unhappy" or "our industry is collapsing" are too amorphous. Those kind of large-scale problems can be fascinating to debate, but your group will likely lack the resources to pursue a solution. Also, people will tune out if they think the problem is so large that they will be unable to make a difference.

Perspective: if you bring together people from different teams, you may encounter a subtle form of finger pointing. People can be parochial in viewing a problem only from their perspective. This isn't bad, it's just not always useful. Framing the problem in the spirit of blame (e.g., "the maintenance department screwed up") may contain kernels of truth, but isn't a perspective that will lead to the best insights.

The proper perspective for framing a problem is external (i.e, the customer's). Everyone will share a basic level of empathy for this view because their job depends on having customers, and so it places all participants on the same side. Soliciting input from a broad cross-section of people involved in an incident ensures exposure to all the relevant viewpoints. I've done 5 Whys by myself, but it's much more powerful in a group setting because of this cross-pollination of perspectives.

Little things can make a big difference…
(image: Delwin Steven Campbell / CC BY 2.0)

5 Whys: Tracing Consequences Back To Roots

There's a famous proverb about a nail:

> **For Want of a Nail**
>
> For want of a nail the shoe was lost.
> For want of a shoe the horse was lost.
> For want of a horse the rider was lost.
> For want of a rider the message was lost.
> For want of a message the battle was lost.
> For want of a battle the kingdom was lost.
> And all for the want of a horseshoe nail.

In real life, the causal chain isn't usually this poetic, but I think you get the point: seemingly small things can have large consequences. "For Want of a Nail" is a great segue to the 5 Whys method of root cause analysis, which not only shows the impact of small causes, but also uncovers hidden ones.

The 5 Whys process was developed by Sakichi Toyoda and helped Toyota Motor Corporation become the largest automotive company in the world. Taiichi Ohno, the architect of Toyota's Just-in-Time production system, describes 5 Whys like this:

> When confronted with a problem, have you ever stopped and asked why five times? It is difficult to do even though it sounds easy. For example, suppose a machine stopped functioning:
>
> 1. Why did the machine stop?
> There was an overload and the fuse blew.
> 2. Why was there an overload?
> The bearing was not sufficiently lubricated.
> 3. Why was it not lubricated sufficiently?
> The lubrication pump was not pumping sufficiently.

> 4. Why was it not pumping sufficiently?
> The shaft of the pump was worn and rattling.
> 5. Why was the shaft worn out?
> There was no strainer attached and metal scrap got in.
>
> Repeating why five times, like this, can help uncover the root problem and correct it. If this procedure were not carried through, one might simply replace the fuse or the pump shaft. In that case, the problem would recur within a few months.
>
> To tell the truth, the Toyota production system has been built on the practice and evolution of this scientific approach. By asking why five times and answering it each time, we can get to the real cause of the problem, which is often hidden behind more obvious symptoms.
>
> **Taiichi Ohno, Toyota Production System: Beyond Large-Scale Production** [1]

The 5 Whys process invites you to look below the surface of "proximate" causes, because acting at this level rarely leads to lasting improvements. It may be satisfying to ask "Why won't the car start?," and answer, "Because the battery is dead." However, stopping at the proximate cause will lead you to believe that inanimate objects have minds of their own. "It's the battery's fault!" would be the inevitable conclusion of this kind of thinking, but of course yelling at batteries like a madman won't change anything. You might chuckle at that imagery, but any time a machine is blamed for a failure it's the same thing.

If people are the source of all system breakdowns, it's reassuring to know that they are also the solution. The entire theme of this book, that all machine problems are human problems, struck me like lightening during a 5 Whys meeting.

The Washington Monument Story

Let's look at another example of hidden causes uncovered by the 5 Whys method, in this oft-told story about the Washington Monument*:

> **Problem:** the Washington Monument was crumbling.
>
> **Why** was it crumbling? Because harsh chemicals were being used on the monument.
> **Why** were the harsh chemicals being used? To clean off all the pigeon poop.
> **Why** were there so many pigeons? The pigeons were attracted to the spiders (pigeons eat spiders).
> **Why** were there so many spiders? The spiders were attracted to the gnats (spiders eat gnats).
> **Why** were there so many gnats? The gnats were attracted to the artificial lights turned on at dusk.
>
> **Solution:** Turn on the lights at a later time. That would attract fewer gnats, which would mean fewer spiders, which would mean fewer pigeons, ultimately lessening the need to use those harsh chemicals. Problem solved!

You can see here that we asked "Why?" five times in a row and came up with an astounding result! Who would have thought that turning the lights on at dusk would lead to a crumbling monument? Also, seeing the full causal chain that led to the problem gives you many possibilities for a solution. Let's say that turning on the lights later was not an option for whatever reason (maybe you need them on for safety concerns). You could, separately or in combination, consider reducing the amount of gnats, spiders, or pigeons. This is in addition to fixing the problem at the proximate cause level by finding a pigeon poop cleaner that isn't corrosive. It's good to have options!

*Note: This story of the Washington Monument convincingly shows the power of 5 Whys in discovering the root cause of a problem. However, I have been unable to confirm that it's a true story. Based on my research, the story is widely cited all over the Internet. It truly is the king of examples for Root Cause Analysis and you'll find it in many a PowerPoint presentation. The problem is, I couldn't find a single reference to an original source verifying that the story is real. I've posted a question on Quora asking about the story's origin , but until someone comes forth with evidence of its veracity, I'd treat it as a very good myth of Management Consulting.

Update (April 14, 2015): Joel Gross has done some intriguing research that sheds light on the origin of this mystery. In "5 Whys Folklore: The Truth Behind a Monumental Mystery," he makes the case that the Washington Monument story has its origins in an unpublished National Park

Service report by Dr. Don Messersmith ("*Lincoln Memorial Lighting and Midge Study*"). While sharing some elements with the real account, I'd say the popular parable is more legend than fact. However, several intriguing questions remain to be answered. First off, how did we get from that unpublished NPS report to the urban legend we have today? More importantly, should we stop using the Washington Monument story to introduce people to 5 Whys? As a teaching tool, I've found that this drama of causality has that "aha!" factor. People instantly get the message that root causes are often buried and require probing to uncover, as well as the power of unintended consequences.

The Washington Monument story is a beautiful illustration of the problem of hidden causes. There, there, I'll get you a tissue to dry your eyes.
(image: Phil Roeder / CC BY 2.0)

Trace The Causes

To conduct a 5 Whys meeting, it's best to use a large whiteboard to note the progress of your investigation. Start by writing down the problem on the left-hand side of the board. From there, you'll proceed through the 5 levels of whys,

branching off as appropriate for places where there were multiple causes. When that happens, be sure to get all of them before you move on to the next level of "Why?" Don't get fixated on the number 5 either, there's nothing magical about that 5th Why. In practice, I've found that root causes can appear as soon as the 2nd or 3rd Why. Don't go beyond that if it's not useful. You'll know when you've reached the end of a line of inquiry because the next "Why?" would result in a question that's silly or obvious. We once played "100 Whys" and ended up with "Why do people have to eat?"

Let's look at an example of a 5 Whys inquiry in a table format. The problem is on the left and the 5 levels of "Whys?" take you to possible solutions on the right:

Problem	**1st Why**	**2nd Why**	**3rd Why**	**4th Why**	**5th Why**	**Solution**
Our client, XYZ Inc., is angry and is considering canceling their contract with us.	The delivery deadline for our client XYZ Inc. was missed by 7 days.	Water damaged the shipment and it had to be replaced.	A water reservoir filled up and spilled onto the factory floor.	Bob wasn't at the controls to flush the reservoir as it became full.	Bob has other work duties he's required to perform.	Increase staffing in general or deploy extra staff when the automated flushing mechanism breaks.
Our client, XYZ Inc., is angry and is considering canceling their contract with us.	The delivery deadline for our client XYZ Inc. was missed by 7 days.	Water damaged the shipment and it had to be replaced.	A water reservoir filled up and spilled onto the factory floor.	Variable fill rate means employees aren't aware of how much water the reservoir is currently holding.	No alerting system in place to let staff know when overfilling is imminent.	Install reservoir alerting system.
Our client, XYZ Inc., is angry and is considering canceling their contract with us.	The delivery deadline for our client XYZ Inc. was missed by 7 days.	Water damaged the shipment and it had to be replaced.	A water reservoir filled up and spilled onto the factory floor.	The reservoir needed to be manually drained.	Automated flushing mechanism has been broken for the past 2 weeks.	Automated flushing mechanism should be repaired immediately when it breaks.
Our client, XYZ Inc., is angry and is considering canceling their contract with us.	The delivery deadline for our client XYZ Inc. was missed by 7 days.	Water damaged the shipment and it had to be replaced.	There's no physical separator between the shipping staging area and the water containment system.	Designers didn't anticipate the installation of the reservoir near the staging area.	Operations department didn't communicate all probable uses of the space to the architects.	Install a barrier to contain flooding. Also, operations needs to formally document its planned use for a space when starting a new construction project.

Our client, XYZ Inc., is angry and is considering canceling their contract with us.	XYZ Inc. wasn't contacted about the delay until 5 days after the deadline was missed (they could have worked around the delay if they had been informed).	The customer service department has no visibility into shipping delays.	→	→	→	Report shipping delays to the customer service department so clients can be contacted.

You can see that we've started with a negative external effect: a client is angry with us. From there, we've identified two 1st-level Whys: a shipping delay caused by flooding and a client communication problem. Already we've learned something, either getting the shipment out on time *or* letting the client know about the delay could have saved our bacon! This is why it's so important to begin the 5 Whys process with the overarching Bad Thing: if you would have started with the flooding, you would have missed this other aspect of the problem. Also, look at all the options we have generated to prevent the problem in the future. We can attack the problem through: staffing, an alerting system, faster-responding maintenance, installing a flood barrier, or better client communication. If losing XYZ Inc.'s business is something that absolutely cannot be allowed to happen, you might choose to make all of these improvements! Most of the time, however, you won't be able to pursue all your good ideas: you'll need to do a cost/benefit analysis to select among them. I've been in 5 Whys meetings that have resulted in a long list of great possibilities for improvements, but resources only allowed us to pursue the best one.

It All Comes Back To Us

The end result of 5 Whys will typically be the *lack* of something: discipline (i.e, following proper procedures), training, knowledge, resources, setting of expectations, maintenance, etc. If you find yourself still describing the goings-on of inanimate objects, you haven't gone deep enough. Keep asking "Why?" How the problem unfolded via cause and effect is interesting, but describing the situation like a news reporter is only the beginning of the process. Your systems were designed by people, purchased by people, installed by people, maintained by people and their output sold to other people (clients or customers). A long series of choices made by *people* is the result of any incident. Recognizing the role of human choice in system failures is different from placing blame: the people who made the choices leading to an incident were probably acting on the best information available to them at the time. Simply recognize that new information has come to light and needs to be integrated into your processes.

The importance of management in drawing the right conclusions from any kind of root cause analysis cannot be understated. If you accept that the origin of all incidents was somebody's choice, you quickly realize that the "top chooser" (the CEO, owner, Board of Directors, etc.) is ultimately responsible but, as stated before, not necessarily to *blame*.

A Touchy Subject

Since you're bound to name people as causes when you do 5 Whys, you'll need to conduct the process with tact. In the example above, we can see that Bob could have saved the day if he was at the controls to prevent the reservoir from overfilling. People love a scapegoat, so there will be the temptation to ignore the rest of the evidence and conclude that Bob was the root cause. However, don't be fooled: someone hired Bob, put him in a position of responsibility, overloaded him with work, and designed a system that requires constant human supervision to work properly. That changes your perspective on the true cause of this incident, doesn't it?

If Bob was truly negligent with respect to his job duties, then perhaps he should be fired. However, if you believe that replacing Bob will stop this type of accident ever happening again, think twice. Bob's contribution to this incident

didn't happen in vacuum: his employer created a situation that ensured maximum damage when there was a human error. People are guaranteed to make mistakes, so any system where you are just one lapse away from a catastrophe is poorly designed. A combination of *multiple* backstops, human and automated, will ensure that something like this doesn't happen again. If you look at the outcome of this 5 Whys investigation, you'll see that both types of fixes emerged as ideas for future improvements.

There will be cases where the right answer to an incident will be a change of personnel. When that happens, do a double–check and make sure you have fostered the conditions for your employees to be successful.

Empathy For Those Affected

An advocate for the client should always be present at your 5 Whys meeting. This "client advocate" could be a project manager, the salesperson who owns the account of an affected client, the person who collects the bills, or someone from a different department who is the "customer" (in the case of internal projects). The advocate will bring the client's point of view to the process and, through their presence, the client will have "a seat at the table." The advocate will see first-hand that you care about improving and rectifying what went wrong. Salespeople especially will feel better after witnessing your efforts to improve a bad situation. In the opposite direction, I've seen salespeople develop empathy for the challenges faced by the engineers who are tasked to fix a problem. This can counteract the classic tension between those who sell promises and those who deliver on them. The salesperson who is included in the RCA process will be more vigilant about not over-promising to clients. Also, what great fodder for their next sales pitch, when they can explain what went wrong and give a detailed rundown on all the improvements you've made!

Some Cheese With Your Whine?

At the end of a 5 Whys meeting, it's useful to ask for feedback on the solutions that were generated during the session. The participants of your meeting can point out flaws and prevent you from pursuing fixes that embody the old saying, "The cure is worse than the disease." Also, people get real smart when you propose creating more work for them: and when I say "smart," I mean whiney. It's an art to tell the difference between principled objections and the "don't make me change anything" type of grousing. I like to call bluffs and ask "What would you have to know or see to be persuaded this is the right fix?" and then design an experiment or collect data to satisfy their concerns. If you're right, there's no way they can object: they told you exactly what they needed to be convinced.

Make It A Habit

Root cause analysis should be done on a regular basis. Depending on your needs, it should be on the calendar for the same time every day/week/month/quarter. Don't have any problems in your organization that are worthy of learning more about? Contact me, I'd like to invest. For the rest, strap in for a lifetime of RCA. Having RCA occur at regular intervals means that sometimes you'll be focusing on minor problems. That's fine, have the meeting anyway. As a manager, I was stunned at the things I learned about my company from digging into even the "smallest" of problems—and I had been there from day one! Remember too about the "tip of the iceberg": little problems sometimes develop into larger problems. Ruthlessly examining your small incidents may prevent you from ever having to experience a large disaster.

Too Much Of A Good Thing

If your organization is constantly in "crisis mode," you might have enough material for a thousand 5 Whys meetings. You can schedule extra sessions, but be aware of the law of diminishing returns. Taking employees out of their daily routine is costly, and the recommendations from each investigation, while typically having long term benefits, will involve w-o-r-k. RCA is a great way to transform your workplace, but even the most nimble and well-managed organization will have an upper limit to how much change they can absorb at any given time.

Criticisms Of 5 Whys

No process is perfect. Let's look at some problems you may encounter when using 5 Whys and how I think you can overcome them. Stewart Anderson, in his article "Root Cause Analysis: Addressing Some Limitations of the 5 Whys," lays out the major pitfalls:

> While many companies have successfully used the 5 Whys, the method has some inherent limitations. First, using

> 5 Whys doesn't always lead to root cause identification when the cause is unknown. That is, if the cause is unknown to the person doing the problem solving, using 5 Whys may not lead to any meaningful answers. Second, an assumption underlying 5 Whys is that each presenting symptom has only one sufficient cause. This is not always the case and a 5 Whys analysis may not reveal jointly sufficient causes that explain a symptom. Third, the success of 5 Whys is to some degree contingent upon the skill with which the method is applied; if even one Why has a bad or meaningless answer, the whole procedure can be thrown off. Finally, the method isn't necessarily repeatable; three different people applying 5 Whys to the same problem may come up with three totally different answers.
>
> Other drawbacks to 5 Whys have been cited, including the method's inability to distinguish between causal factors and root causes, and the lack of rigor where users aren't required to test for sufficiency the root causes generated by the method.
>
> **Stewart Anderson, "Root Cause Analysis: Addressing Some Limitations of the 5 Whys"** [2]

It's true that 5 Whys process won't, by itself, reveal an unknown cause to you. Anderson's criticisms are fair, but for me 5 Whys is just a framework for learning *more* about an incident. If you have a diverse set of people participating in your 5 Whys process, you'll at least capture all the knowledge available in your organization. That's frequently more than enough to solve most problems. Also, I don't know of any process that guarantees revelations of causes unknown. Anderson is right that results are contingent upon "the skill with which the method is applied," but this objection would seem to apply to all systems designed to acquire knowledge. Even our vaunted Scientific Method benefits from a skilled and disciplined practitioner who can adhere to the principles and draw the right conclusions from an experiment.

For those unknowns that can't be discovered by talking it through, a perfectly acceptable outcome of a 5 Whys meeting is the realization that you need more information to complete your analysis. You may resolve to start collecting data, set up a monitoring system, or conduct an experiment to test a theory about the cause. I've suspended 5 Whys meetings to allow for additional time to gather information we identified as relevant. This was especially true for situations where we'd have an incident and no one could answer the question "What is normal?" Reconvening later, after a chance to acquire the missing information, we were better able to tackle the problem.

If you get stuck, don't be afraid to put a big question mark as a placeholder in your 5 Whys tree and move on with your meeting. Sometimes the answer will reveal itself later. Even if it doesn't, in those situations with multiple root causes, you can still fill in the other branches. In the example above, even if the cause of the flooding was unknown and needed further investigation, we still had the opportunity to discover the client communication problem.

You can read the rest of Stewart's article if you're interested in learning more about improving the 5 Whys process. I hope you eventually reach a level of sophistication where those suggestions are useful, or perhaps you'll branch out into other systems. In the meantime, 5 Whys remains a very accessible entry point to starting your RCA journey.

Beyond Solving Problems

The benefits of RCA go beyond learning about any one incident and spill over into improved morale. Having a good RCA program sends its own message:

- You take improvement seriously.
- You care enough about your product or service that major mistakes and incidents will be thoroughly reviewed (i.e., problems will be brought out in the open, not swept under the rug).
- You value your employees: you seek and act upon their feedback while making them integral to the improvement process.

References:

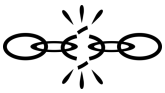
- Header image: *Uprooted tree at Elizabeth A. Morton National Wildlife Refuge (NY)* . November 1, 2012. U.S. Fish and Wildlife Service Northeast Region. Retrieved from Flickr, https://www.flickr.com/photos/usfwsnortheast/8152103662/.
- [1] Taiichi Ohno, *Toyota Production System: Beyond Large-Scale Production* (Portland: Productivity Press, 1988), pg. 17.

- [2] Stewart Anderson, "Root Cause Analysis: Addressing Some Limitations of the 5 Whys," *Quality Digest*, December 17, 2009.

Down To The Roots was originally published March 14, 2013.

Notes:

Moral Authority



> I used to have people say “If it was yours, would you fix it?” I’d say, “No, I wouldn’t. But, if my wife was driving it I would.”
>
> **Dan McCormick**

Once you get familiar with a system, you start to get a good sense of what could push it over the edge. As a troubleshooter, there will be times that you labor in vain to alert your colleagues about the dangers of a meltdown. It’s human nature to ignore warnings for things which we have little or no experience. Many times your fate will be like a modern day Laocoön, who tried to warn the Trojans about the giant wooden horse/metaphor they had just wheeled into their stronghold. He famously said: “Beware of Greeks bearing gifts.” Actually, I hope you fare a little better than good ol’ Laocoön, who was blinded and then strangled by two giant sea serpents (courtesy of Athena) as repayment for his whistle-blowing efforts. Unfortunately, he was deprived of a righteous “I told you so!” moment at the end of that whole saga.

Unlike Laocoön, maybe you didn’t see disaster coming and were blindsided like everyone else. Either way, in the wake of a crisis, people will be ready to listen to your plans for improvement. Because they will be the most receptive to change after a catastrophe, you must take advantage of this limited window. Act while the pain of a crisis is fresh in people’s memories. This quote sums it up quite well:

> You never want a serious crisis to go to waste. And what I mean by that is an opportunity to do things you think you could not do before.
>
> **Rahm Emanuel, former White House Chief of Staff**

In a political context, I'm suspicious of the sentiment contained in a statement such as this. When the government does things it "thought it couldn't do before"—watch out! You might've heard the old saying, "No man's life, liberty, or property are safe while the Legislature is in session."

That aside, I must admit that Mr. Emanuel would make a mighty fine troubleshooter. He's right, you shouldn't let a crisis go to waste. Afterwards (or during, if it persists long enough), you'll have the opportunity to make changes that would have been difficult without the undeniable reality of a recent catastrophe. In the aftermath of a crisis, here are some areas where you should boldly forge ahead:

- **Budgets:** risks you identify have an "average" expected cost. When an accident finally does happen, that cost is fully realized. Whereas before, the probabilities and expenses were once hypothetical, afterwards you will have the *actual* numbers to point at. The resulting expenses are a great starting point to advocate for new spending to prevent an accident from reoccurring: new machines, increased staffing, more frequent maintenance, infrastructure improvements, etc.
- **Policies:** new gear isn't the only way to prevent an accident from reoccurring. Instead, what you might need are changes to how your organization works. Training and procedures should be reviewed and updated to deal with the new reality. Because people like routine and are resistant to change, or because of organizational politics, this may be your *only* opportunity to make meaningful policy changes.
- **Information flow/transparency:** sometimes a lack of information is the origin of a disaster. As a consequence of territorial or political disputes, departments and processes can become closed off from each other. In times of peace, vital information might be painful to assemble because of this organizational friction. However, in the wake of a calamity, demands for transparency will be taken more seriously.

The Psychology Of Disasters

Why don't we do a better job anticipating disasters? It probably has to do with:

- **Reality is complicated:** poorly understood systems, interacting with other poorly understood systems, can produce even more things you don't understand. Also, there's the power of unintended consequences: your actions can set things in motion that may only be understood with the power of hindsight.
- **"Availability heuristic":** psychology studies have shown that your mind places additional importance on things that have happened recently. This effect is called the "availability heuristic" and the mental bias also applies if you're able to think of an example of something. Disasters, being rare, won't likely trip this "availability" trigger–you'll have to champion their importance without this psychological boost.
- **"Bystander effect":** group dynamics come into play in disaster preparedness psychology. If no one else is concerned, you'll be more likely to brush off the risk as well.
- **Personal or institutional solipsism:** this is an inability to project beyond your own experience (or your organization's). It's the sentiment that, if it hasn't happened to *you* (or *us*), it cannot happen at all.

Data collection (prediction) and routine maintenance (prevention) can be a bulwark against catastrophes. But, let's be realistic, eliminating all future disasters is like trying to stop the rain. This is *not* an argument for complacency, but rather a reminder to prevent accidents when you can and to learn the most from the rest that will inevitably slip through your best defenses.

When you have the moral high ground, the good people will listen.
(image: Ludwig Von Langenmantel / Wikimedia Commons)

"With Great Power, Comes Great Responsibility"

Invoke the power of Moral Authority sparingly and with the restraint of a wise, old sage like Mr. Miyagi. The goal is to pave the way for a better future, not to wield power for its own sake. The phenomenon of unintended consequences, which underlies many disasters, can also befall plans to prevent disasters. Using the emotion of a disaster to take resources and attention away from other worthy causes has its own cost, and can lead to problems just the same.

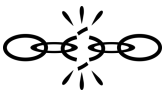
References:

- Header image: Harris & Ewing, photographer. *Podium/desk on porch of U.S. Capitol, Washington, D.C.* [Between 1921 and 1923] [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016885695/.

Moral Authority was originally published March 27, 2013.

Notes:

Making A List, Checking It Off



> It's such a hard lesson to learn, but you need to go all the way through the basics.
>
> **Jamie Karrick**

In the relatively short span between the start of the Industrial Revolution and now, the amount of knowledge accumulated in just a single industry would take many lifetimes to fully understand. Imagine trying to know everything there is about cars or computers or petroleum extraction. These industries haven't been around that long, but the people associated with them have been very busy!

On top of that, many machines *combine* innovations from a variety of sources. Think about a modern airliner, which is an amalgamation of the latest-and-greatest from a large number of other industries: high-efficiency turbine engines, lightweight composite materials, advanced computer systems for communication and navigation, etc. Within a jet is an endless list of technological advances, all cobbled together to create one awesome super-system. "Systems of systems" like this have long surpassed the ability of any one human to fully understand them. Yet, by finding ways to manage this complexity, we are still able to use technology to better our lives.

Without further ado, let me introduce the humble checklist. Within its boxes contains a method for handling complexity that is so effective (and simple) that large portions of our advanced industrial civilization would be impossible without it. At the same time, I also want to familiarize you with a great book on the subject that shows just how much we can gain from using checklists (and why). Creating a checklist is the ultimate way to communicate the

results of what you've learned from a troubleshooting exercise. As opposed to other passively consumed forms of post-repair communication like "incident reports" and "service bulletins," the checklist puts the best way of doing something directly into the hands of those doing the work. As you'll see, it's not enough to *know*: if facts aren't translated into *doing* on a consistent basis, your learning is meaningless. Because the checklist distills information into an action-oriented form, I shed a lone tear when contemplating the beauty of its bridge between what is known and what should be done about that knowing.

This guy knows his way around a checklist.
(image: Bart Fields / CC BY 2.0)

So Many Things Needed To Go Right…And Luckily They Did

Atul Gawande's *The Checklist Manifesto* is a call to arms for people who want to improve the world. The book begins with a compelling story showing the awesome power of advanced medical technology. Gawande recounts the details of a 3-year-old who fell into an ice-covered fishpond in the Austrian Alps. The little girl sat at the bottom of a small lake for 30 minutes, in icy cold waters, before her parents located her and pulled her out. The fight to save her life began and what unfolded can only be described as a miracle of modern medicine: a well-choreographed dance between dozens of personnel and the advanced technology they used. These professionals managed to give this young girl a second chance at life, but so much had to go right for that to happen:

> To save this one child, scores of people had to carry out thousands of steps correctly: placing the heart-pump tubing into her without letting in air bubbles; maintaining the sterility of her line, her open chest, the exposed fluid in her brain; keeping a temperamental battery of machines up and running. The degree of difficulty in any one of these steps is substantial. Then you must add the difficulties of orchestrating them in the right sequence, with nothing dropped…
>
> **Atul Gawande, The Checklist Manifesto** [1]

Now, this little girl is living a normal life. However,

> For every drowned and pulseless child rescued, there are scores more who don't make it—and not just because

> their bodies are too far gone. Machines break down; a team can't get moving fast enough; someone fails to wash his hands and an infection takes hold.
>
> **Atul Gawande, The Checklist Manifesto** [1]

"Too Much Plane" For One Person?

As a technical field or industry matures, it gets exponentially more complicated. Knowledge accumulates indefinitely as innovations occur (i.e, there are always new things to keep up with). Also, the number of applications for an industry's technology tends to grow over time. For example, microchips were used in just a few applications in the 1950s, but today they are everywhere. The first internal combustion engines were deployed in cars, but today you can find them in a wide variety of uses, from lawnmowers to portable electric generators. As a technology finds new applications, the intersections with each new industry creates even more complexities. The principles of the internal-combustion engine may be a known quantity, but its use in a car, an airplane, and a lawnmower are all different and have special considerations.

The tragedy from which the modern use of the checklist was born. This Boeing Model 299, piloted by Major Ployer P. Hill, crashed during a demonstration flight at Wright Field on October 30, 1935.
(image: National Museum of the US Air Force)

For aviation, this same march of technological progress resulted in increasingly sophisticated designs. Enter stage right, the Boeing Model 299: a prototype bomber that was all set to become the go-to plane for the US Army Air Corps in 1935. The result of a design competition, the 299 trounced the offering by rival Martin & Douglas in all of the important categories: range, payload capacity, and speed. However, during a demonstration in front of the Army's top brass, there was a terrible accident. Major Ployer P. Hill, the Air Corps' chief of flight testing, was in command during the demonstration flight of the Model 299. After takeoff, he climbed the plane to 300 feet, where it stalled, turned towards the ground, and exploded in a giant fireball on impact. Tragically, two of the five crew members aboard died in the accident, including Major Hill.

A demonstration flight with fatalities that ends in a fiery explosion in front of the press: that isn't just a tragedy, it's also bad marketing. To add to the woes, the Model 299 was in the midst of a design competition and the destruction of the prototype meant Boeing was disqualified. The Army ended up choosing Martin & Douglas' design and only a few 299's were initially ordered. However, what happened next changed aviation forever:

> Still, the Air Corps faced arguments that the aircraft was too big to handle. The Air Corps, however, properly

> recognized that the limiting factor here was human memory, not the aircraft's size or complexity. To avoid another accident, Air Corps personnel developed checklists the crew would follow for takeoff, flight, before landing, and after landing. The idea was so simple, and so effective, that the checklist was to become the future norm for aircraft operations. The basic concept had already been around for decades, and was in scattered use in aviation worldwide, but it took the Model 299 crash to institutionalize its use.
>
> **"The Checklist", Air Force Magazine** [2]

This point is important and bears dwelling on for a moment: if someone had to die at the controls of the 299, it was actually a blessing that it happened to be Major Hill. Because his death was difficult to ignore, it wouldn't be in vain; a rookie crashing the plane would have been easy to dismiss as a lack of experience. You can just imagine the conclusion of that hypothetical incident report: "…we recommend more training: only experienced pilots should be flying complex airplanes like the 299." Thank goodness those test pilots dealt so honestly with the fact that Major Hill was an expert and didn't simply recommend more preparation.

Gawande notes that the adoption of the checklist came when the aviation industry's designs had reached a tipping point in terms of complexity. In addition to that, I feel there are some special aspects of aviation that made this leap inevitable. Flying airplanes is an endeavor where human failures are made brutally plain because their effects are so deadly. A person sits conspicuously at the controls, making the link between erroneous actions and bad outcomes readily apparent: either a safe landing and a "see you tomorrow," or a massive fireball with coverage on the 10 o'clock news.

All human action is worthy of our scrutiny for betterment but, in other fields, the link between human error and bad outcomes may not be as readily apparent as it is in aviation. Deaths from airplane crashes aren't any more tragic than those stemming from other kinds of poor judgement, yet I think we may be willing to give a "free pass" to mistakes in other fields. Evidence for this is the average number of annual aviation-related deaths: they have never exceeded more than 5,000 in any given year worldwide (and for most years it's around 1,000). Contrast that to what Gawande found about the impact of human error with regards to surgery:

> Avoidable surgical complications thus account for a large proportion of preventable medical injuries and deaths globally. Adverse events have been estimated to affect 3–16% of all hospitalized patients, and more than half of such events are known to be preventable. Despite dramatic improvements in surgical safety knowledge, at least half of the events occur during surgical care. Assuming a 3% perioperative adverse event rate and a 0.5% mortality rate globally, almost 7 million surgical patients would suffer significant complications each year, 1 million of whom would die during or immediately after surgery.
>
> **"WHO Guidelines for Safe Surgery"** [3]

Those are unbelievable numbers. You can see from these statistics that the opportunity to improve our lives by reducing the impact of human error is substantial. That sets up the narrative arc of *The Checklist Manifesto*: taking the checklist's successes in other fields (aviation, construction, etc.) and applying them to medicine. Shortly, we'll do likewise and apply the checklist to troubleshooting.

Coming To Terms With Human Fallibility

It's clear that Major Hill's death forced his fellow test pilots to make a tough realization. When the most respected, talented, experienced, and trained person dies attempting a task, where does that leave the rest of us? The implications for the average person, trying to do the same job, aren't good. Cue the scary music.

When we're talking about machines like airplanes or respirators, a single detail can be the difference between life and death. Or, less dramatically for other machines, between working and not working. That reminds me, I didn't tell you the cause of the crash of Major Hill's Boeing 299. I wish I could say it was some exotic failure, but the reason the plane crashed was easily preventable. On most airplanes, there is a system to lock the control surfaces (rudder, ailerons, and elevator) when not in use. This mechanism is called a "gust lock" and it prevents these all-important parts from being damaged by the wind when sitting on the ground. Of course, the ability to freely move the rudder, ailerons, and

elevator is vital to controlling the airplane when in flight, so the locks must be disengaged before takeoff. Unfortunately, this critical step was missed:

> From the evidence submitted the Board reached the conclusion that the elevator was locked in the first hole of the quadrant on the "up elevator" side when the airplane took off, for had the elevator been in either of the "down elevator" holes on the quadrant or the extreme "up elevator" hole, it would have been impossible for the airplane to be taken off in the former case, and in the latter case the pilot could not have gotten into the seat without first releasing the controls. With the elevator in this position they are inclined at an angle of 12.5 degrees.
>
> During the take-off run the airplane could not assume an angle of attack greater than the landing angle of the airplane (7.5 degrees) plus the angle of incidence of the monoplane wing to the fuselage (3 degrees) or a total angle of 10.5 degrees. This would not be particularly noticeable to the pilot during the ground run.
>
> However, as soon as the airplane left the ground, which several witnesses testified was in a tail low attitude, the elevators, with increasing power, varying as the square of the air speed (approximately 74 miles per hour at take-off), tended constantly to increase the angle of attack, until the stall was reached. The trim tab on the elevator also tended to aggravate this extreme tail heavy position, since with locked elevators, and the pilot pushing forward on the control column, the trim tabs were up, and themselves acted as small elevators on the fixed elevator proper.
>
> **National Museum of the US Air Force, Model 299 Crash Fact Sheet** [4]

We'll never know for sure why the gust locks were left engaged on that fateful morning of October 30, 1935. Was Major Hill nervous about the demonstration flight? Was he distracted at that key moment in his routine when he normally disengaged them? Or, perhaps the thought just never crossed his mind (or his co-pilot's) that day, a forgetfulness of which we've all been guilty.

The rub is, for those *critical* steps (like disengaging gust locks), the airplane has to be set up the right way, all the time. That one time you forget will be your last. Luckily, most missed steps don't result in death, but that doesn't lessen the number of things that need to be gotten right to make our technological civilization chug along every day.

Managing Complexity

We may have evolved with the natural world, but it still has mysteries that our reason hasn't penetrated. Even our best mathematical models, running on massive supercomputers, can only predict the weather 6 days out. On top of that, we've been busy creating things that rival our natural environment in complexity. However, the specific ability to fly airplanes, manage a team building a skyscraper, or perform surgery simply weren't a selective factor in our evolutionary history. There was no equivalent to "perfectly execute these 30 steps in the same sequence every time" as our species was grappling with surviving in the wilderness over the millennia. On the savannah, if 747s and computers had been ubiquitous as Woolly Mammoths and Saber-toothed Tigers, there'd probably be no need for the checklist! Forgetting to disengage the gust locks on an airplane puts you in the same amount of danger as getting too close to a steep ledge or staring a lion in the face, but it won't automatically invoke a primal fear-based response in the same way. Dangers from machines must be *learned*.

Of course, we *have* learned how to master complex tasks on a consistent basis: to fly airplanes, to build skyscrapers, and to perform life-saving surgeries. In order to do these amazing things, we bolster our innate abilities with research, training, procedures, teamwork, and experience. However, as the Boeing 299 crash showed, along with thousands of other disasters before and after, this is not enough. As valuable as preparation is, we still need a safeguard against our vulnerabilities at the *moment* of execution. Being distracted, forgetful, tired, or stressed can negate years of training when it results in a critical step being missed.

Connecting The Checklist To Troubleshooting

There are two juicy opportunities to incorporate checklists into your fix-it routines:

1. As a means to guide the problem discovery or repair phases. That is, using a checklist *while* troubleshooting.
2. By transforming what you *learned* while making a repair into recommendations for the best way of doing

something in the future. That is, using checklists to prevent failures from recurring.

Checklists As A Repair Guide

We've talked about various ways to formalize and communicate your troubleshooting knowledge, such as the venerable "troubleshooting tree." Troubleshooting trees are good for situations where there are a lot of "if X do this, but if Y then do this instead" branches. Trees easily communicate an extraordinary level of complexity with respect to the discovery phase of the fix-it process. I've seen dense troubleshooting trees that look like circuit diagrams when viewed from a distance.

Just as often, you'll have troubleshooting recipes that are simply a series of steps that must be done in a particular order. While the more open-ended problem discovery phase may be better suited to trees, the repair/correction phase is usually linear and therefore well suited to a checklist. In my company, we used troubleshooting checklists for recovery situations, like restoring a database. These procedures would have so many steps, needing to be done in the correct sequence, that making a checklist was essential.

Checklists As Preventative Medicine

If you've fixed enough things, the thought that "This was preventable!" is bound to cross your mind at some point. Even more so if you are consistently scrutinizing your breakdowns with a formal method like 5 Whys. Checklists are appropriate for any human-machine interaction, especially for those situations where the person at the controls can prevent something bad from happening by altering their behavior.

- **For operators:** when I say "operators," I mean anyone who uses machines for work, the people actually getting the job done. Of course, this is a huge class of people: every organization has a broad section of their employees that utilize machines, big and small, to accomplish their work. There's usually a right way and a…less than right way to operate a machine, but checklists go beyond education of "do's and don'ts." Training may improve awareness, but there will be situations where you need something done the right way every time. If you're encountering failures that could be prevented through an enforced procedure, the checklist is a great match. Consider physically attaching a checklist to a machine so that it gets used every time.
- **For designers:** since this section is about prevention, let's go all the way back to the people who are responsible for creating machines. Many firms make machines or tools that are used internally: manufacturer and consumer all in one. I've worked in software, where this is especially true: we'd often write programs for our own purposes, as well as for external clients. When you make your own tools, you will inevitably discover their flaws when they fail. Be sure to learn from these incidents by pushing that knowledge all the way back into the design process. Consider a "design principles checklist": we had one for our programmers that asked them to consider things like security and performance when developing a new feature or fixing a bug.

Doing The Right Thing…Automatically

For all the benefits of the checklist, it still requires vigilance. What good is a checklist if it's not enthusiastically used *every* time? Therefore, it helps to have a workforce that is educated on its benefits. But education may not be enough to force adoption, so the checklist will likely need management support as an official policy. Teams must be trained to use the checklist as a point of collaboration, and the organizational culture must tolerate challenging superiors when an item is missed. And so on and so forth: you can see the checklist benefits from an ecosystem that is favorable to it sustained use.

The organization that has integrated the checklist is an advanced species, but I believe there is another level beyond that, where you use automation and alter workflows to mitigate the need for checklists. Let me give you an example that shows this transition: when Discovery Mining first launched, I was the *de facto* systems administrator. Setting up a new server, I would do everything by hand: build the computer, install the operating system, partition the hard drives, set the networking parameters, choose the administrator password, load the correct software packages, etc. As we continued to grow, the number of servers also grew. Because I was doing everything by hand, every server turned out a little bit different. I'd eventually catch these discrepancies, but they would often lead to problems, and sometimes even an outage of our service!

To correct the situation, I made a checklist: I wrote down all the steps for setting up a server. Following the checklist whenever I deployed a new server dropped my error rate substantially. Also, it was now easy to get help for this task

when I finally hired our first systems administrator. Using the checklist, we set up a few servers together. With checklist in hand, he was able to take over from there. By the way, this is a good example of how checklists—or any documentation—can be the basis for collaboration and delegation.

The checklist worked well and was far superior to the "artisanal" phase that came before: errors stemming from incorrect server configurations dropped dramatically. However, it wasn't perfection. Forgetfully, sometimes the checklist wouldn't be used. Even when it was, occasionally a step would be missed. Also, now we were really growing fast and we might need to install 20 new servers in a single day. People would try to set up multiple servers at a time and get confused as to where they were in the checklist. We needed a new solution for this "high-volume" era and we found it in the world of Configuration Management (specifically, we used a software package called Puppet). Once Puppet was deployed on our servers, we specified the ideal configuration and the configuration management software would enforce it everywhere in our infrastructure. The software even protected against errors *after* installation: if a systems administrator came along and made a change to a server that violated the established policy, it would automatically be rolled back and made right. Beautiful.

From then on, I was fanatical about having our systems "do the right thing" on their own. If you're using a checklist to ensure a tricky sequence of events goes correctly, you have to ask yourself, "Why is it so hard for people to get this workflow right? Why have we designed something that is so prone to failure that it requires a checklist?"

It's a fair question and may lead you to some big improvements. Going back to the story of the origin of the checklist, we can push ourselves and think of ways to do better. Can we add another layer of automated safety so that the checklist isn't the only thing standing between life and death? What if that Boeing 299 had a single button that adjusted all of the plane's settings to "takeoff" mode (including disengaging the gust locks)? Or, a warning system that would blink and beep if the engines were started while the gust locks were still engaged? Better yet, perhaps the ignition system could be wired so that it's impossible to start the engines if the gust locks are engaged. Or, what about a backup system that would *automatically* disengage the gust locks if the plane exceeded a certain speed?

If you combine these kinds of improvements with the corrective power of the checklist, any task can be done safely.

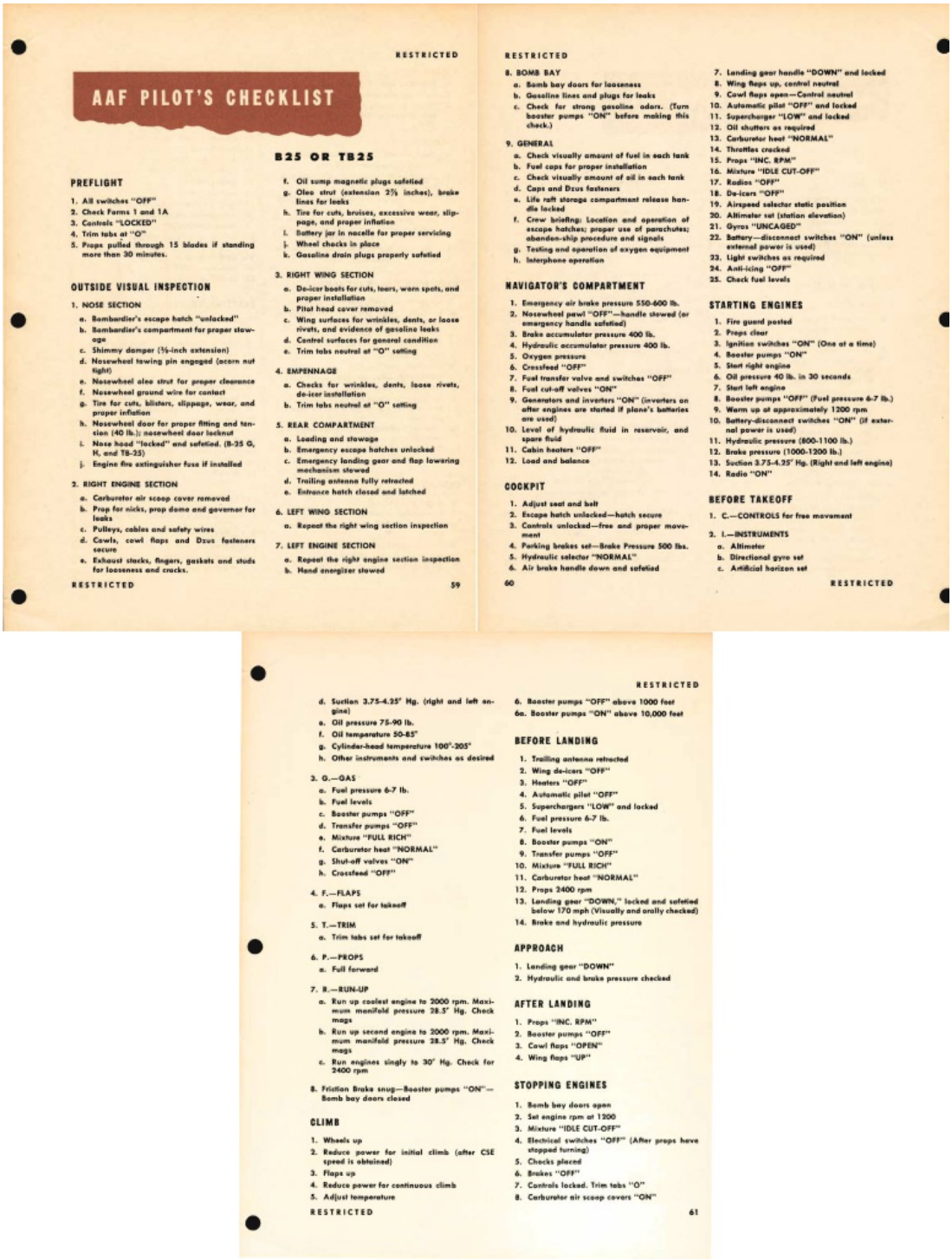
Saving Lives

The final chapter of *The Checklist Manifesto* is a gripping account of the checklist actually saving someone's life. This time it is Gawande himself who sees the benefits first-hand, although he was initially reluctant to introduce it to his operating room:

> But in my heart of hearts—if you strapped me down and threatened to take out my appendix without anesthesia unless I told the truth—did I think the checklist would make much of a difference in my cases? No. In my cases? Please.
>
> **Atul Gawande, The Checklist Manifesto** [1]

Although I have readily advised others to use them, I'll admit that in the past I've also been reluctant to *personally* adopt process-related constraints on how I work. Like Gawande, I used to think that, because I'm good at what I do, using something like a checklist was beneath me.

But, I've been humbled too many times to think I'm above the checklist. Even with vigilance, talent, and experience, you will make mistakes. As a pilot, I've seen first-hand the value of checklists. I've flown the Cessna 172, a relatively simple airplane (at least compared to a modern airliner). Even so, the 172's checklist contains over 100 items! Here's the checklist for the B-25, a WWII-era bomber. As you can see, there's a lot to get right:



RESTRICTED

AAF PILOT'S CHECKLIST

B25 OR TB25

PREFLIGHT

1. All switches "OFF"
2. Check Forms 1 and 1A
3. Controls "LOCKED"
4. Trim tabs at "O"
5. Props pulled through 15 blades if standing more than 30 minutes.

OUTSIDE VISUAL INSPECTION

1. NOSE SECTION
   - a. Bombardier's escape hatch "unlocked"
   - b. Bombardier's compartment for proper stowage
   - c. Shimmy damper (½-inch extension)
   - d. Nosewheel towing pin engaged (acorn nut tight)
   - e. Nosewheel oleo strut for proper clearance
   - f. Nosewheel ground wire for contact
   - g. Tire for cuts, blisters, slippage, wear, and proper inflation
   - h. Nosewheel door for proper fitting and tension (40 lb.); nosewheel door locknut
   - i. Nose hood "locked" and safetied. (B-25 G, H, and TB-25)
   - j. Engine fire extinguisher fuse if installed
2. RIGHT ENGINE SECTION
   - a. Carburetor air scoop cover removed
   - b. Prop for nicks, prop dome and governor for leaks
   - c. Pulleys, cables and safety wires
   - d. Cowls, cowl flaps and Dzus fasteners secure
   - e. Exhaust stacks, fingers, gaskets and studs for looseness and cracks.
   - f. Oil sump magnetic plugs safetied
   - g. Oleo strut (extension 2⅝ inches), brake lines for leaks
   - h. Tire for cuts, bruises, excessive wear, slippage, and proper inflation
   - i. Battery jar in nacelle for proper servicing
   - j. Wheel chocks in place
   - k. Gasoline drain plugs properly safetied
3. RIGHT WING SECTION
   - a. De-icer boots for cuts, tears, worn spots, and proper installation
   - b. Pitot head cover removed
   - c. Wing surfaces for wrinkles, dents, or loose rivets, and evidence of gasoline leaks
   - d. Control surfaces for general condition
   - e. Trim tabs neutral at "O" setting
4. EMPENNAGE
   - a. Checks for wrinkles, dents, loose rivets, de-icer installation
   - b. Trim tabs neutral at "O" setting
5. REAR COMPARTMENT
   - a. Loading and stowage
   - b. Emergency escape hatches unlocked
   - c. Emergency landing gear and flap lowering mechanism stowed
   - d. Trailing antenna fully retracted
   - e. Entrance hatch closed and latched
6. LEFT WING SECTION
   - a. Repeat the right wing section inspection
7. LEFT ENGINE SECTION
   - a. Repeat the right engine section inspection
   - b. Hand energizer stowed

RESTRICTED 59

RESTRICTED

8. BOMB BAY
   - a. Bomb bay doors for looseness
   - b. Gasoline lines and plugs for leaks
   - c. Check for strong gasoline odors. (Turn booster pumps "ON" before making this check.)
9. GENERAL
   - a. Check visually amount of fuel in each tank
   - b. Fuel caps for proper installation
   - c. Check visually amount of oil in each tank
   - d. Caps and Dzus fasteners
   - e. Life raft storage compartment release handle locked
   - f. Crew briefing: Location and operation of escape hatches; proper use of parachutes; abandon-ship procedure and signals
   - g. Testing and operation of oxygen equipment
   - h. Interphone operation

NAVIGATOR'S COMPARTMENT

1. Emergency air brake pressure 550-600 lb.
2. Nosewheel pawl "OFF"—handle stowed (or emergency handle safetied)
3. Brake accumulator pressure 400 lb.
4. Hydraulic accumulator pressure 400 lb.
5. Oxygen pressure
6. Crossfeed "OFF"
7. Fuel transfer valve and switches "OFF"
8. Fuel cut-off valves "ON"
9. Generators and inverters "ON" (inverters on after engines are started if plane's batteries are used)
10. Level of hydraulic fluid in reservoir, and spare fluid
11. Cabin heaters "OFF"
12. Load and balance

COCKPIT

1. Adjust seat and belt
2. Escape hatch unlocked—hatch secure
3. Controls unlocked—free and proper movement
4. Parking brakes set—Brake Pressure 500 lbs.
5. Hydraulic selector "NORMAL"
6. Air brake handle down and safetied
7. Landing gear handle "DOWN" and locked
8. Wing flaps up, control neutral
9. Cowl flaps open—Control neutral
10. Automatic pilot "OFF" and locked
11. Supercharger "LOW" and locked
12. Oil shutters as required
13. Carburetor heat "NORMAL"
14. Throttles cracked
15. Props "INC. RPM"
16. Mixture "IDLE CUT-OFF"
17. Radios "OFF"
18. De-icers "OFF"
19. Airspeed selector static position
20. Altimeter set (station elevation)
21. Gyros "UNCAGED"
22. Battery—disconnect switches "ON" (unless external power is used)
23. Light switches as required
24. Anti-icing "OFF"
25. Check fuel levels

STARTING ENGINES

1. Fire guard posted
2. Props clear
3. Ignition switches "ON" (One at a time)
4. Booster pumps "ON"
5. Start right engine
6. Oil pressure 40 lb. in 30 seconds
7. Start left engine
8. Booster pumps "OFF" (Fuel pressure 6-7 lb.)
9. Warm up at approximately 1200 rpm
10. Battery-disconnect switches "ON" (if external power is used)
11. Hydraulic pressure (800-1100 lb.)
12. Brake pressure (1000-1200 lb.)
13. Suction 3.75-4.25" Hg. (Right and left engine)
14. Radio "ON"

BEFORE TAKEOFF

1. C.—CONTROLS for free movement
2. I.—INSTRUMENTS
   - a. Altimeter
   - b. Directional gyro set
   - c. Artificial horizon set

RESTRICTED 60

RESTRICTED

- d. Suction 3.75-4.25" Hg. (right and left engine)
- e. Oil pressure 75-90 lb.
- f. Oil temperature 50-85°
- g. Cylinder-head temperature 100°-205°
- h. Other instruments and switches as desired

3. G.—GAS
   - a. Fuel pressure 6-7 lb.
   - b. Fuel levels
   - c. Booster pumps "OFF"
   - d. Transfer pumps "OFF"
   - e. Mixture "FULL RICH"
   - f. Carburetor heat "NORMAL"
   - g. Shut-off valves "ON"
   - h. Crossfeed "OFF"
4. F.—FLAPS
   - a. Flaps set for takeoff
5. T.—TRIM
   - a. Trim tabs set for takeoff
6. P.—PROPS
   - a. Full forward
7. R.—RUN-UP
   - a. Run up coolest engine to 2000 rpm. Maximum manifold pressure 28.5" Hg. Check mags
   - b. Run up second engine to 2000 rpm. Maximum manifold pressure 28.5" Hg. Check mags
   - c. Run engines singly to 30" Hg. Check for 2400 rpm
8. Friction Brake snug—Booster pumps "ON"—Bomb bay doors closed

CLIMB

1. Wheels up
2. Reduce power for initial climb (after CSE speed is obtained)
3. Flaps up
4. Reduce power for continuous climb
5. Adjust temperature

RESTRICTED

- 6. Booster pumps "OFF" above 1000 feet
- 6a. Booster pumps "ON" above 10,000 feet

BEFORE LANDING

1. Trailing antenna retracted
2. Wing de-icers "OFF"
3. Heaters "OFF"
4. Automatic pilot "OFF"
5. Superchargers "LOW" and locked
6. Fuel pressure 6-7 lb.
7. Fuel levels
8. Booster pumps "ON"
9. Transfer pumps "OFF"
10. Mixture "FULL RICH"
11. Carburetor heat "NORMAL"
12. Props 2400 rpm
13. Landing gear "DOWN," locked and safetied below 170 mph (Visually and orally checked)
14. Brake and hydraulic pressure

APPROACH

1. Landing gear "DOWN"
2. Hydraulic and brake pressure checked

AFTER LANDING

1. Props "INC. RPM"
2. Booster pumps "OFF"
3. Cowl flaps "OPEN"
4. Wing flaps "UP"

STOPPING ENGINES

1. Bomb bay doors open
2. Set engine rpm at 1200
3. Mixture "IDLE CUT-OFF"
4. Electrical switches "OFF" (After props have stopped turning)
5. Checks placed
6. Brakes "OFF"
7. Controls locked. Trim tabs "O"
8. Carburetor air scoop covers "ON"

61

The checklist for the B-25 Mitchell Bomber.

(image: U.S. Army Air Forces – Office of Flying Safety / Internet Archive)

Do you think you could remember all this *all* the time? Me neither! But you know that every item on that checklist has been the source of trouble, perhaps even a fatality, for some unfortunate pilot.

Let's return to Dr. Gawande's close call. During surgery one day, he accidentally makes a tear in a patient's vena cava (one of the large veins that carries blood into the heart):

> But we had run the checklist at the start of the case. When we had come to the part where I was supposed to discuss how much blood loss the team should be prepared for, I said, "I don't expect much blood loss. I've never lost more than one hundred cc's." I was confident. I was looking forward to this operation. But I added that the tumor was pressed right up against the vena cava and that significant blood loss remained at least a theoretical concern. The nurse took that as a cue to check that four units of packed red cells had been set aside in the blood

> bank, like they were supposed to be—"just in case," as she said.
>
> They hadn't been, it turned out. So the blood bank got the four units ready. And as a result, from this one step alone, the checklist saved my patient's life.
>
> **Atul Gawande, The Checklist Manifesto** [1]

Reading this for the first time was very moving—if you're a process aficionado, it doesn't get more beautiful than this. Of course, I highly recommend reading *The Checklist Manifesto* for the whole story, including how the nascent adoption of the checklist in medicine has significantly reduced surgical complications. Also, take a look at Gawande's companion "Checklist for Checklists," which outlines the principles for creating a really effective checklist.

But, you don't need any more preparation, you're ready to begin working with checklists right now. I hope I've made the case for how simple yet powerful they are! When you start using them, you too can experience the kind of payoff Dr. Gawande had in his operating room, of a crisis averted and a job well done.

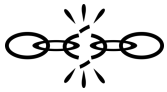
References:

- Header image: *"STS-96 mission specialist Ellen Ochoa beside the Volatile Removal Assembly Flight Experiment (VRAFE) located in the Spacehab DM during the flight. Ochoa is floating upside down beside the module and is holding a checklist."* June 2, 1999. NASA. Retrieved from Flickr, https://www.flickr.com/photos/nasacommons/29863224741/.
- [1] Atul Gawande. *The Checklist Manifesto: How To Get Things Right,* (New York: Metropolitan Books, 2009), pgs. 18, 187, 191.
- [2] Walter J. Boyne, "The Checklist," *Air Force Magazine*. August, 2013.
- [3] "WHO Guidelines for Safe Surgery" (First Edition, 2008), World Health Organization, pg. 8.
- [4] Model 299 Crash Fact Sheet. National Museum of the US Air Force.

Making A List, Checking It Off was originally published October 1, 2013.

Notes:

Failure Most Foul: Fraud and Sabotage



> People trust each other by default, and most of the time we're right to do so. Society would break down if we distrusted everything that everybody did.
>
> **Alex Chaffee**

In the Big Idea, I laid out my thesis for *The Art Of Troubleshooting:* that "all machine problems are human problems." Unfortunately, there's a class of "human problems" that you need to be aware of, the deliberate and malevolent kind. We're talking about things like fraud and sabotage. You need to recognize these as possibilities when you are out in the field solving problems.

The troubleshooting strategies I've presented have been discussed in a context you probably weren't even aware of: **that the people you are working with share the goal of having your systems operate smoothly.** Sure, there may be personality conflicts, differences of opinion, attempts to be helpful that are ultimately misleading, or even some outright hostility if someone feels like their territory is being violated. However, at a base level, most of the people you interact with will respect your goal of getting a system running again. Even if they don't offer to lend a hand, usually the worst that will happen is that they will be indifferent to your efforts. When it comes to your co-workers, a common goal is implicit in your relationship. After all, if they didn't support the purpose of your organization, even at the superficial level of "it's only for the paycheck," they'd be working somewhere else!

When someone intentionally violates the tacitly held belief that "we're all on the same team," things are going to get weird—and ugly. Stopping a machine from working on purpose adds a wrinkle to the normal "A causes B" model that you're used to relying on for troubleshooting. That's because the "A" that causes "B" is typically an *unintentional* force (wear and tear, unforeseen consequences, acts of nature, etc.). If you're not considering the possibility of intentional failures, you will be entertaining theories that belong in a Tolkien novel. The hands of a malevolent person can make things happen that look like magic.

Is someone trying to bring down your house? Some industries are very aware of the possibility of "intentional failures." You should be too…

(image: Charles O'Rear / The U.S. National Archives)

Failures And Judgements

The origin of every machine is mankind's purposeful intentions: that's ultimately the sense in which "all machine problems are human problems." By now, our collective experience tells us that **all machines eventually break down.** Deploying a machine for a given purpose carries with it an unspoken assumption of failure somewhere down the road. Routine maintenance and periodic replacement are responses to this reality.

Even though we know that all machines will eventually fail, the how and when are often unexpected. Sure, you may know that your car can break down, but you didn't know that it would gasp its final breath last Tuesday in the midst of rush-hour traffic. Unfortunately, the inevitability of your car's demise does not give you specific knowledge of the exact date and time. Isn't it interesting that we live in a world where it's certain that every machine will eventually break down, and yet our *experience* of those failures is one of surprise?

Even a well-maintained system can unexpectedly malfunction, so we don't judge a machine's owner when that happens, at least not in a moralistic "you are a bad person" kind of way. If we rely upon a company's services (and hence, their machines) and they are unable to make good on their promises, we may judge them as being *incompetent* for not having a contingency plan or sufficient redundancy. However, no one says Netflix is evil when they can't watch movies on Christmas Eve. Again, the fact that the other party is acting with positive intentions and in good faith makes the difference.

Intentions Don't Matter

Intentions may matter for moral judgements, but when it comes to repair, a failure is a failure. It doesn't matter if the cause is a squirrel in the substation, normal wear and tear, a storm, forgotten maintenance, or sabotage, because the goal of fixing something is always the same: fulfill the need that the machine was serving.

When it comes to choosing strategies, the efficacy of a troubleshooting recipe is tied to the design of a system. The *intentions* (or lack thereof) behind a cause is not a factor. For example, "copying one that works" or quickly narrowing in via half-splitting will be effective, or not effective, based on the machine's nature. Think of it like this: a cut needs the same medical care regardless of whether the knife pierced the skin accidentally or intentionally. Stitches, bandages, and antiseptic are used in either case: the reality of the wound is the primary consideration, not the story behind the injury. The broad strategies I've given you are useful precisely because they are applicable to a wide variety of failure scenarios, independent of cause's origin. If you had to know the origin of a cause before you chose a strategy, troubleshooting would be difficult, given that the cause is frequently unknown!

Therefore, when fixing something damaged by sabotage, troubleshooting can proceed like any other case. However, there are some special considerations, especially during the "cleaning up" phase. At some point during a repair, every good troubleshooter asks, "Why did this happen? How can it be prevented from recurring?" These questions have special pitfalls with respect to deliberate failures.

Mistaking Intentional For Natural

Happily, most malfunctions are free of malice. They are unintended and happen in the normal course of people trying to complete their work. Many times, no human is even present during a breakdown; the machine is operating on its own when it dies, and the problem is discovered later. This is the bulk of our experience and we know that experience is a powerful guide. But this well-worn path can blind a troubleshooter to the possibility of sabotage, resulting in failures that look supernatural.

Be especially suspicious when troubleshooting anything related to the security of people or property. We're talking about things like cash registers, safes, locks, lockers, and entryways (doors and windows). Likewise with their digital equivalents: firewalls, account credentials, virus scanners, etc. I'm always urging you to consider context when troubleshooting; awareness of the environment in which a system is installed is instrumental to finding causes. Up till now, that meant paying attention to things like upstream or downstream machines in a workflow, shared resources like electricity, or environmental variables like temperature and humidity. Now, we'll add one last critical part of a machine's context: the people who have access to it. What if someone doesn't share the goal of having everything run smoothly?

Be sure to consider the *entire* ecosystem of which your systems are a part: attacks can come from customers, vendors, and the general public. Finally, there's always the possibility of the "inside job," where your own co-workers are up to no good.

Mistaking Natural For Intentional

On the flip side, there are industries and occupations that are hyper-aware of the possibility of foul play: banks, casinos, drug dealers, credit card issuers, pawn shops, couriers, accountants, etc. A never-ending history of fraud and theft have plagued these professions, leading to a higher level of suspicion in the possibility of foul play. If something weird happens on the floor of a casino, you can be sure the pit boss is watching out for a scam. When a large amount of money goes missing from an account, most bank managers probably are thinking about embezzlement alongside other, more benign possibilities.

This mindset is justified: the casino that isn't proactively searching for people trying to cheat them won't stay open for long. Likewise, the too trusting pawnbroker will have an unpleasantly short career. For these scam-prone industries, their paranoia is verified by experience, by the innumerable attempts, foiled and successful, to "bring down the house."

We've already seen how the naive troubleshooter, closed to the possibility of foul play, can be blindsided. On the other hand, always suspecting malevolence can be a tiresome burden for you and your employees. Low-trust environments, where everyone is a always a suspect, are not conducive to getting the best out of your people.

Leaving Traces

Sometimes, it will be very difficult to tell the difference between an intentional failure and a "normal" one. If the person sabotaging you is smart, they may take great care to prevent detection and make the failure appear like it was naturally caused. As always, a heightened sense of awareness and being present can aid in the detection of foul play. Rich Kral, a veteran HVAC repairman, related this to me when I interviewed him about troubleshooting:

> There have been times when I've questioned whether or not it [a breakdown] has been sabotage. You have a particular piece of machinery, and you open up the panel and think, "Someone else has been in here," when you're the only one who was supposed to have been in there. You can tell, almost by the energy, that someone else has been in the panel, because you put your screws in a certain way. I have a certain sense of feel for tightness and when they are so doggone tight that I can't get them undone without stripping the screws, I think, "Somebody has done this, somebody else has been here, this is not me." You know your own trail, you know your own path, you know where you have walked before, and you can sense when someone else is following you. It's very real.
>
> **Rich Kral**

Can you focus and tune into these nuances when a failure seems suspicious? What in a machine's context seems out of place? Rich is clearly in touch with the details of a situation when he can tell by just the tension of the screws whether someone else has been "following his path."

A Pattern Of Behavior

> Goldfinger's flat, hard stare didn't flicker. He might not have heard Bond's angry-gentleman's outburst. The finely chiselled lips parted. He said, 'Mr Bond, they have a saying in Chicago: "Once is happenstance. Twice is coincidence. The third time it's enemy action."'
>
> **Ian Fleming, Goldfinger** [1]

Sadly, the ambiguity between intentional and unintentional often means that *multiple* instances of sabotage may be required to definitively tell the difference. Good record-keeping is essential to begin piecing together the puzzle. For further examples, we turn to the real-life thriller *The Cuckoo's Egg* by Clifford Stoll. In this compelling book, Stoll recounts his exploits of tracking a hacker that has penetrated the computer systems at Lawrence Berkeley National Laboratory. The tale starts out modestly, with the lab's accounting system showing a very small discrepancy ($0.75!) in the billing records for the computer system:

> The computer's books didn't quite balance; last month's bills of $2,387 showed a 75-cent shortfall.
>
> Now, an error of a few thousand dollars is obvious and isn't hard to find. But errors in the pennies column arise from deeply buried problems, so finding these bugs is a natural test for a budding software wizard.
>
> **Clifford Stoll, The Cuckoo's Egg** [2]

Notice that Stoll understandably assumes a benign cause as he begins his investigation: he thinks the cause of this bookkeeping error is a bug in the accounting software. This is a good example of the bias I was talking about before: we usually attribute failures to unintentional causes before considering more sinister explanations. My aim is to not change this basic belief, just to put sabotage on your list of possibilities.

Once you suspect sabotage, you will need to start collecting information (often from surveillance) to prove or disprove the theory. To make this happen, Stoll "liberates" every printer in the organization and unrolls his sleeping bag on the office floor:

> All we'd need are fifty teletypes, printers, and portable computers. The first few were easy to get—just ask at the lab's supplies desk. Dave, Wayne, and the rest of the systems group grudgingly lent their portable terminals. By late Friday afternoon, we'd hooked up a dozen monitors down in the switchyard. The other thirty or forty monitors would show up after the laboratory was deserted. I walked from office to office, liberating personal computers from secretaries' desks. There'd be hell to pay on Monday, but it's easier to give an apology than get permission.
>
> **Clifford Stoll, The Cuckoo's Egg** [2]

The thing I like about *The Cuckoo's Egg* is that Stoll's zeal really comes through in his account. You get a good sense of the engrossing passion that accompanies the pursuit of someone who's wronged you. I've felt these same emotions, having also been harmed by a saboteur at work. During that incident, I too felt like I would go to the ends of the earth to catch the perpetrator. Be careful though, while this passion can give you the energy to see an investigation through to the end, it needs to be tempered with the goal of justice. While Stoll clearly becomes obsessed with catching his hacker, he also involves law enforcement and coordinates his investigation with them. You should do the same.

When it comes to interpreting patterns of behavior, you might need to accelerate the process. If you want to make sure it's actually sabotage, then decoys, honeypots, or traps should be on the table (warning: make sure what you're doing is legal!). However, you'll need to balance catching your nemesis with getting your regular job done. Stoll grapples with this tension as his preoccupation with the hacker begins to conflict with his other work responsibilities:

> Lawrence Berkeley Laboratory was tired of wasting time on the chase. I hid my hacker work, but everyone could see that I wasn't tending to our system. Scientific software slowly decayed while I built programs to analyze what the hacker was doing.
>
> **Clifford Stoll, The Cuckoo's Egg** [2]

The Simplest Explanation

If you're dealing with sabotage, but mentally exclude it as a possibility, you're going to find yourself entertaining wild, fantastic theories of how A caused B. For a related example of the flights of fancy people embark on when something obvious is excluded, we turn to the repeated attempts to explain the fascinating Voynich Manuscript. The Voynich Manuscript is a book from the 15th century that contains an unknown language, interlaced with mysterious diagrams and artwork. What does it mean? Who wrote it? It's captivated many people over the centuries.

Some of the best minds have tried to break its code, even up to the current era with the latest cryptographic methods. Then along came Gordon Rugg, a linguist at Keele University. Rugg's work says that the manuscript means nothing, that it's just a clever hoax![3] Not only that, but he also showed how a similar text could be created. Now, what's more

likely: that the text's cipher is so well constructed that it continues to elude even the most advanced code-breaking techniques? Or, that it's just a clever ruse? I love a good mystery, but my money would be on the latter.

Once you make the assumption that the Voynich Manuscript means something, you're lead to believe some very unlikely things, among them that 500 years of scrutiny and advancements in cryptography still aren't good enough to make sense of it! Rugg's work may be brilliant, but the foundation of his breakthrough was elegantly simple: a willingness to dispense with the notion that it had meaning. Similarly, if you catch yourself entertaining bizarre theories about the cause of a failure while troubleshooting, make sure you double-check your key assumptions. Among the most important to reexamine is that something just "happened on its own."

A page from the mysterious Voynich Manuscript. Is it a hoax?
(image: Yale University Library)

Keep Good Records

When starting to investigate what you think is a case of sabotage, get out a notebook and start writing. What happened, when, and your response will be invaluable information as the situation evolves. When assessing why Stoll prevailed, it's tempting to focus on the fact that he both was smart and determined, but his diligent documentation was equally important. He kept a logbook of everything that happened, constantly updating it as the investigation proceeded. This record allowed him to cross-check information, make correlations, and narrow the search. Oh yeah, and afterwards he got to turn it into a best-selling book. How's that for an incentive?

Stoll's record-keeping made a huge difference in the trial that eventually convicted Markus Hess, who was selling the information to the KGB. Between his logbook, phone traces, and printouts of Hess' activity, the case was easily won:

> How did I feel? Nervous, yet confident in my research. My logbook made all the difference. It was like presenting some observations to a room of astronomers. They may disagree with the interpretation, but they can't argue with what you saw.
>
> **Clifford Stoll, The Cuckoo's Egg** [2]

Picking Up The Pieces

Cleaning up after an act of sabotage is a messy thing. It's going to have an entirely different feeling than a typical round of root cause analysis. The main difference is healing your violated trust. Obviously, if the culprit was one of your own, the impact on your staff's morale will be much worse. Either way, your team's cohesion will be tested both during the incident and after. People don't like to be betrayed.

Not Dishonest, But Not Exactly Forthcoming Either

Deception lies along a spectrum: not everything will be so extreme as the active malevolence of a saboteur. There are many gray areas, like when someone just isn't giving you the whole story. I've often encountered a reluctance to be forthcoming when interviewing people about a system failure. In my experience, it usually has to do with a feeling of embarrassment. A person may feel bad because they view themselves as partly to blame for a breakdown. They may also feel stupid that they didn't know the "proper" way to operate a machine. If the situation borders on negligence, perhaps they want to avoid the consequences of being held accountable.

They may react to these feelings by being overly vague, curt, or by omitting details that implicate themselves. Dealing with this is tricky: if you push too hard for answers and back them into a corner, they may double down and start lying. At the same time, if they were partially responsible, you want to make sure they understand their role in the problem and how they can prevent it from happening in the future.

For milder cases where malice isn't likely, I think there's a middle way. I've investigated situations where a person was obviously feeling guilty and their behavior communicated embarrassment. Forcing them to say, "I was an idiot!" is a face losing proposition that isn't necessary to make your point. Without making them admit guilt, I would launch into an impromptu training session regarding proper use of the system in question. While the subtext is "you were responsible and I'm teaching you the right way," this allows them to graciously accept your guidance, free of shame.

References:

- Header image: Thew, R. & Fuseli, H. *Shakespeare–Hamlet–Prince of Denmark / painted by H. Fuseli R.A.; engraved by R. Thew.* 1796. London: published by J. & J. Boydell. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/95521789/.
- [1] Ian Fleming, *Goldfinger* (London: Penguin Books, 2004), pg. 166.
- [2] Clifford Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage,* (New York: Pocket Books, 2005), pgs. 3, 24, 195, 396.
- [3] Joseph D'Agnese, "Scientific Method Man,"*Wired.* September, 2004.

Failure Most Foul: Fraud and Sabotage was originally published November 15, 2013.

Notes:

Release The Chaos Monkeys: Intentionally Creating Failures

> The best way to get better at troubleshooting is to break things on purpose.
>
> **Austin Quade**

My Grandfather gave me one piece of romantic advice: before getting serious, take a prospective mate camping. Not RV'ing with all its creature comforts, but rather roughing it on a backwoods adventure. He said that hopefully it would rain, your sleeping bag would get soaked, you'd arrive at the campsite after sunset and be forced to set up in the dark, the fire would be impossible to start, and large ravenous bears would eat your food. The upside wouldn't end there: you'd also get to see your partner at their worst and get a glimpse of what they were really made of. Ah, those hardy Depression-era values. While courtship doesn't have to include a trip down the river, Deliverance-style, I always thought his scheme was a good idea. It's best to know how a person will react in advance of when you might really need them.

Once again troubleshooting mirrors life: it's also true that important knowledge about a machine is gained when seeing it at its worst, and best obtained before it matters. Like my Grandfather's camping regimen, there are ways to learn this

critical information about a system, in advance. That's right, you don't have to wait for a breakdown to occur to understand what something will do under duress. We'll take a lesson from the world of product testing, where designers concoct shenanigans to see how a new model will react to a variety of conditions, trying to get it to fail before it gets in a customer's hands. While this is something that you might normally associate with manufacturing, I want you to understand that it's also a tool for the troubleshooter.

Are you ready to let them run wild among your machines?
(image: Tambako The Jaguar / CC BY-ND 2.0)

Monkey On Your Back

When it comes to creating mischief to foster understanding, the entertainment company Netflix has one of the most advanced systems ever conceived. When Netflix moved their services into the "cloud," they encountered some unique problems. Amazon's Web Services (aka, AWS, a cloud computing environment) offered savings to the bottom line, but presented some challenges that forced them to rethink the design of some of their systems. The details aren't important, but suffice it to say that the co-tenancy of a shared resources environment includes a more variable performance profile and lowered reliability versus what the Netflix engineers were used to in their own data centers.

Netflix took this new way of life and embraced it:

> **3. The best way to avoid failure is to fail constantly.**
>
> We've sometimes referred to the Netflix software architecture in AWS as our Rambo Architecture. Each system has to be able to succeed, no matter what, even all on its own. We're designing each distributed system to expect and tolerate failure from other systems on which it depends.
>
> If our recommendations system is down, we degrade the quality of our responses to our customers, but we still respond. We'll show popular titles instead of personalized picks. If our search system is intolerably slow, streaming should still work perfectly fine.
>
> **5 Lessons We've Learned Using AWS** [1]

But, we haven't gotten to the best part. Out of this desire to be truly resilient, the **Chaos Monkey** was born:

> One of the first systems our engineers built in AWS is called the Chaos Monkey. The Chaos Monkey's job is to randomly kill instances and services within our architecture. If we aren't constantly testing our ability to succeed despite failure, then it isn't likely to work when it matters most – in the event of an unexpected outage.
>
> **5 Lessons We've Learned Using AWS** [1]

That's right, the Chaos Monkey is a program that is constantly scurrying about, shutting things **off.** Not within the safe padded walls of a test environment, but in Netflix's live, customer-facing service. I want you to understand just how cool and unusual this is: it's like hiring some crazy drunk guy to run around your workplace, his shirt off, screaming obscenities, flipping switches, slamming doors, and knocking things off desks and shelves. And no one cares because your processes are so robust.

In addition to the challenges of moving their service to a shared environment, Netflix has another problem: at peak times, their service has been responsible for 1/3 of all Internet traffic in North America.[2] One third! Netflix is operating at a scale that is hard to test. How does one simulate that much Internet traffic? Exactly, you don't. Netflix's test environment will always be a toy compared to the real thing, so the Chaos Monkey is even more vital. Without something like it, there wouldn't be any way to definitively know the resiliency of their infrastructure.

It's Yours, So You Can Wreck It

Netflix has some special challenges, and the cookie-cutter building blocks of a cloud computing environment makes the Chaos Monkey a feasible option for them. You're probably thinking, there's no way I'd introduce such a thing into my work environment! And of course, you'd be right: unless you're prepared for it, unleashing a Chaos Monkey in your workplace would put you on the fast track to going out of business. However, there are lessons and strategies to be extracted from the concept.

First off, just get used to the notion that stress testing is an option for your machines, tools, etc. As noted before, this is a way of life for manufacturers who want to protect their reputations; they need to know what's going to happen when people actually use their products. Plus, consumers demand this information: they want to understand if a product is going to be suitable for their particular use. While the cover of a marketing brochure may feature beautiful people, the "technical specifications" section is vital for making a buying decision. These "technical specifications" come from somewhere—testing!

While manufacturers are well versed in testing, I feel it's something many consumers don't consider. The product is unwrapped from its beautiful packaging. There it sits, not to be messed with, only to be used as intended. Maybe some of this reluctance is a fear of loss: you paid your hard-earned money for a new thingamajig and so the last thing you want to do is to harm it while testing its limits. However, if lives or livelihoods are dependent upon something, a higher standard of certainty is required.

Stirring Up Trouble

You might not be as brave as Netflix: it takes a lot of preparation and the right circumstances to make mischief in your production environment. However, there still are plenty of opportunities to know your systems by seeing them at their worst, albeit in safer ways. Consider making a test environment, with duplicates of the machines you use in production, so that you can muck around and learn without risking disruptions to your business. What happens if you shut off a particular subsystem or reintroduce a known broken part? Try it and observe the results.

One major caveat is to understand that test environments are, by definition, not the same as the real thing. Knowing when it's appropriate to extrapolate your results, to make predictions about what will happen in real life, can be extremely difficult: history is littered with great ideas that tested well, but ultimately didn't work. The only generic advice one can give is that tests should closely resemble the real conditions, in all possible dimensions: time, machines, usage, personnel, environment, and context. This list is a mirror of the one presented in "Duplicate The Problem." Both testing and problem duplication share the same of goal of trying to match the details in which problems originate, that original context of the production environment.

Find a wide open space and let 'er rip.
(image: Library of Congress)

We did a lot of testing at Discovery Mining, mostly of the non-destructive variety. We also heavily vetted new equipment before putting it into service: for example, running a new server through its paces, perhaps for as long as a week, before trusting it in our infrastructure. Later on, we got our vendors to build our tests into their manufacturing processes. We'd order a new machine, and it would be tested and broken-in to our specifications before it was shipped to us.

Speaking of which, a break-in period is a great idea for new or recently repaired machines, especially if it can be done *before* a machine is deployed for real work. Statistically, a good number of failures will be clustered around those initial hours of usage. That's where manufacturing defects, installation problems, and configuration errors first show themselves.

If you designed a machine yourself, or have cobbled together something in the style of a systems integrator, then it's up to you to understand its limits. The software and hardware industries have long used a technique called "fault injection." This is where errors are purposefully introduced to a system and the results are observed. This process is applicable to both the digital and analog worlds: whether it's sending bad data to an API, shorting a connection on a circuit board, or loosening a belt in an engine, the principle is the same.

So, here's to monkeying around! Go forth and create some controlled mischief.

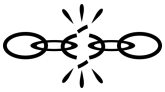
References:

- Header image: "NASA Langley Researches Crash Test." August 21, 2013. NASA. Retrieved from Flickr, https://www.flickr.com/photos/nasacommons/14883442256/.
- [1] 5 Lessons We've Learned Using AWS. *The Netflix Tech Blog.* December 16, 2010.
- [2] Don Reisinger, "Netflix gobbles a third of peak Internet traffic in North America," *CNET*. November 7, 2012.

Release The Chaos Monkeys: Intentionally Creating Failures was originally published November 19, 2013.

Notes:

You're Not Done Until You Tell Someone Else



> When you solve a problem, you want to share it with everybody!
>
> **Jamie Karrick**

The final act of any tough troubleshooting exercise should be to communicate the result. I learned this lesson the hard way: after going the extra mile to solve a problem, I'd often see it reappear later because I was silent on the matter. Your heroics won't provide a lasting benefit to society (or yourself) if the co-worker sitting next to you is unaware of the dragons you've slain.

When I think about this problem, my mind throws up an image of George Mallory. Was he the first to summit Everest? Sadly, it doesn't matter because he didn't live to either confirm or deny the possibility. While I respect the idea of climbing Everest solely because "it's there," if I was going to die trying to be the first, an entry in the record books would be a nice consolation prize. How different our view of history would be if a photo or diary survived, showing that Mallory was the first to the top! Again, communication is vital.

As a troubleshooter, your mountaintop has the fix at the summit. The "getting down," as Sir Edmund Hilary puts it, is learning from what happened and then communicating these lessons to others. Your metaphorical climb shouldn't be considered complete without *all* of these elements. This is just a general principle, so it's up to you to figure out the best way to feed back your insights to your colleagues, your industry, and the world. We've discussed many options for learning from a breakdown like root cause analysis (e.g., 5 Whys) and routine maintenance programs. Likewise, there an equally wide range of options for communicating the results of a fix-it victory (defeats, too): checklists,

troubleshooting trees, manuals, service bulletins, incident reports, on-line discussion forums, meetings with your colleagues, etc.

It's a tragedy to possess hard-won knowledge and not take that last step to disseminate it to others. Who could be saved by what you know? That's the spirit in which I wrote this book, it's not enough for me to be good at fixing things—I want to live in a world filled with amazing troubleshooters.

Pick up the mic, crank the volume, and tell the world what happened.
(image: Juan_Alvaro / CC BY 2.0)

Toot Your Own Horn Once In A While Or Be Taken For Granted

In your role as a troubleshooter, you're often responsible for maintaining the "invisible infrastructure" of people's daily lives. These are the things that people take for granted until they break down: their car, Internet connection, electricity in the wall socket, hot water, etc. The problem with things that people take for granted is they're not always appreciated like they should be. In "Zen And The Art Of Routine Maintenance," I related the ancient parable of the 3 physicians who were brothers. I told you to be like the eldest who "sees the spirit of sickness and removes it before it takes shape, so his name does not get out of the house." I'll add a qualification to that: be like that eldest brother, but also be your own PR agency and make sure that your name does get around. You won't keep your job if you don't!

As CTO, I managed the systems group at Discovery Mining. This department maintained the infrastructure for the company: the servers that ran the web site, the phone system that the salespeople used to call clients, the printers that the project managers used to print their reports. And so on. Exactly the kind of stuff that easily falls into the "taken for granted" category. At first, when presenting to the company during our weekly engineering meeting, I would focus on recent incidents and secondarily on progress reports for our current projects. This seemed reasonable because "if it bleeds, it leads," just like the 5 o'clock news, right?

It took a while, but eventually I realized that the systems team and I weren't getting much credit for everything that went right. Sure, the duty of maintaining all those servers, computer networks, phones, and printers was in our job description, but that doesn't mean it should go unappreciated. Also, focusing just on incidents made it seem like there was more chaos and instability in our infrastructure than there really was. It was likely the same phenomenon as the "Mean World Syndrome": a diet of sensational news with a large helping of violent crime can make you believe your surroundings are more dangerous than the statistics would indicate. By fixating people's attention on just the bad things, I had become the tabloid news of the company.

Therefore, I decided to change course and try a different approach. On a regular basis, I started to include an overview of all the the things that had gone right during our weekly meetings. Basically, periodic reminders of all those things

that people relied on, but might take for granted. During the latter part of my tenure, after our data collection program was bearing fruit, I was able to show these "hidden successes" graphically. I'd include things like:

- "Did you know? The website was up 100% last month."
- "Our Internet connection has plenty of spare bandwidth. I hope you've been enjoying your YouTube and FaceBook time."
- "Isn't it nice that the printers worked while you were preparing for that big presentation?"
- "Look at all that extra disk storage we anticipated you'd need. Now we're able to make good on that big contract that the sales team just closed."

Learn Then Tell

My parting advice is to go forth and learn from your failures. When you have insights to share, spread the news far and wide.

After that, you're done.

And I am too.

> Do your work, then step back.
> The only path to serenity.
>
> **Tao Te Ching (verse 9)** [1]

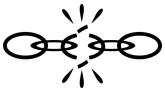
References:

- Header image: Rizzuto, A., photographer. *Group of onlookers gathered around two men engaged in conversation*. United States, New York. November, 1953. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2020636033/.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 9.

You're Not Done Until You Tell Someone Else was originally published November 19, 2013.

Notes:

The Boy Who Cried Wolf



> But this time the villagers, who had been fooled twice before, thought the boy was again deceiving them, and nobody stirred to come to his help. So the Wolf made a good meal off the boy's flock, and when the boy complained, the wise man of the village said: "A liar will not be believed, even when he speaks the truth."
>
> **The Shepherd's Boy, Aesop's Fables**

Over the years, in my tenure as a manager, I went through several iterations of "What exactly is my role?" Various answers have appeared: mentor, disciplinarian, maker of lists, taskmaster, procurer of take-out food, motivational speaker, and he-who-should-just-get-out-of-the-way.

What exactly I was managing seemed highly contextual, fluid, and often fleeting: this week's challenges were not like last week's. Therefore, a flexible approach seemed important. However, sometimes I would discover a management principle that would turn out to be long-lived. While I was in charge of a group of systems administrators, I had a powerful epiphany. For them, I realized the *thing* I was actually managing was their attention span. Indeed, I found it was a very precious commodity.

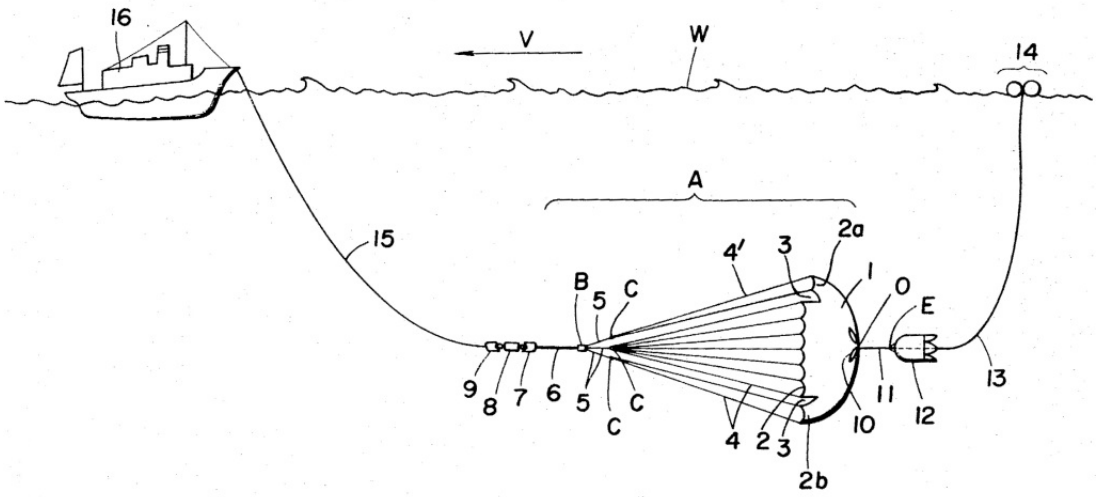
Next time, it might not be a goat…
(image: Library of Congress)

False Alarms

At Discovery Mining, the systems group maintained the infrastructure that sustained a very complicated web site. In line with our mantra **"know before the client knows,"** we implemented an extensive monitoring system. If something bad was going to happen, we wanted to to be aware of it, proactively meeting it head on. To that end, we automatically kept track of thousands of aspects of the systems under our care. The level of detail was impressive: within an individual server we could tell if a specific disk drive was working or the speed of its cooling fans.

You can imagine that monitoring all this minutiae produced a staggering amount of data. We did our best to make sure all our records were well-organized and graphically represented, so they could be quickly consulted on an as-needed basis. However, in the case of an emergency, our monitoring system was designed to grab our attention by lighting up our phones, pagers, and email inboxes.

Tuning our alerting system was a never-ending struggle, and perfectly illustrates the theme of this article. That's because not every alert turned out to be indicative of a real problem. There were many moments when our monitoring system morphed into that boy who cried wolf, with my staff playing the role of the circumspect villagers.



A sea anchor might provide stability and slow you down, but it's not attached to a fixed point.
(image: US Patent 3472195)

Normalization Of Deviance

Diane Vaughan, a professor of sociology at Columbia University, studied the Challenger space shuttle disaster in depth. Her resulting book, *The Challenger Launch Decision*, introduces an interesting concept called the **normalization of deviance**:

> Social normalization of deviance means that people within the organization become so much accustomed to a deviant behavior that they don't consider it as deviant, despite the fact that they far exceed their own rules…
>
> **Diane Vaughan** [1]

I think this problem is richly relevant for those that troubleshoot professionally: by definition, front-line technicians deal with "deviant behaviors" (aka, malfunctions) on a constant basis. When broken becomes part of your daily routine, it will reset your sense of normal. Likewise, the warning systems that troubleshooters rely upon are subject to these same psychological factors: too many false signals will eventually be ignored, just like the shepherd boy.

Homo sapiens has been remarkably adaptive, finding a way to survive in deserts, jungles, mountains, prairies, and the frozen tundra. Eking out a living in a wide variety of climates, terrains, and social conditions is a noble part of our humanity. This ability is surely aided by various psychological coping mechanisms: if you are thrown into an unfavorable environment, the ability to reset your expectations, focusing on what "success" means in the new context, is a must.

But alas, virtues can also be vices: because our expectations are elastic, they can be ratcheted up indefinitely in a never-ending exercise of "keeping up with the Joneses." If you've ever spent time on the hedonic treadmill, you know that a constantly moving target of success can feel unsatisfying; this is why people crave experiences that restore a sense of perspective to their lives.

Those same human capabilities that favor adaptation appear to be at work when it comes to the normalization of deviance. Vaughn eloquently explains the process:

> Signals of potential danger tend to lose their salience because they are interjected into a daily routine full of established signals with taken-for-granted meanings that represent the well-being of the relationship. A negative signal can sometimes become simply a deviant event that mars the smoothness of the ongoing routine. As the initiator's unhappiness grows, the number, frequency, and seriousness of signals of potential danger increase. While they would surely catch the partner's attention if they came all at once, this is not the case when new signals are introduced slowly amid others that indicate stability. A series of discrepant signals can accumulate so slowly that they become incorporated into the routine; what begins as a break in the pattern becomes the pattern. Small changes—new behaviors that were slight deviations from the normal course of events—gradually become

> the norm, providing a basis for accepting additional deviance.
>
> **Diane Vaughan, The Challenger Launch Decision** [2]

I definitely observed this process in action with my team. Over the course of a typical day, a systems administrator would receive many "signals" that included phone calls, emails, project managers stopping by to ask for updates, colleagues asking for help, etc. This "daily routine full of established signals" would be mixed with messages generated by our monitoring system. Within this context, evidence of deviations can accumulate slowly. Every new alert can subtly move your mental anchor, integrating it as the "new normal."

Alert Fatigue

Let's go back to the early days of the Homeland Security Advisory System. I remember making travel plans when this scheme was in place:

Living in orange and yellow: what does it mean to always be on alert?
(image: Wikimedia Commons)

If the threat level was blue or yellow or orange, what exactly was I supposed to do about it? Not travel at all? The most frustrating thing about this advisory system was the lack of specifics. The precise nature of the threats underlying the various levels were a secret, making it hard to assess your own situation. Most importantly, what particular course of action should be taken in response to a given threat level?

Using the DHS' "Chronology of Changes to the Homeland Security Advisory System," [3] I filled in a table with the time spent at the various threat levels*:

Start	End	Days	Threat Level
Mar 12, 2002	Sep 9, 2002	182	YELLOW
Sep 10, 2002	Sep 23, 2002	14	ORANGE
Sep 24, 2002	Feb 6, 2003	136	YELLOW
Feb 7, 2003	Feb 26, 2003	20	ORANGE
Feb 27, 2003	Mar 16, 2003	18	YELLOW
Mar 17, 2003	Apr 15, 2003	30	ORANGE
Apr 16, 2003	May 19, 2003	34	YELLOW
May 20, 2003	May 29, 2003	10	ORANGE
May 30, 2003	Dec 20, 2003	205	YELLOW
Dec 21, 2003	Jan 8, 2004	19	ORANGE
Jan 9, 2004	Jul 31, 2004	205	YELLOW
Aug 1, 2004	Nov 9, 2004	101	ORANGE
Nov 10, 2004	Jul 6, 2005	239	YELLOW
Jul 7, 2005	Aug 11, 2005	36	ORANGE
Aug 12, 2005	Aug 9, 2006	363	YELLOW
Aug 10, 2006	Aug 12, 2006	3	RED
Aug 13, 2006	Apr 20, 2011	1712	ORANGE

*The RED threat level was used only once from August 10-12, 2006: "for flights originating in the United Kingdom bound for the United States." The rest of the USA's commercial aviation activity was set to ORANGE for these days. For this brief split situation, I counted these days under the higher threat level (RED) set by the DHS.

Then added everything up:

Days	%	Threat Level
3	0.09%	RED
1942	58.37%	ORANGE
1382	41.54%	YELLOW
0	0.00%	BLUE
0	0.00%	GREEN
3327	**100.00%**	

Given that the threat level was never set below "elevated" (yellow), you have to wonder about the effect of having the

entire nation on alert for so long. When I added it all up, I didn't realize that we had spent over 5 years at the orange level ("High"). That's a long time! Five years spent doing anything will eventually become your baseline point of reference, the "new normal." But, if there really was an imminent threat, a sense of normalcy is probably not what the designers of the system intended.

After a while, I did what millions of other Americans did: ignore the Homeland Security Advisory System. This phenomenon is called "alert fatigue" and is well known in the health care industry, often studied in the context of doctors grappling with the deluge of alerts created by medical software. Attention is a finite human commodity and there's simply a limit to the number of alarm messages that can be processed before they are ignored and established as routine.

Mixed Messages and Vigilance

As we were monitoring thousands of details about our infrastructure at Discovery Mining, errors made configuring or tuning our alerts would also be multiplied several thousand-fold. Sometimes, the system would generate an overwhelming number of false positives. When receiving a large volume of alerts over the course of a short period of time, the tendency is to consider all the messages equally important (and therefore, unimportant!).

I physically cringed every time our monitoring system became an unrepentant peddler of falsities, as I knew it was resetting my team's expectations for "normal." False alarms change the meaning of the message from "this is important" to "this can be ignored."

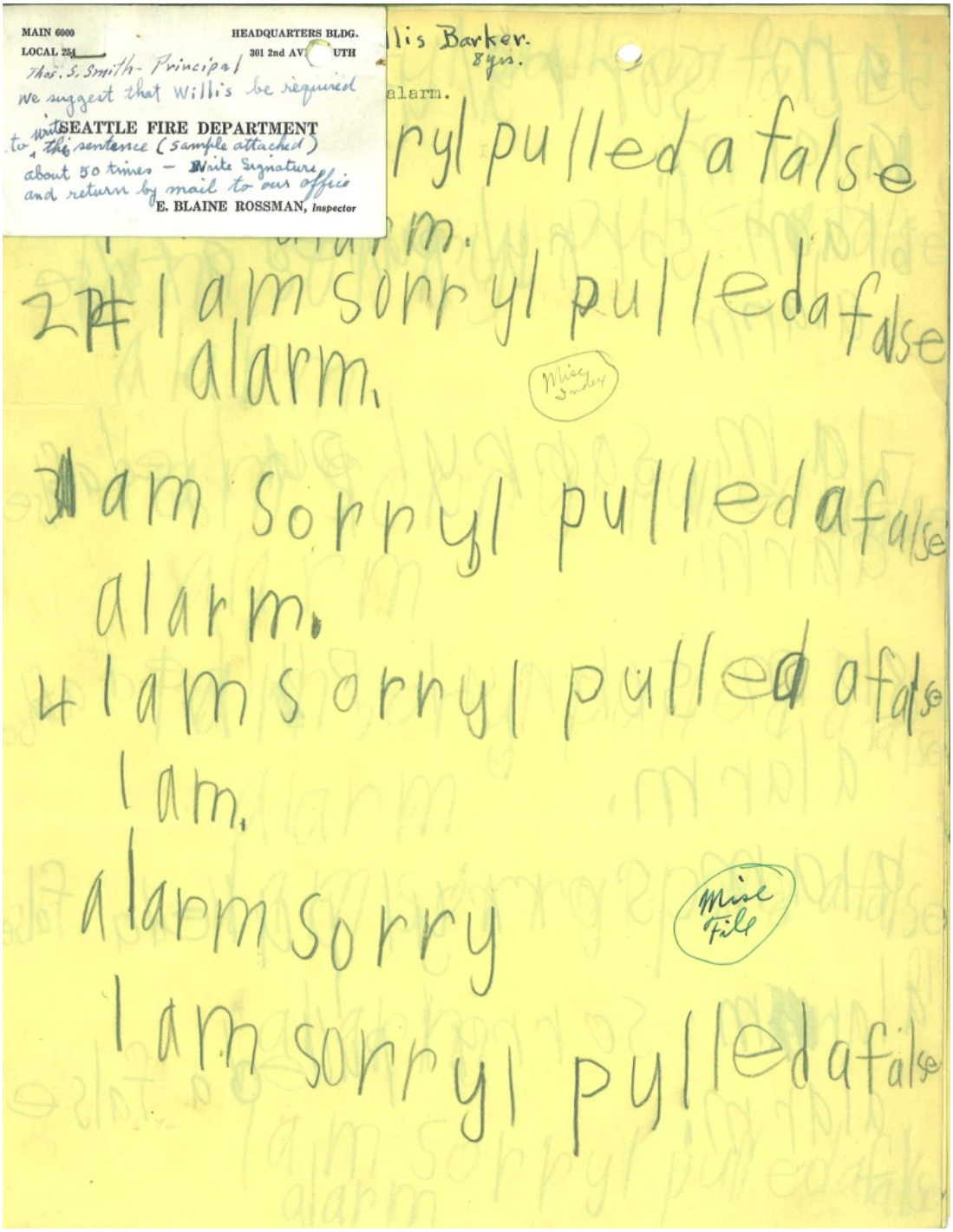
Is it a crisis or do I just need an oil change?
(image: Open Clip Art)

Speaking of meaning, be careful about overloading your alerts. The venerable Check Engine light, found on the dash of your typical car, is a great example of the problems associated with mixing messages. The Check Engine light used to be a big deal: it meant the car had a serious problem, and ignoring it could lead to "chuck engine." However, at some point, it became fashionable for some automobile manufacturers to turn on the Check Engine light for much less urgent reasons, among them the need for scheduled maintenance.

There's nothing wrong with bringing the need for routine maintenance or a loose gas cap to a driver's attention, but encumbering the Check Engine light with these multiple meanings leads to all the problems previously discussed. Overloading is not quite a false alarm, but the effect is similar. Diluting a warning's potency predictably leads to indifference, especially when the statistics favor the less urgent problems. I used to take the Check Engine light *very* seriously, but these additional meanings have lead me to be nonchalant about its appearance on the dash. A

catastrophic engine meltdown is a comparatively rare event, so if the warning can mean either “crisis” or “get an oil change,” chances are it’s the more benign explanation. Therefore, I have reset my expectations accordingly. A better solution would be to preserve the Check Engine light’s meaning of emergency by adding a separate alert for those less critical issues.

In summary, be aware of the effects that permanent and false alerts have on your team. Always aim for a 1:1 relationship between triggered alarms and consequences felt. If your processes require constant vigilance, like those required for aviation or medicine, it’s far better to distill warnings into action-oriented processes and routines (see: checklists). Alarms should always result in *doing*, and it’s even better if the actions to be taken are clearly laid out and practiced beforehand.



Sorry about the false alarm. Can I go outside and play now?
(image: Seattle Municipal Archives / CC BY 2.0)

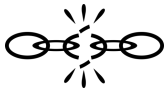
References:

- Header image: Dave Phillips, photographer. Retrieved from Unsplash, https://unsplash.com/photos/Q44xwiDIcns.
- [1] *Consulting News Line*. "Interview: Diane Vaughan" May, 2008.
- [2] Diane Vaughan. *The Challenger Launch Decision: Risky Technology, Culture, and Deviance at NASA* (Chicago: University of Chicago Press, 1996), pg. 414.
- [3] Department of Homeland Security, "Chronology of Changes to the Homeland Security Advisory System."

The Boy Who Cried Wolf was originally published January 29, 2015.

Notes:

Did It Ever Work?



> I have not failed. I've just found 10,000 ways that won't work.
>
> **Thomas A. Edison**

As a troubleshooter, you have one big advantage over your fellow engineers and inventors: the things you are trying to repair **worked at some point in the past**. Inventors dream up new forms, which engineers take and adapt to a myriad of novel contexts. Manufacturers then replicate these models, putting them in the hands of the masses. After the many hurdles required to finally get a product on store shelves, the question of "Will it work?" has likely been answered in the affirmative.

Knowing that a machine once worked means repair can focus on restoring that ideal. Relying on these existing conceptual models, the prior efforts of inventors and engineers become assumptions that can pragmatically be taken for granted. Economics drives this expediency: it would be costly to revalidate the long chain of reasoning that led to a machine's creation (scientific principles, marketplace conditions, design choices, testing, etc.) every time you wanted to make a repair. Put another way, it's much easier to simply assume that you can fix it.

Check out your local Flugtag for creative examples of machines that aren't supposed to function.
(image: brandsteve / CC BY 2.0)

However, there's a critical distinction between a machine operating as its designers intended and its ability to meet your needs. A laptop from 1986 may boot up just fine, but that's little help if you want to run modern applications. A system's operational status and its capacity to do useful work (from your perspective), are two equally important, but different dimensions.

Therefore, my personal troubleshooting script now includes challenging both of these notions, now and in the past. A malfunction is an invitation to reconsider the underlying need and the way it was being served by a particular system. Along these lines, one of my favorite ways to start an investigation is by inquiring **"Did it ever work?"**

I'm often surprised at the answers I get to this question.

Slipping Past The Guards

> My name is on the building.
>
> **Henry Ford II**

Let's consider the vetting process that manufacturers use to ensure their wares will be suitable for a given purpose and accepted by their customers. For companies that design their own products, there's usually a whole array of people, processes, and facilities to get all this right: prototyping, specification reviews, testing labs, quality assurance engineers, statistical sampling techniques, proving grounds, focus groups, etc. A company's reputation and the profit motive drive their adoption: businessmen don't want to damage their company's image or the bottom line by releasing a shoddy product. Also, in an environment where information is free-flowing, things that work well tend to be rewarded with sales. People talk and word gets around, good or bad.

While the free market system creates incentives that reward good products and punish bad ones, the process is a feedback loop that takes time (and requires your participation!). If a bad product slips through the fine mesh of a company's vetting process and then is pumped up by a well-tuned marketing machine, you may be the recipient of something that truly doesn't work. I think the worst experiences I've had along these lines is when I was growing up. During Saturday-morning cartoons, the commercials for toys made them look so amazing! More than once I saved my money or cajoled my parents for some new toy, only to bring it home and have it break shortly thereafter. More often the disappointment was emotional: that thing I had coveted from afar failed to live up to the grandiose dreams fueled

by those slick ads.

Unsafe at this speed? ***Not every design is a good one…***
(image: The Library of Virginia)

Great Expectations

It's a subtle point, but *expectations* often color the response to "Did it ever work?" This is why I find it such an interesting question to ask while troubleshooting. In response, I will frequently hear complaints that indicate the original purpose for which a machine was acquired remains unfulfilled.

As CTO of Discovery Mining, I was the person responsible for outfitting a large portion of our business infrastructure. This was one of the most satisfying parts of my job: to identify a pressing need and then find the perfect *thing* to meet that need. To this end I bought desks, computers, fans, phones, printers, lightbulbs, toner, extension cords, surge protectors, routers, switches, cables, hard drives, tools, and lots of take-out food. I had a bizarre passion for product research, spending hours poring over spec sheets, diagrams, and feature lists to find exactly the right matériel to move our business forward.

However, as thorough and careful as I was, not everything I bought turned out to be effective. For example, early on when we still built all of our computers by hand, I outfitted our servers with removeable hard drive enclosures. The idea was to make replacing hard drives easier, avoiding the hassle of taking the server off the rack and opening the case. Well, that was the theory anyway. We eventually discovered that this particular brand of enclosure was unreliable and would cause disks to go offline. When managing storage in a RAID (Redundant Array of Independent Disks) configuration, having drives go AWOL was a big problem, as the rebuild times could last days and significantly slow a server down. More frightening was the risk of data loss: I had several white-knuckle moments when I wasn't sure if an array was going to be recoverable. My expectations were that the enclosures would make swapping drives easier *and* be just as reliable as before. This turned out to be a wrong assumption. At the time, if you had asked me if they worked in the way that I envisioned, I would have said "Um…not really."

Sometimes, I would buy something for our employees and follow up later, only to find that my purchase really hadn't improved things. Often this was because there was a mismatch between what I heard and the actual needs of the person I was trying to help. Another variation on this theme was that a machine would have been sufficient for a given purpose, but our training program was inadequate, and so it wasn't being used to its full potential. Lastly, I've seen

mismatches between how a system is being utilized and its true capabilities: the perception may be that it "doesn't really work," but that's only because the machine is being asked to do something it wasn't designed to do.

Be careful what you choose to repair, lest you waste your time on something that never really worked in the first place…
(image: Belmiro de Almeida / Wikimedia Commons)

It Worked A Long Time Ago In A Galaxy Far, Far Away…

> Past performance is not an indicator of future results.
>
> **The Tiny Print From That Thing You Lost Money On**

For those who employ machines to do their bidding, the passage of time results in an ever-shifting context that adds further complexity to the problem of recreating a stellar past performance. To illustrate, I like to think of a favorite pair of jeans that I once owned: this perfect specimen of denim had been patched and repaired so many times that I called them my Franken-jeans. The answer to the question "Did they ever work?" was "Yes!" However, that doesn't mean these pants could be restored back to their original state—too many threads were missing.

The point is that even if a machine worked well in the past, that doesn't mean you can (easily) return there. Classic car restorations are a great example of this dilemma: sure, that rusted '57 Chevy may have worked in the past (specifically, in '57). However, the ravages of time make the prior accomplishments of machines little more than historical footnotes.

It might still work, but that doesn't mean you can use it to watch YouTube.
(image: SDASM Archives)

Guard Your Time

Questioning if something ever worked is an important way to protect your most precious resource: time. At worst, fixing something that never really functioned can be a quixotic quest; at best, it's not even repair, lying somewhere between engineering and invention. This line of inquiry can also reveal the unfulfilled purposes of the people you are trying to help, a problem that can be wholly independent of a machine's functional status. As I've found time and again, assessing and then aligning yourself with that underlying need is the sure path to avoid wasting your effort.

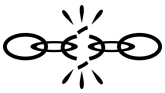
References:

- Header image: Harris & Ewing, photographer. *AVIATION, ARMY, COLLEGE PARK. TESTS OF CURTISS PLANE FOR ARMY. SINGLE CONTROL.* United States, College Park, Maryland. 1912. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016863983/.

Did It Ever Work? was originally published June 18, 2015.

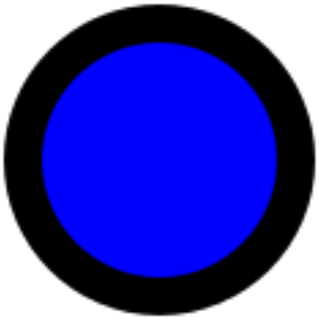
Notes:

Network Effects



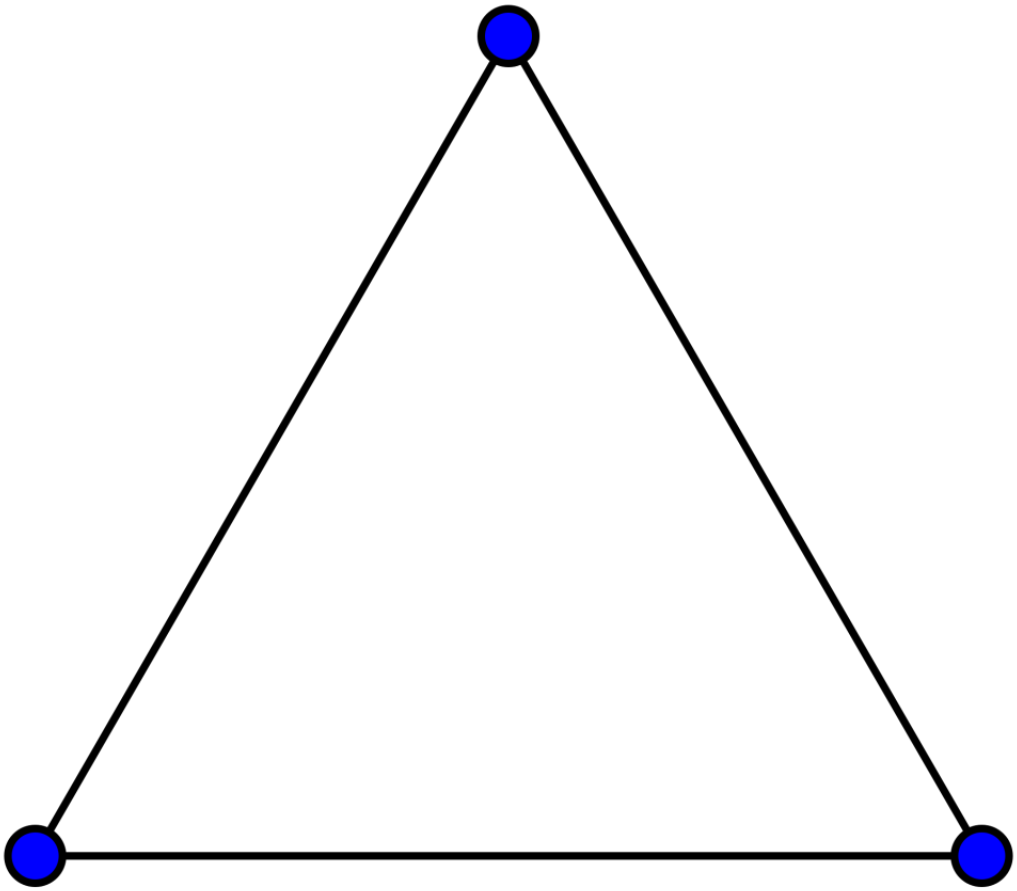
> The Web as I envisaged it, we have not seen it yet. The future is still so much bigger than the past.
>
> **Tim Berners-Lee**

It all started with just one:

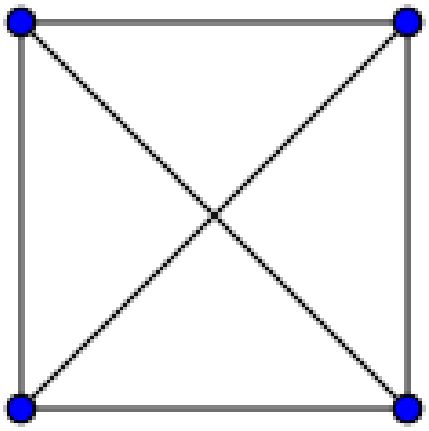


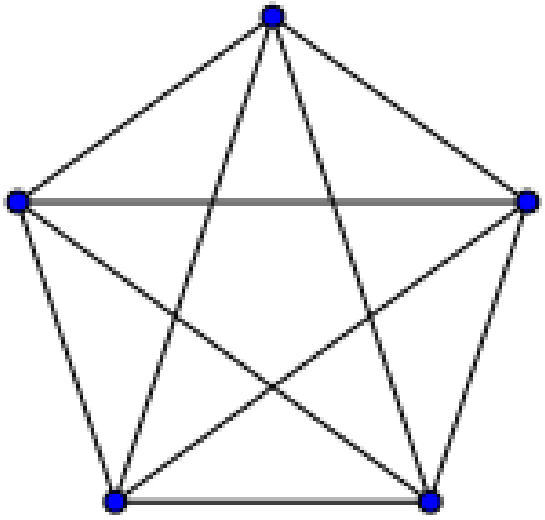
Pretty lonely, huh? Then another showed up:

That's better. A third joined in:

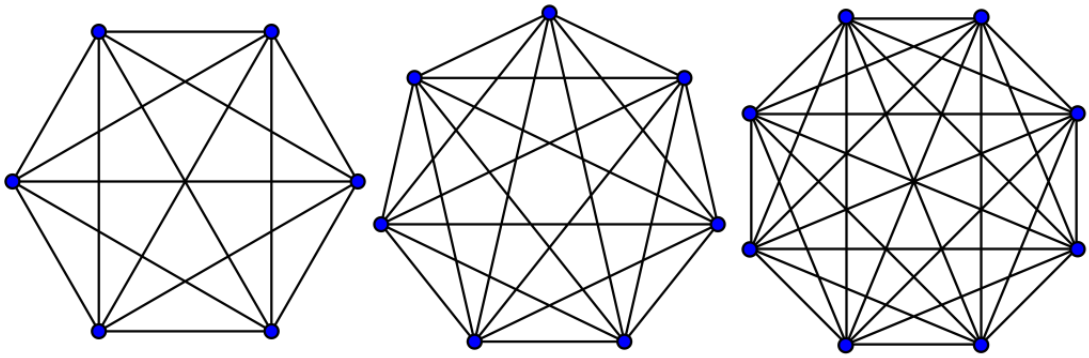


A fourth and fifth too:

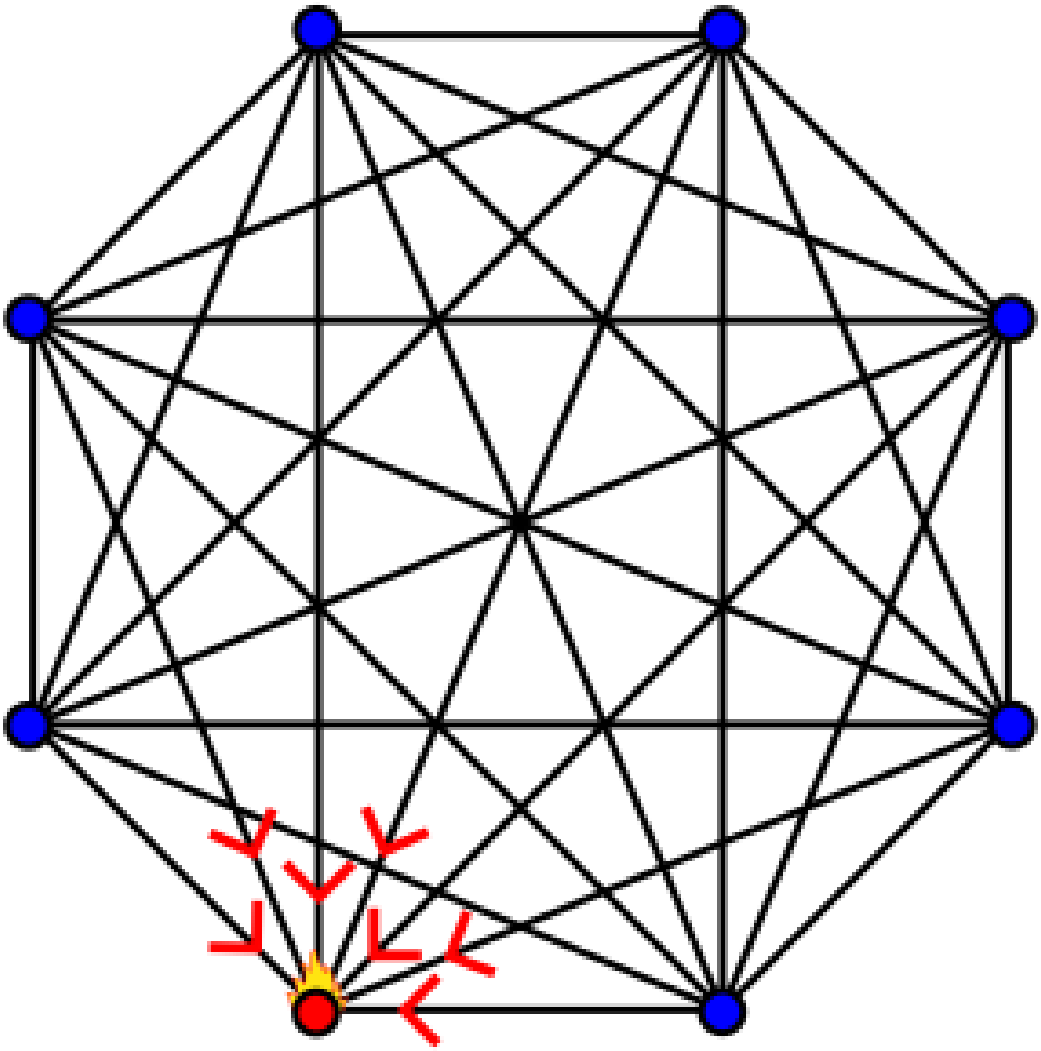




Then six, seven, and eight really opened up the possibilities to connect:



Then, one got a little too popular and things started to heat up:



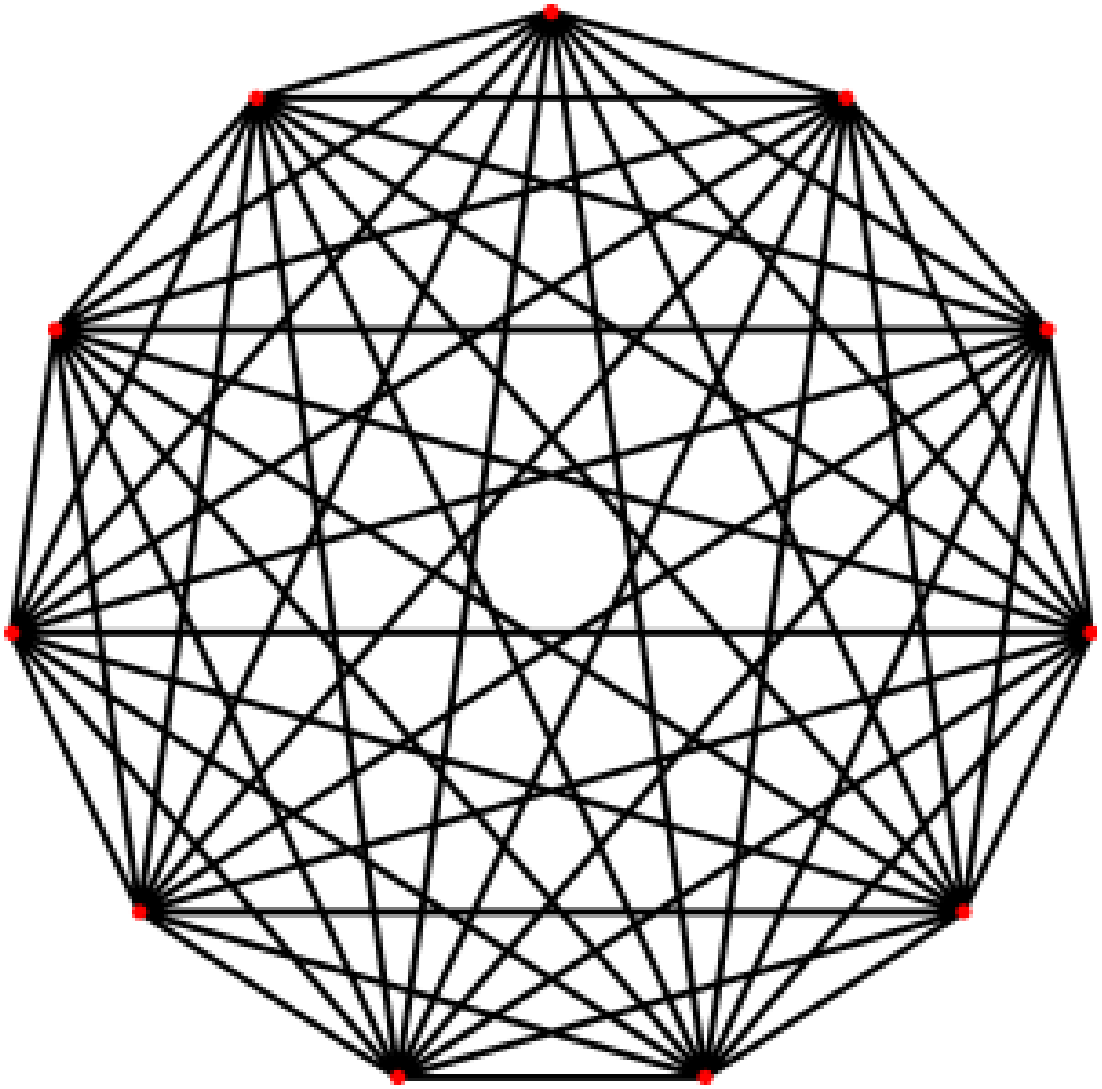
It's a familiar story…

Counting Connections

Highly interconnected systems are among the most important of our modern industrialized civilization. To name just a few examples: roads, the Internet, pipelines, electrical grids, financial markets, and waterways. If you somehow don't rely on those and think you're not involved, we should also include human organizations too: businesses, clubs, governments, churches, families, and social circles. You will see how the growth of the connections within these networks are a double-edged sword: they simultaneously make a system more useful and increase the likelihood of

congestion.

Let's start the discussion by introducing the Complete Graph, a powerful model whose form I've encountered again and again in so many different contexts. Whatever kind of systems you troubleshoot, there's likely a portion that resembles this form. The Complete Graph is a mathematical term for a model where every vertex is connected to every other vertex (mathematicians call these connections **"edges"**). Visually, they look like the images in the introduction to this essay. Here's a Complete Graph with 11 vertices:



The number of connections in a Complete Graph can be described mathematically using the standard formula for combinations (valid for $n > 1$):

$n! / (n\text{-}2)! \times 2!$

Alternatively, this can also be expressed as:

$(n^2 - n) / 2$

You can see that there is an exponential factor (n^2) in this formula, which will come to dominate its scaling as n grows. In other words, the total number of connections will expand much quicker than the number of vertices. How fast will the edges grow? Let's look at some numbers:

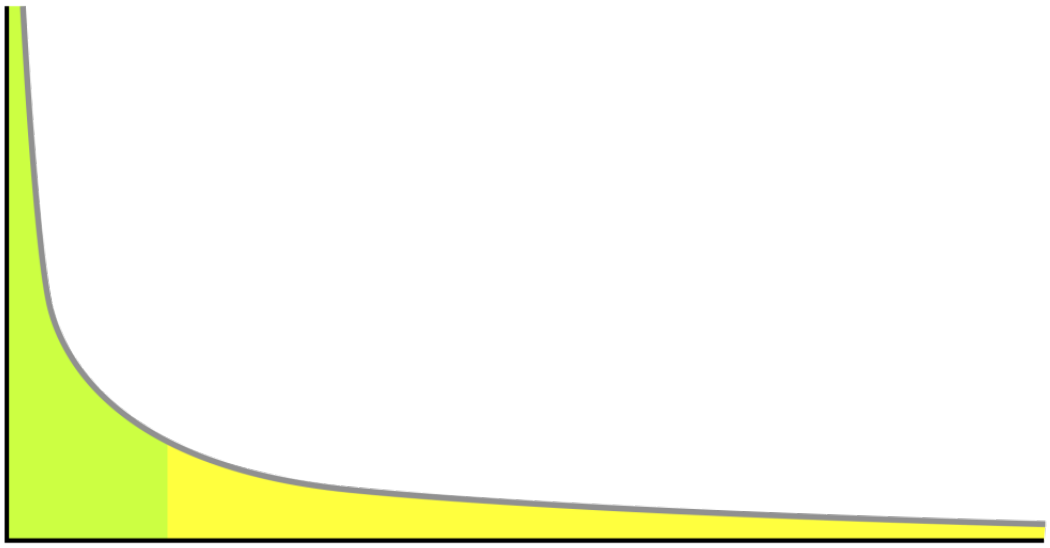
# of Vertices	# of Edges	Ratio (E::V)
1	0	0

2	1	0.5
3	3	1.0
4	6	1.5
5	10	2.0
6	15	2.5
7	21	3.0
8	28	3.5
9	36	4.0
10	45	4.5
100	4,950	49.5
1,000	499,500	499.5
10,000	49,995,000	4,999.5
100,000	4,999,950,000	49,999.5
1,000,000	499,999,500,000	499,999.5

The important thing to notice in the table above is the growth in the *ratio* of edges to vertices. The reason why this matters is, in any networked environment, the vertices are typically not equal, nor are the connections among them random. If the Complete Graph represents the people working in a corporation, only one is the CEO; if it's a network of roads, there might be only a few streets that go into a giant stadium's parking lot. Therefore, a theoretical understanding of network effects is usually paired with the Pareto Principle, which you might have heard as the "80/20 rule."

The economist Vilfredo Pareto noticed that 80% of the land in Italy was owned by about 20% of the population. Pareto surveyed other countries and noticed a similar distribution of ownership. Since then, this same pattern has been observed in countless other contexts from sales ("80% of sales at company XYZ comes from 20% of its customers") to health care spending ("20% of patients consume 80% of healthcare resources").

Now, there's nothing magical about the numbers 80 and 20, they're just representative of the Power Law distribution. Visually, it looks like this:



The long tail: whether known as the power law, the Pareto Principle, or the 80/20 rule, the distribution of many naturally occurring phenomena take on this lopsided form.
(image: Husky / Wikimedia Commons)

The power law is found everywhere in nature*, from the magnitude of earthquakes to the size of craters on the Moon. When it comes to human-based activity, we find the form repeated in such wide-ranging contexts as contributors to

Wikipedia and the population of cities. For my first "real" job after college, I was hired by a company called Alexa to analyze data on how people were using the Internet. There, I discovered the power law by accident: I remember graphing the distribution of traffic amongst the top websites, messing around with the scale of the axes. I changed them to logarithmic and suddenly the relationship turned into a straight line!

I stood back and said, "whoa." I didn't know what it meant, but I thought that a straight line was a bit weird. There was an engineer in another department who had a PhD in physics, so I printed out the graph and marched over to his office. He looked at it, paused, and said "Power Law. Look it up."

I did.

(*If you're interested in learning more about the Power Law and lopsided distributions, see my study "Tube Of Plenty: Analyzing YouTube's First Decade," where I analyzed the metadata of over 10 million YouTube videos.)

Sorry, but I don't have enough wire to connect you directly to everyone else. You'll have to go through the local exchange…

(image: Russell Lee / Library of Congress)

Call Me, Definitely

Before we get into the downsides of interconnected systems (and hence the need to troubleshoot and optimize them), we should first recognize their immense benefits: being able to deliver information, money, fuel, water, electricity, packages, or people between arbitrary points is a huge benefit to humanity. Most of the stuff that sustains our lives flows through networks of some kind.

One of the widely cited theories about why networks become more useful as they grow is Metcalfe's Law, which states that "the value of a telecommunications network is proportional to the square of the number of connected users of the system." Metcalfe is the co-inventor of the ubiquitous Ethernet networking standard, which is the foundation for most of the data networks in existence today. Some people have criticized the way Metcalfe's Law has been used, and likewise that Bob Metcalfe's own formulation and intent was narrower (after all, maybe he was just trying to get people to buy Ethernet gear). Also, the "value" the law cites isn't given in any measurable terms.

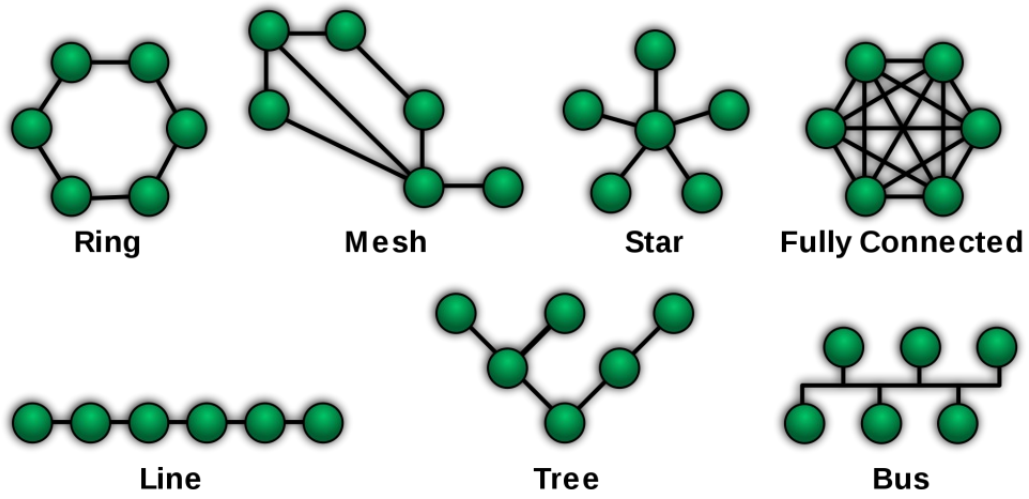
However, I think the underlying principle behind Metcalfe's Law is easily understood and useful. Imagine if there were just 2 telephones in the world: sure, they would presumably be useful to the 2 lucky people who had them, but it's nothing compared to what you can do if there are billions of phones. You can order a pizza, alert someone you're going to be late, book a hotel, ask out that good-looking girl or guy, or get a taxi to pick you up. What you can *accomplish* with a phone increases dramatically when everyone you want to communicate with has one. Plus, each new phone that joins the network makes the entire system even more beneficial, allowing more potential contacts to happen.

Other Ways To Connect

There are also counter-trends that frustrate the value and existence of networks that try to expand indefinitely. The "value" isn't the only thing that increases with the number of interconnections: congestion, maintenance costs, fraud, theft, and a whole host of other negative effects are also likely to appear. Even though an "any-to-any" concept of a network may be marketed to end-users, scaling becomes its own challenge because the actual implementation requires compromises between connectivity and costs.

For example, my computer might theoretically be able to contact every other device on the Internet (just like the Complete Graph would suggest). However, there isn't a cable going from my computer to every other computer on the planet: the Complete Graph has little to do with how the Internet is actually constructed. This is because the Complete Graph is a very expensive model to actually use when building a large interconnected system.

On that note, let's introduce some of the more common topologies that you'll see when observing networks in the wild:



(image: Malyszkz / Wikimedia Commons)

Let's go through each of these network types, thinking about how they solve certain kinds of problems and find examples in the real world:

Ring

- **Pros:** each node has two options for egress. Should one of the links be severed, connectivity between nodes is still possible by traversing the network in the opposite direction.
- **Cons:** each node is part of the network's path. If a node or conduit malfunctions, the throughput of the network is immediately halved and the result is a line network.
- **Examples:** one of the ISPs I used for business had a ring network connecting all of their data centers in the Bay Area. If one of these connecting links went offline, data would continue to flow around the ring in the opposite direction.

Mesh

- **Pros:** multiple pathways through the network improves redundancy and throughput.
- **Cons:** additional interconnections come at a price. Also, because each node can potentially be used as a conduit, automated smarts are needed to make the network resilient and efficient.
- **Examples:** a popular model for wireless networks, because adding additional links between nodes only requires proximity (i.e., there are no cables to lay). Meshes are harder to implement for physical networks, because of the increased cost.

Star

- **Pros:** centralized control and isolation of nodes.
- **Cons:** the center node is an obvious point of congestion, as all nodes must use it to reach any other point on the network.
- **Examples:** the hierarchical relationship of employees to their manager is a type of star network.

Fully Connected (aka, "full mesh")

- **Pros:** this model wins the award for redundancy! Because every node is directly linked to every other node, the removal of one will have no impact on material attempting to traverse the remaining paths.
- **Cons:** expensive to implement, with exponentially rising costs as the network grows. Also, because each node can be contacted directly by all the others, congestion problems can show up at any node. This is in contrast to more conventional networks, like the star topology, where the predictable choke point can be strengthened.
- **Examples:** large, fully connected networks are extremely rare in an industrial context (because of their cost). However, you have direct experience with them. Most people belong to social networks (like their immediate family or a tight-knit group of friends) that operate like a fully connected network.

Most pipelines can be conceptualized as unidirectional line networks (occasionally, their flow is even reversed).

(image: Carol M. Highsmith / Library of Congress)

Line

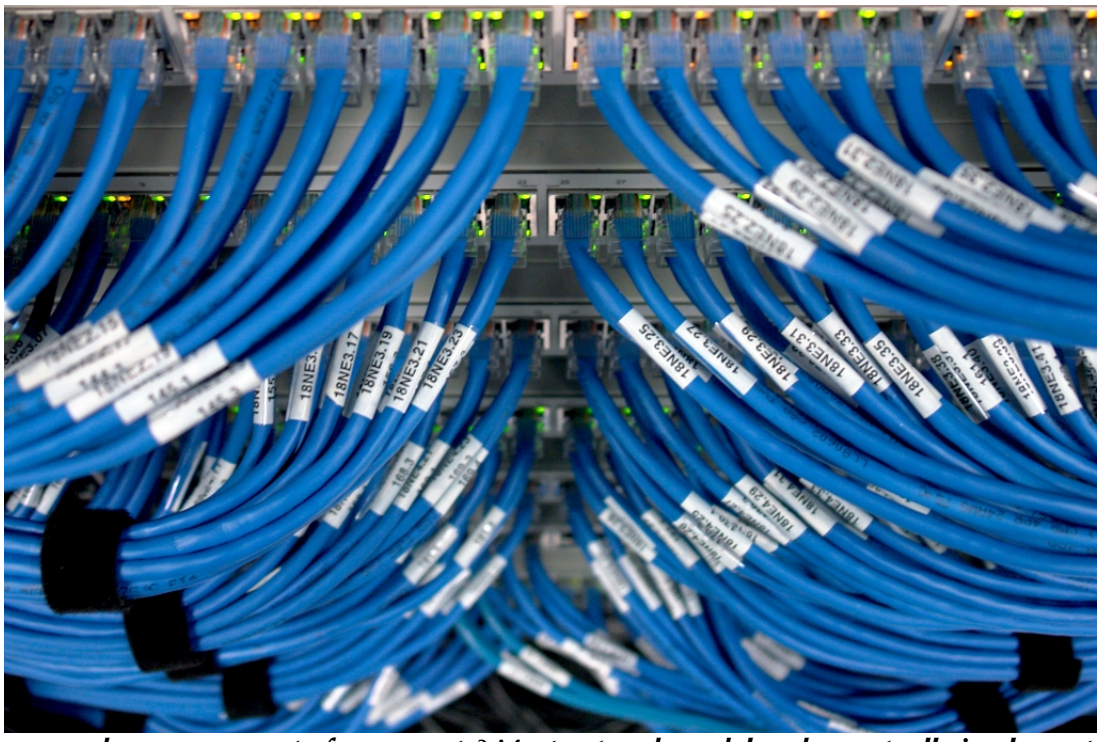
- **Pros:** like the ring, every node can pass material in either direction. Also, because each node only needs to be connected to the next closest node, this can be an inexpensive network to implement.
- **Cons:** every node needs to be operational to pass material through the system from end-to-end. Also, because there is just a single path of flow, the severing of a link will cut the network in two parts.
- **Examples:** oil pipelines. Or, a particularly hilarious application you might have played at a party: the "telephone game."

Tree

- **Pros:** combines the star form with other topologies, leading to more manageable clusters of nodes, which then connect to other parts of the network via trunks.
- **Cons:** like any model that uses single links to connect parts of the network, a severed trunk connection will leave the network fragmented.
- **Examples:** hybrids like the tree network are the model most likely to be found in real-world use. Most organizations that create their own networks will (eventually) end up with some kind of tree structure. An example that immediately comes to mind is any large airline's route map (my favorite part of any in-flight magazine), usually having several hubs ("spoke and wheel" star networks) that are connected by flights between them (trunks).

Bus

- **Pros:** this might be the cheapest of all the network types to implement. Links only need to make it to the nearest part of the shared bus to connect to the network.
- **Cons:** like the line network, a shared conduit limits throughput. Severing of the bus at any point can leave the network split in pieces.
- **Examples:** early Ethernet networks using 10BASE2 technology.



What happens when you run out of open ports? Most network models, when actually implemented, come with scalability limits.

(image: Andrew Hart / CC BY-SA 2.0)

Too Many Requests

> A large-scale concert on sale is, in essence, a denial-of-service attack.

> **Andrew Dreskin, Ticketfly CEO**

There's a downside to all this interconnectedness: what happens when the whole world shows up at your doorstep? When it comes to scaling networked systems, it may be easy to add a node by redrawing a graph on paper, but your actual implementation may not be so easily adaptable. The process of taking an abstract idea about how a network *should* function and putting it into physical form inevitably includes compromises about capacity. If there were no tradeoffs or cost, you'd gladly have your house plumbing sized to run a large waterpark, or your Internet connection transfer at the rate of a gazillion bits-per-second. However, capacity isn't free and so every chosen network implementation must reconcile our infinite desires with our finite means.

A note about "troubleshooting" network problems: relieving congestion by adding resources or changing network models can easily cross over into the land of engineering and invention. Recognizing the distinction isn't a slavish devotion to semantics, but is instead about recognizing the limits of your equipment. Sometimes, a piece of gear can be operating *exactly* as designed and still won't be able to handle the load generated by the network to which it is attached. In these cases, the machine isn't broken, it's just being used in a context that isn't productive. What has likely changed is the circumstances surrounding its use (often, the volume of stuff being sent through the network).

The pure "troubleshooting" answer to network model problems would be: the system wasn't designed for this level of usage, so you should *decrease* usage. You may think this is an unsatisfactory answer, but this is actually a bona fide solution. I've seen smart business owners do this, refusing work rather than inviting the chaos of running at overcapacity. I always appreciate when a maître d' at a busy restaurant simply turns me away, rather than seating me for a never-ending meal à la *Waiting for Godot*.

People do this in their personal lives all the time: there's simply a limit to the number of close social connections you can maintain. Partly, this is because maintaining relationships takes the scarce commodity of time, but there might be another limitation: our minds. Robin Dunbar, the Oxford anthropologist and creator of the eponymous Dunbar number, accurately predicted the size of various primate social groups by looking at the size of their brains. Using this method, he then turned his sights on us, predicting that humans would have social circles of approximately 150 people:

> The essence of my argument has been that there is a cognitive limit to the number of individuals with whom any one person can maintain stable relationships, that this limit is a direct function of relative neocortex size, and that this in turn limits group size.
>
> **Robin Dunbar, Co-evolution of Neocortex Size, Group Size and Language in Humans**

These mental limitations to our social circles appear to hold even in the era of FaceBook and Twitter, with Dunbar noting that "when you actually look at traffic on sites, you see people maintain the same inner circle of around 150 people that we observe in the real world." While we may have to tacitly accept these limitations in our social lives, the profit motive and the desire to constantly reach for new heights makes us rarely satisfied with the troubleshooting answer of "do less" in other contexts. For a business owner, it will gnaw away at you knowing that there is business out there that you can't satisfy. This compulsion drives us to grow our networked systems by swapping out the underlying model with something more suitable.

To prime our thinking about scaling networks, a good place to start is with the public switched telephone network. Let's consider the Central Office, a building block of the phone system (still in use): this is the physical place where all the phone lines in a locality are terminated. There are practical limits to the length of phone cables, both because of the signal degradation that sets in over longer distances, as well as the expense of stringing them along poles or through underground conduits.

Therefore, you'd expect that the size of a typical Central Office (CO) would be limited by these geographical and economic constraints. You're not going to be able to have a single CO for a whole country (except for maybe Monaco or the Vatican). If you were laying out a new city, you'd need to plan to have a certain number of these facilities: X number of COs per Y square miles of land. You can also predict that the type of equipment deployed in a CO would

be of a certain scale, designed perhaps for thousands, but certainly not millions of incoming lines. Why? Again, economics. The industry will consolidate around certain implementations and designers will take advantage of these economies of scale. Put another way, if everyone else is building COs of a certain size because the available equipment favors it, then you will too.

While many systems use the abstract network models shown above, seemingly flexible and conducive to scaling to any number of nodes, their in-the-flesh implementations will typically only function within a fairly narrow ratio of edges to vertices. The problem is that networked systems are deployed in contexts that are ever-changing: the ebb and flow of businesses, organizations, families, cities, and states.

Scaling Inside The Existing System

To understand the scaling of networked systems, let's start with the low-hanging fruit of their built-in capacities. Think of a simple electrical power strip with, let's say, 6 outlets:

(image: Malvineous, license: CC BY-SA 3.0)

Conceptually, when you plug devices into a power strip you form a star network. Plugging in the power strip itself then creates a tree network. The scalability is built-in, yet limited: 1 to 6 devices can be attached to this particular power strip. Let's say you buy one and plug in your laptop, monitor, printer, scanner, modem, and router. It's a full house!

Then, you go on a shopping spree and buy a stereo and a space heater. Hmm…you can see this presents a problem. Previously, we made a choice that lead to a purchase and the result was a *fixed* amount of scalability. Even though a power strip might be a star network in the abstract, you can't just draw lines and magically have more outlets appear. Maybe you buy a second power strip, raising your office capacity to 12 outlets. That's great, but this method of scaling can't continue indefinitely. The power strips themselves are plugged into a circuit that has a fixed capacity.

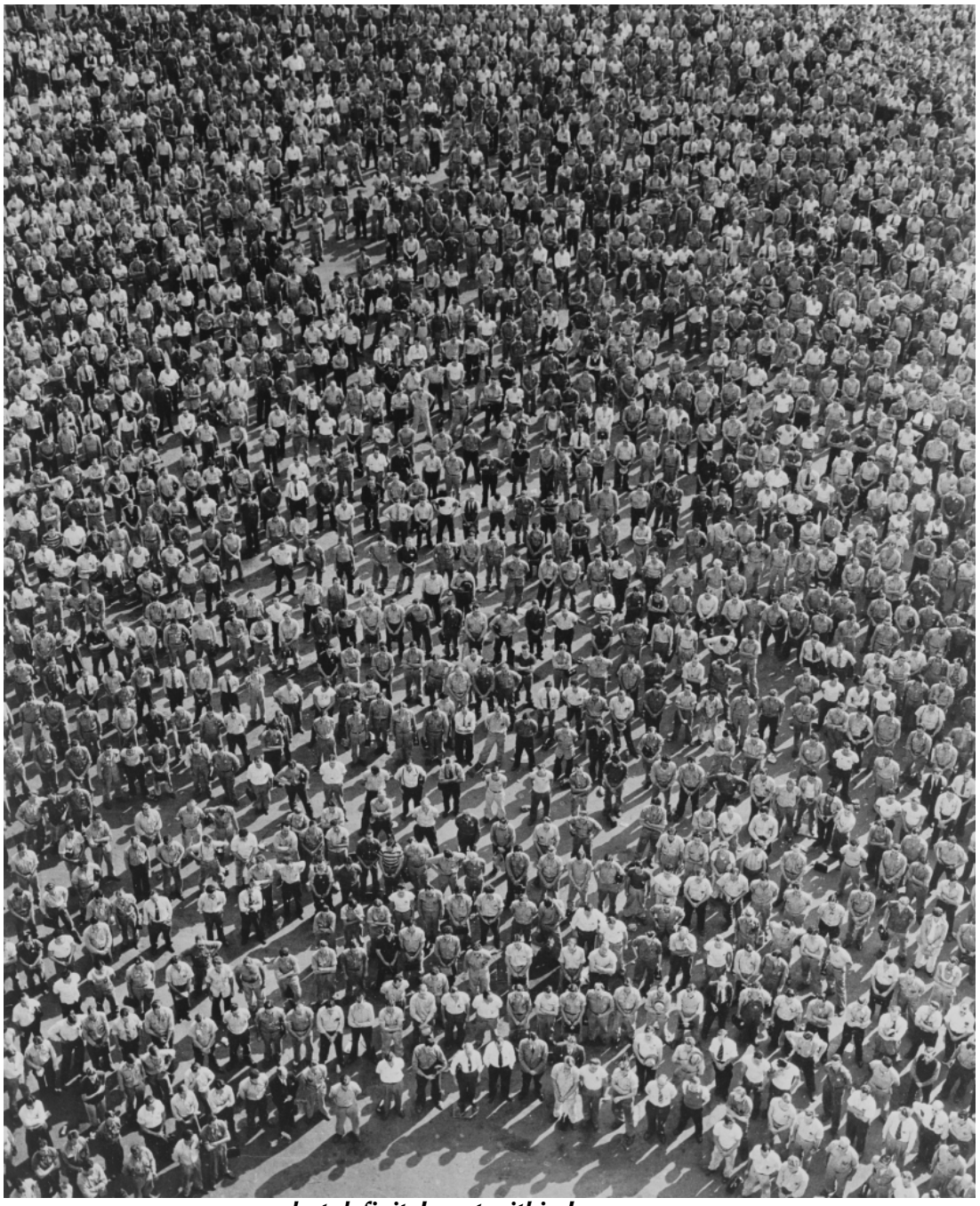
From this simple example, you can see that networked systems, when actually implemented, will have these scaling thresholds (or "cliffs," should you be pushed over them). There will usually be a range within which it's easy to scale, but after that it will require more resources. Beyond that, you will eventually come to another breaking point where adding more of the same will not be enough to increase capacity. At that point, you'll need to rethink the underlying model.

Within small teams, perhaps you can meaningfully keep up with everybody…
(image: Library of Congress)

Scaling Outside The Existing System

Human-based organizations are a great example of how network models need to be discarded and remade, especially during a period of growth (or decline, as the case may be). What works for a solo enterprise will not work for 10 people, what works for 10 people won't work for 100, what works for 100 people won't work for 1,000, and so on. I'm not saying that the network model exhaustively explains every reason why, but it is a powerful framing of the problem.

For very small groups, it may be possible to have a "fully-meshed" network model. Think of a small business, like the auto repair shop my Grandfather used to run. It was just he and a partner. With only a few people working together, it's definitely possible for everyone to interact with everyone else. If you share the same physical workspace, you can be privy to the same information and develop a meaningful personal relationship with everybody (and they with you), approaching the ideal of the fully-connected network envisioned in the Complete Graph model.

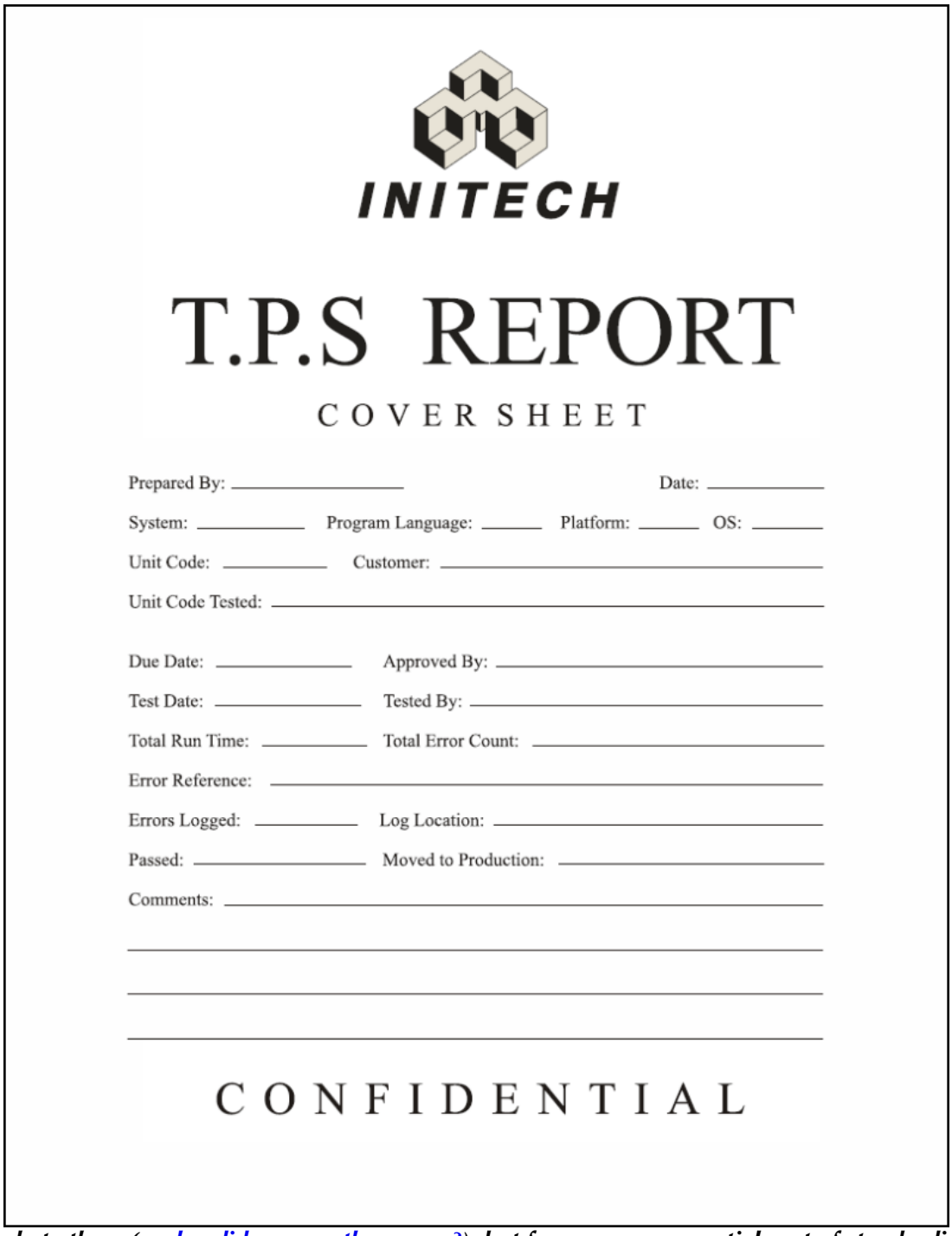


…but definitely not within large groups.
(image: Library of Congress)

A fancy piece of new equipment glistens in the sunshine. You may bask in all of its chrome-reflected glory. But, no matter how many rays are reflected, its capacity is finite. It's also likely to be explicitly stated, usually somewhere in the manual (albeit in the small print). While these technical limitations are known and respected, our human ones are frequently disregarded (to our detriment).

For the sake of productivity (and everybody's sanity), as a firm adds people it's a crucial task to migrate to a structure that limits interactions between the parts of your organization's network. The reason why is plain: our time and ability to focus is scarce. Imagine if you, a lone person, had to keep up with all the employee-to-employee interactions inside a group of millions, as in a mega-corporation like Walmart or McDonald's. It simply wouldn't be possible. It would be boring too. Like, really boring.

Isolation may sound like a bad word, but when it comes to your job it's an absolute necessity for you to get anything done. This is one reason why companies create barriers of all sorts, from the physical (e.g., offices and cubicles) to the hierarchal (e.g., you can only access a particular person through their boss or secretary) to the procedural. Along the lines of this last category, I present the form:



You may hate them (yeah…did you see the memo?), but forms are an essential part of standardizing and limiting interactions between parts of an organization's network.
(image: Don Meyers / CC BY-SA 3.0)

The very low-tech form is an example of using procedures to standardize how parts of an organizational network interact with one another. I'm not sure who invented the form, the kind which needs to be filled out in triplicate, but if they didn't someone else would have. Before the venerable form appeared, people were getting requests verbally, scrawled on post-it notes, via emails, voicemails, smoke signals, in skywriting and interpretive dance. Having other things to do, all this free-form access from other parts of the organizational network probably got a bit tedious. Then the clouds parted and the form appeared. Even though the entire group may contact a single employee, those little lines provide protection when they do.

I'm not promoting any particular management structure (flat, functional, product, geographical, etc.), because isolation, hierarchies, and procedures each have their own problems. Further, implementing a particular system will also generate a host of unintended consequences: the end result may be superior, but that doesn't mean that customers, vendors, and employees will all simultaneously be better off. However, you need to be sympathetic to the underlying

argument for the existence and pursuit of these competing organizational systems: the limiting of unwanted network effects.

Growth will present scaling problems to your networks (both human and machine), exposing unsustainable relationships between edges and vertices. Just like our power strip, it may be possible to scale within the existing system. A single manager may well be able to increase his subordinates from 2 to 3, and again from 3 to 4. A router might be able to switch 1,000 mb/s of traffic just as well as 1 mb/s. However, there will come a time when the old ways of scaling will no longer be possible and you must change your network model.

There may be a lot of connections here, but don't be fooled into thinking that the capacity is infinite…
(image: Russell Lee / Library of Congress)

The Physics Of The Network

Any time you centralize something, whether it be information, decision-making, or physical goods, there are going to be tradeoffs. As networks are often the *means* by which these things are concentrated, we should look at their role in these schemes and discuss the pros and cons of bringing things together.

When you move stuff around a network, you lose its original context. This becomes a problem for the transfer of information, especially the kind that is used to coordinate the actions of individuals within a group. If you're a lowly army corporal standing in the general's office and he gives you an order, several important things are quite clear: you know the general is speaking to you and the general knows with whom he's speaking. If you're brave enough, you can even ask for clarification.

But, let's say you are that same corporal, thousands of miles of away in the middle of a heated battle, and someone hands you a piece of paper telling you what to do. At the bottom is the general's name…but how do you know it came from him? Has the order been changed or delayed along its path? You can see that by removing context, networks present problems of trust and identity.

They're fine sitting in the parking lot, but what happens when they all want to get on the road?
(image: John Vachon / Library of Congress)

Another interesting aspect of networks is that their capacities are often small in relation to the type of material they transmit. This is because it's (relatively) expensive to move things around. Walking around San Francisco, I'll often see a crowded street with cars taking up every possible bit of free space along the curb. I've wondered: "What would happen if everyone decided to use their car at the same time?" The capacity of our roadways is small compared to the number of cars in existence. To the extent they are usable in a crowded urbanity, we rely upon the vast majority of cars *not* being in use! Whatever medium you can think of (oil, information, cars), their corresponding networks (pipelines, the Internet, roads) will only be able to transmit a small fraction of the available material at one time.

When it comes to the movement of information on a network, in addition to any context that might be lost, an *abbreviation* must also take place. This has always been true: imagine that you were the commander of a far-flung outpost like Vindolanda, sitting on the edge of the Roman Empire's frontier by Hadrian's Wall. One day, you get a request from your superiors asking you for a report of the goings-on over the last year. There are nearly an infinite number of things you could include in your reply: enemy incursions, movements of troops in and out of the fort, levels of supplies, agricultural output, temperature, rainfall, births, deaths, disputes, details of social occasions, or the latest gossip about Agrippina and Plinius (they were seen kissing behind the Thermae!). But, those wooden tablets aren't exactly easy to write on, plus the courier only has room for one in his pouch and he's leaving in 15 minutes. So, you just scribble the following: "We're all fine. Still alive. Send wine."

Funnily enough, that's probably all they could reasonably handle within the busy context of running a large empire. As a general rule, the nodes on the edge of a network can produce much more data than can be transferred, analyzed, and acted upon. There is an infinite amount to be measured within the simplest of systems and so, any time you collect data, you are gathering only a tiny fraction of what can be known (see "Is This Normal?" for more). Because of the expense of preparing the data and then moving it, centralizing information usually involves a further culling. The core can't know everything about the edges, and the edges can't know everything about themselves.

Lastly, we should note that centralization (via networks) is often pursued as a measure of control. Whatever it is you're trying to protect (information, resources, etc.), the theory is that it will be easier to do so when it's all in one place. The adage "don't put all your eggs in one basket" neatly summarizes the potential downside of concentrating important things.

Let's make some new connections.
(image: btphotosbduk / CC BY 2.0)

Breaking And Making New Connections: Reforming Congested Networks

Whether we're talking about Dunbar's Number or the availability of open ports in a telephone switch, the common theme for system builders is: be aware of the scalability thresholds that limit the size of the networks under your care (and of which you are a part). When the easy options for expansion have been exhausted, here are some basic strategies for mitigating network problems:

Augment

This is the most obvious path to increase your network's capacity and, because it's along the lines of what you already know, will be the most tempting. If your Internet router is overloaded, you can buy one with more capacity. If a 4-lane highway is constantly jammed, you could widen it to 12 lanes. Whatever your network currently does, you can imagine replacing the component parts with ones that simply do more.

Prioritize

A network is a finite resource, so you should ask yourself if it is being used for the most urgent or valuable purpose. If a network is being utilized by customers, maybe you need to raise prices to bring usage under control. Toll roads are an easy to understand scheme that reduce congestion by giving access to those who value it most. It doesn't have to be about the bottom line: whatever your organization deems important, you want to make sure that your networks are similarly aligned.

Re-model

This is where you alter the network model, adopting a new form that changes how the various nodes connect to each other. Modifying the ratio of edges to vertices can go in either direction: congestion may lead you to seek ways to lower this number, but efficiencies can allow you to raise it too. Consider an overloaded manager, connected to subordinates in a typical star network: a common strategy is to split up teams that are too large, appointing additional leadership for the new groups, lowering the ratio between employees and managers (effectively creating a tree network in place of the former star shape). You can also imagine this going the other way: maybe a new HR system automates many of the tasks that used to burden a particular manager (scheduling, reporting, etc.), leading to an *increase* in the number of employees under their direction.

Isolation and exposure are two more opposing themes that can drive the remodeling of your networks. We've talked

about the need to limit access to nodes, especially when the context is an organization. There simply aren't enough hours in the day for the CEO of a mega-corporation to be continually and directly accessed by everyone who works there. Even when you're not the CEO, you probably have appreciated those jobs where you were left alone to focus on getting things done. Devolutions done in the name of isolation can transform star networks into trees, pushing nodes further away from the core.

Network reformations can also promote the opposite, exposing previously hidden nodes by *increasing* access to them. Maybe Bob from accounting is a little too reclusive, so you publish his telephone extension and office number in the employee newsletter. In computing, wireless mesh networks attempt to promote connectivity between nodes by allowing information to flow freely among them. Tear down those walls!

Not designed to work efficiently with 20-ton boulders.
(image: Russell Lee / Library of Congress)

Standardize/Limit

Within a network, a way to preserve connectivity and reduce congestion is to enforce rules about *how* nodes connect to each other. Remember the mighty fill-in form: it all comes back to this idea of giving relief to overloaded nodes by placing restrictions on how others can connect to them. Standards for inputs ensure that a given node can process material in the most efficient manner. Another way to ensure a node can operate efficiently is to place limits on the rate it receives materials or instructions from elsewhere in the network.

Be Flexible

Network forms, along with their physical manifestations that we actually use to do work, all have a proper scale in which they can be used. A network's context is constantly shifting within the ups and downs of your life and business. One model cannot satisfy your needs forever, so a fluid mindset regarding their use is paramount.

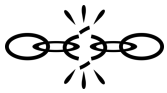
References:

- Header image: Trikosko, M. S., photographer. (1959) W*omen working at the U.S. Capitol switchboard, Washington, D.C.* Washington D.C, 1959. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2013651433/.
- The images of complete graphs used in this article are from the public domain collection on Wikimedia Commons ("Set of complete graphs").

Network Effects was originally published April 11, 2016.

Notes:

On Selfies And Showboating: Troubleshooting The Imminent Dangers Of “Look At Me!”



> He who tries to shine
> dims his own light.
>
> **Tao Te Ching (verse 24)** [1]

A tourist steps closer and closer to a steep cliff, obsessively searching for that perfect selfie. A group of teenagers, trying to impress each other, jump off a bridge into a fast-flowing river. Drivers slow to look at a wreck by the side of the road, taking their eye off of the road.

Whether we’re trying to generate attention, or giving that attention to others, it can distract us from reading the dangers of our current context. Humans are social animals, so it’s no surprise that group recognition is important; however, when this desire is mixed with an unforgiving natural world and the complexities of machines, we can pay a dear price for the pursuit of consideration from our peers.

The modern systems, simple and complex, that we use to accomplish our goals are a *combination* of man and machine. There is no separation: humans are involved at all levels. We determine their need, design, manufacture, purchase, installation, maintenance, and use: that unique combination of personnel and machinery employed to get a task done. Therefore, it shouldn’t be a surprise that a human imprint can be found in the failure patterns of these systems too: social effects are omnipresent in malfunctions at all scales.

The bar for selfies was raised to impossible heights [by Buzz Aldrin in 1966](#).
(image: NASA / Wikimedia Commons)

This is all getting a bit serious, so now is probably a good time to bring up a reality TV show. Recently, I was binge-watching *Bondi Rescue*, a series about the lifeguards who work at Bondi Beach in Sydney, Australia. In the midst of my TV marathon, I saw something that brought together the many strands of the "Look at me!" factor that have been floating around in my head.

I'm fascinated by things going wrong—I suppose that is why I've written so much about troubleshooting. Whenever an accident happens in my life, whether at home or work, I like to mull it over in my head. This is the most satisfying last step of any troubleshooting exercise, learning from the failure and trying to prevent it from happening again.

First, let me set the scene: *Bondi Rescue* has held my attention over many episodes because it's a perfect cocktail of dramatic elements. For starters, the location of Bondi Beach is very close to the sprawling metropolis of Sydney and its amazing beach weather. This proximity means a never-ending stream of unwitting tourists, many of whom seem to be getting acquainted with the ocean for the first time. Bondi's shoreline isn't a shallow kiddie pool: there are often big waves, dangerous currents which shift with the tides, and sharp rocks bookending the sides of this beautiful crescent-shaped beach. Add to the mix alcohol, massive crowds, and the group dynamics of protests, parties, classes, baptisms, etc. and it's a recipe for non-stop drama. Kudos to the creators of the show, who likely recognized the setting as the basis for a successful and predictable TV formula: the longevity of 13 seasons speaks for itself.

In Season 11, Episode 11 of *Bondi Rescue* there is an incident which I think perfectly encapsulates the dangers of the "Look at me!" instinct. The setup is simple: two students from the USA go swimming near Backpackers' Rip with a GoPro camera, mounted on a selfie stick. What could go wrong? If you're a fan of *Bondi Rescue*, what happens next is predictable: the tourists, unaware of the dangers present, get caught in a riptide and are carried away from the shore. Luckily, the Bondi lifeguards recognize the peril of the situation and intervene.

The essential difference between an accident and a near miss is often difficult to pinpoint. I've been in *seemingly* identical situations where others have gotten hurt and I have been spared, and I have gotten hurt in circumstances where others have walked away unscathed. We can speculate on the possible "outs" that weren't taken and the compounding circumstances present in this case. You could start with the failure to tap into the local knowledge about known hazards. For example, having a chat with the lifeguards about your intentions and getting their feedback. Or, simply noting the red and yellow flags that mark the safe swimming area on Bondi Beach. Unfortunately for those seeking fame, taking these preventative steps makes it unlikely that you'll be included in an episode (alas, staying alive does have its downsides). Maybe they could have been better prepared by being stronger swimmers: perhaps an olympic-class athlete could swim one-handed, with a camera, against a riptide, and have more than enough breath to give an Oscar-worthy monologue. Another factor might be the seemingly innocuous presence of a friend. People have been known to take more risks when in groups.

We could go much further, analyzing the factors present and chances for prevention in this *Bondi Rescue* incident. Instead, I want to connect it to the troubleshooting arts by focusing on one important factor: the presence of a camera. For better or worse, people act differently when a camera is around. After the tourists have been rescued, lifeguard Jesse Polock succinctly sums up what happened: "GoPros and rips don't mix… You can film yourself drowning, but that's about it."

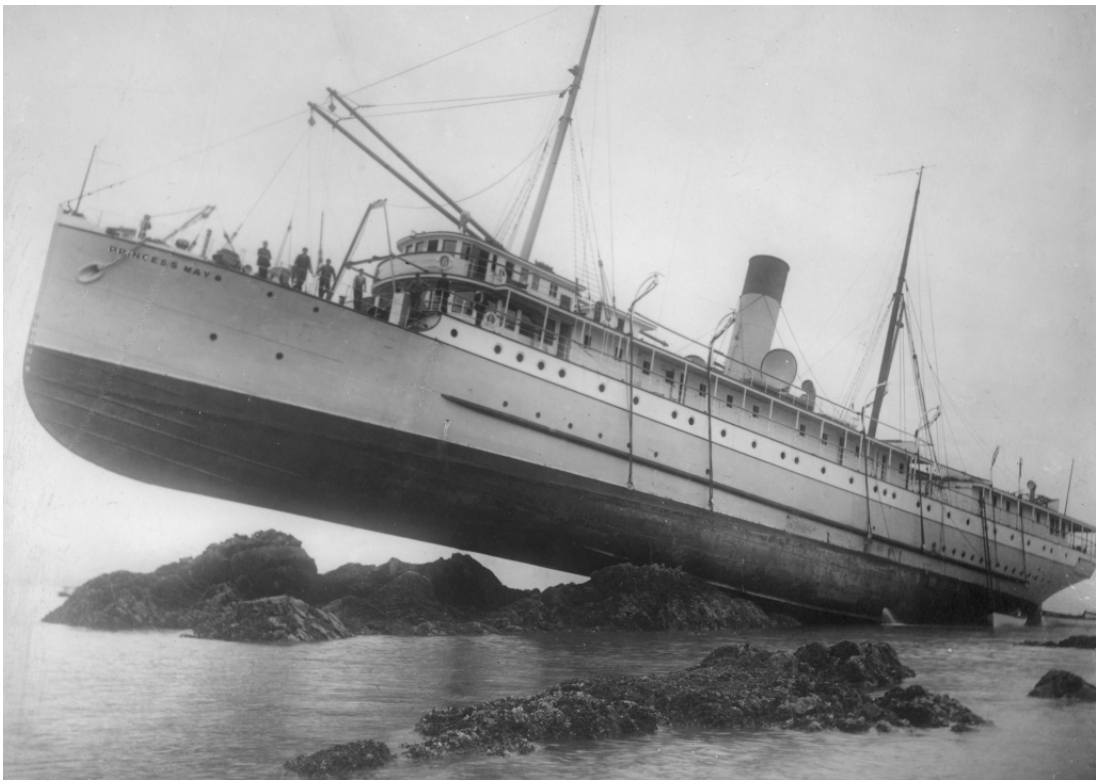
Check out the fine feathers. Attention-seeking isn't just for humans…
(image: Library of Congress)

"Say Cheese!"

In a dangerous situation, like swimming in a riptide, the insidious thing about a camera is that it competes for a precious resource: your awareness. Whether a menace is looming or fully actualized, awareness precedes corrective action. But, if your attention is focused on getting the best angle for your adoring followers, it can't also be used to see an imminent threat. Mugging for the camera is a distraction that will inevitably delay the recognition of critical facts, like a steady drift away from the shore.

That tradeoff might seem obvious, but there was another unexpected effect caused by a camera's presence on that episode of *Bondi Rescue*. Namely, the person was determined to hold on to the selfie stick to which it was attached, even though they must have sensed that they were in danger! In the segment, Bondi lifeguard Corey Oliver explicitly makes the connection between the close call and the presence of the camera: "I guess when you've got an expensive little toy in your hand that you don't want to lose, it's going to make it a lot harder to swim with one hand. So, we get a lot of problems with people trying to swim one-handed and not getting back to the beach."

This phenomenon appeared many times in the episodes of *Bondi Rescue* that I watched: people clinging to sunglasses, cameras, phones, toys, surf and boogie boards, etc., all while battling for their lives in the pounding surf. I guess they *really really* want their deposit back at the rental shop. The problem is that, when you're holding onto a precious item with one hand, you are severely handicapping your chances at a self-rescue. When the chances for survival narrow, desperately grasping a selfie stick isn't just a mental impediment, but a physical one too. It's *much* harder to save yourself when one hand is occupied and can't be used for swimming. If ever there was an example of narcissism literally killing us, this is it!



Salvage what you can, even if it's just the publicity value. If you ever get to a point like this, please take a photo or two. Posterity will thank you!
(image: William H. Case / Library of Congress)

Look at me—running into this reef

There's no shortage of accidents, famous and obscure, that have a "Look at me!" component. A recent and major one that instantly came to mind when I began writing this article was the *Costa Concordia* disaster. You probably saw this event heavily covered on the news in 2012. The gist is that a massive cruise ship, carrying over 4,000 people, ran into a reef in the Mediterranean Sea off the coast of Italy. The collision ripped a massive hole in the boat's hull and eventually sunk the *Costa Concordia*, resulting in 32 deaths.

As the details of the disaster were reported, perhaps the most puzzling detail (for me) was the ship's very close proximity to Giglio Island when it ran aground. Reading about Giglio Island, you find that it's a natural feature that formed millions of years ago, and has been occupied by humans since the Stone Age. Plenty of time for word of its existence to get around. Plenty of time for it to appear on nautical charts. Why was the *Concordia* cruising so close to a known hazard?

The answer is: getting close to Giglio Island was *intentional*. The *Concordia* was doing a sail-by salute for the residents of the island. This navigational diversion, executed exclusively for the visual and ceremonial pleasure of those on the ship and for those on land, is the very definition of a "Look at me!" stunt. Whether this particular salute was authorized seems to be a contested matter. However, it was easy to uncover the numerous forces pushing for the sail-by. For starters, consumers (i.e., the passengers) seem to expect jaunts like these:

> In the days after the disaster Costa Cruises chairman and CEO Pierluigi Foschi told an Italian parliamentary committee that sail by salutes do happen with the approval of cruise lines.

> He defended the practice of what he called "tourist navigations" and added: "It's something that enriches the cruise product. There are many components of the cruise product and we have to do them like everyone else because we are in a global competition."
>
> ***"Calls for cruise ship 'sail by salutes' to resume after Costa Concordia tragedy"*, *The Telegraph*, May 6, 2012**

A tour operator like Costa Cruises probably appreciates the free publicity that a (safe) sail-by offers: when a massive cruise ship passes within sight of a population center, it becomes a giant floating billboard. Unless they agree with David Foster Wallace's sardonic stance on cruising ("…any fool knows that Dr. Pepper is no substitute for Mr. Pibb, and it's an absolute goddamned travesty…"), they're more likely to pick up the phone and book a trip after such an encounter.

But the "Look at me!" factors of the *Costa Concordia* disaster go beyond the obvious commercial compulsions. They kept appearing as I read more about the circumstances of the tragedy. For instance, the island's residents and the mayor of Giglio really liked these sail-bys too. "Tourists and locals gather on the jetty to see the ships go by…it's a great sight," said the mayor of the island town of Giglio. Give those visitors a spectacle, maybe they'll talk about it on social media. "Look at me!" and "Give me something to look at!" are counterparts; exhibitionists and voyeurs need each other.

Further, I was also surprised to find in the news reports that there were a number of by-standers on the ship's bridge during the accident. One of these spectators was the head waiter (maître d'hôtel, if you're feeling Frenchy) Antonello Tievoli, who was invited up to the bridge for the sail-by. Tievoli was a Giglio native and his sister and parents, residents of the island, knew the *Concordia* would be passing near. Likewise, the Captain's lover was on the bridge during the crash. It's reasonable to wonder about the effects, subtle or overt, of these voyeurs on the Captain's attention and choices. For prudence's sake, the bridge on a cruise ship is typically off-limits to passengers, especially during complex maneuvering (like docking) when focus and intra-crew communication is critical. Did the presence of these spectators put pressure on the Captain to do something "impressive," "special," or "memorable" with the sail-by?

Building Buffers

Noting opportunities for prevention that *weren't taken* is equivalent to saying that an accident had a number of "causes." These are expressed in the negative, things that weren't done. This is different than the physical processes at work in a malfunction. Batteries dying or bolts breaking may be identified as a proximate cause, but these immutable facts of reality can't be used by themselves as a basis for future prevention. That's because the laws of physics are a given: we can't fix things by supernaturally declaring steel to have a different strength, or by having a particular chemical compound magically hold more electrons. Prevention always considers alternative courses of action: things that weren't done in the past, but you hope to do in the future. Next time, we will use a stronger bolt, employ a battery that stores a bigger charge, use the machine differently, make more frequent inspections, etc.

Along these lines, poring over a really thorough accident report can feel unsatisfying when it reads something like: "The pilot was fatigued AND the landing was hard AND the crew's training outdated AND critical routine maintenance was neglected AND the weather was bad AND it was dark AND…" I think our instincts would prefer a single cause, so we can feel better by thinking, "If I just do this *one thing*, then I'll be safe!"

I like getting deep into the finer details of the circumstances surrounding a particular incident because it's here you'll often discover *multiple* paths for prevention. This complexity is actually an opportunity because methods for prevention will vary in cost. Since our prevention resources are scarce (just like our fix-it resources), identifying and choosing the ones that can be realistically implemented is key.

With this broader mindset, you can begin to think about failure prevention in terms of buffers. These are all the things that stand between you and a fiery end. And the more of them the better! Your training, processes, procedures, equipment, the prevailing conditions, random chance, etc.: these can all work for or against you. This gets back to what I said before about similar situations ending with different outcomes. Not every tourist that goes swimming in the ocean with a camera ends up drowning. Some do, most don't. Sail-by salutes don't usually result in a multi-billion dollar loss (in fact, the *Costa Concordia* had previously saluted Giglio Island).

Troubleshooting Lessons

You may be thinking that these human factors are outside the realm of troubleshooting. If "Look at me!" leads to harm, it's hardly the fault of the machine, right? Was that selfie stick and camera operating properly when those two students on *Bondi Rescue* got caught in a riptide? Yes. We know this for certain because—the footage is included in the episode! Likewise, a ship can be steered in any direction of the compass. Whether towards a rocky reef or to the safety of open water, the rudder and engines will happily oblige. In either case, you can't say there has been a mechanical malfunction.

But we can't stop there, simply concluding that "the machine was operating properly," and wash our hands of the incident. Nothing to see here, folks! Move along. No, that goes against the ethos of what I try to teach here. Humans and machines, when combined together, form a super-system. The interaction between man and machine should be designed to gracefully accommodate our human tendencies, good or bad. To that end, I have a few suggestions for ways to troubleshoot "Look at me!" problems:

- **Transparency:** we've looked at many ways that social pressures can lead to accidents. It may surprise you that the same forces can also be harnessed in their prevention. It's not pleasant to always be monitored while at work, but if you're entrusted with lives and expensive property…sorry. Prior to the accident, the captain of the *Costa Concordia* turned off the alarm on the ship's navigation system. If I was the administrator of such a system, such a deactivation would instantly light up the phones of senior management. Everyone, including the captain, would know that such a move would be widely reported and scrutinized. It would be a BIG DEAL. There would be paperwork. There would be conference calls. Oh yes, there would be many conference calls.
- **The Sterile Cockpit:** machines often require the dedicated focus of their operators for tricky maneuvers. Think of: landing an airplane, docking a supertanker, or backing up a fuel truck. During these critical moments, a crew's attention needs to be vigorously defended from unnecessary distractions.
- **Gear up:** if there's no way to stop the showboaters, it's time to deploy bubble-wrap, bumpers, life jackets, helmets, harnesses, shin guards, and belay devices. If you're ultimately responsible for accidents, the use of safety gear may need to be official policy. Businesses that have tourists doing semi-dangerous things (e.g., kayaking, zip-lining, snorkeling, rock climbing, etc.) understand this: they know that they're dealing with inexperienced people who lack the ability to accurately assess their capabilities with respect to the activity. When that's the case, get out the pads!
- **Create safe spaces for distracted self-promotion:** if you know that people like to take selfies next to a cliff, then it might be time to install railings and warning signs. Interestingly, some cruise lines implement a variation on this theme, whereby they create a designated time for passengers and the crew to interact. Whether it's called a "Captain's Dinner," "Captain's Gala," or "Captain's Ball," the idea is that people are going to want to talk to the officers and take pictures with them. So, you create a designated time for this attention exchange that doesn't interfere with the crew's official duties. If it simultaneously allows the crew to feel honored and quenches the public's desire for voyeurism that might otherwise be channeled in distracting ways, these events can even be considered a roundabout safety measure.
- **Prompted (or forced) to do the right thing:** the latest version of Apple's iOS includes a standard feature to prevent access to your smartphone while driving (good idea, but I suspect that most people turn it off). Make a distraction-free environment the default, or *enforce* preferred usage with something like a speed governor. Procedures and the culture of a workplace can also mitigate showboating: your training, professional expectations, organizational policies, and colleagues may provide a much-needed reminder that a selfie can wait.
- **Raise awareness:** help people understand the consequences of showboating. On that note, I leave you with this eery video of the *Costa Concordia* wreck.

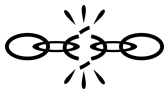
References:

- Header image: Rothstein, A., photographer. *Showboat anchored at levee, Saint Louis, Missouri*. Missouri Saint Louis Saint Louis. United States, 1939. Jan. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2017724674/.
- [1] Lao Tzu and Stephen Mitchell, *Tao Te Ching: An Illustrated Journey* (New York: HarperCollins, 1999), verse 24.

On Selfies And Showboating: Troubleshooting The Imminent Dangers Of "Look At Me!" was originally published June 13, 2018.

Notes:

Accident Causes ≠ Preventative Measures



> Action should be taken before a thing has made its appearance; order should be secured before disorder has begun.
>
> **Tao Te Ching (verse 64)** [1]

Troubleshooters are naturally curious about prevention. With sweat on your brow and grease under your fingernails (or the digital equivalent thereof), your mind instinctively wonders: **"Does this have to happen again?"** Clients may like a quick fix and a return to business as usual, but they will appreciate you on a deeper level if you can show them how to stop a problem from recurring in the future. This is the realm of awkward hugs (in exchange for problems bested, I've gotten a few of these).

The script for a preventative remedy seems straightforward. First, identify the cause of a malfunction. But hold on, what does it really mean for something to be the *cause* of an incident? That might seem like a silly question, but I had an important realization about causes while writing my last article about selfies and showboating: both the physical processes that lead to an accident and the measures to prevent them are often, and confusingly, referred to as "causes." As you'll see, they are different concepts, and understanding the distinction is critical for both your repair and prevention efforts.

Ripped From the Headlines

To show you how the two meanings of the word "cause" are conflated in popular usage, I did a quick scan of the

news. It didn't take long for me to find examples of the confusion between accident processes and prevention methods. Here are a few quotes that I pulled from recent news stories. While reading, pay close attention to the word "caused" (and the synonymous phrase "resulted in"):

- "The Ionia County Sheriff's office responded to an accident on Wednesday morning that was most likely caused by fog." ***WILX News 10, July 25, 2018***
- "Unenforced policies, lack of communication and ineffective traffic rules resulted in last year's double-fatality accident at SSR Mining Inc.'s Marigold Mine, according to an investigations report from the Mine Safety and Health Administration." ***Elko Daily Free Press, August 6, 2018***
- "A tug operator eventually pleaded guilty to involuntary manslaughter, acknowledging that the accident was caused largely due to his use of a cellphone and laptop while steering the barge." ***USA Today, July 20, 2018***

Visibility, communication, and the presence of distractions were obviously relevant *circumstantial factors* in these incidents. Perhaps improvements in these areas would have resulted in these accidents never happening. However, advanced students are probably wondering: has there ever been a catastrophe which involved instances of good visibility, clear communication, or focused operators? (Answer: yes.) And further, if their presence is optional, how can we point at something and say confidently, "We've found it—forever and always, this is the cause!"?

In the headlines above, many of the causes listed are statements of things *missing* or *not* done within the accident's context: there was a *lack* of visibility, a *failure* to communicate, traffic rules *weren't* followed, etc. Fog is simply water droplets suspended in the air. Even when I tried the foul-tasting fernet once, I never saw a weather phenomenon grab a steering wheel and lead a car into a ditch. Water vapor is not a supernatural force, so of course we know what is meant when someone says that "fog *caused* an accident": namely, that it's difficult to safely steer an automobile when you can't see. This implies that something was missing that might have stopped the accident. What was it? Adequate visibility.

Here's the problem: thinking of *causes* in this way involves a semantic sleight of hand, because these are actually pitches for prevention. When strongly worded like this, a "cause" appears to the reader as authoritative and definitive, perhaps even singular. The underlying message is: "Sleep well tonight reader, we've figured this one out." Doing so, we've skipped over the accident processes and are boldly projecting out into the future, using subjective and value-laden opinions about the best way to prevent a certain type of accident. That's because avoidance involves an *economic* calculation, the result of choosing among alternatives in the face of scarcity.

Causes: Inevitable and Finite

> **cause,** noun
> 1 b: something that brings about an effect or a result
>
> **Merriam-Webster**

Let's deal first with the sense that a cause is that "something," referenced in the dictionary definition above, which results in a given effect. When investigating an accident or malfunction, we're talking about that particular set of circumstances and actions under which an incident would be certain.

There's a relentless march of logic at work here: a ship ramming into a reef at full speed will result in a dent (or worse). Contacting the rocks with sufficient force will *cause* the hull to tear. On a car, applying the brakes with completely worn pads will score the rotors. Metal touching metal *causes* telltale lines on the surface of the rotors. Computer code that attempts to access restricted memory results in a segfault; a program that tries to interact with RAM that is off-limits *causes* the operating system to shut it down. Just like the sun rising in the east and setting in the west, these specific pre-conditions and actions produce inevitable results.

However, you can probably spot the troubleshooting dilemma that this type of surety presents: if something is inevitable, **then nothing can stop it!** Preventative measures can't be marshaled at this stage of the analysis because we're dealing with immutable truths of reality. You can't change the physics of steel contacting rock, nor change the logic of adding 2 and 2 to get 4.

Even if you did have the ability to change the laws of physics, you wouldn't necessarily want to be able to

"troubleshoot" at this fundamental level of nature. Imagine being able to "fix" shipwrecks by being able to declare that steel can't be damaged by rock. First, it would take magical powers and second, there would be unintended consequences: presumably, it would be impossible to stop any ship!

The ways that steel and rock interact are dependent upon physical laws, and their predictability is something you rely upon. That reality might not be favorable when the context is a ship's hull and a reef. But, if you're quarrying stone, you'd like your steel drill to actually have an effect on the rock. After all, if you're going to sweat all day in the baking sun while swinging a hammer, it would be nice for it to matter. A solid object exerting a force upon another solid object can be a benefit or a hazard, depending on the context.

Finally, the causes which we are called upon to discover don't necessarily have to originate in the fixed laws of the universe; they could simply be the rules under which you choose to work. For example, an operating system isn't required to jealously guard protected memory. Some OS's may, some may not. A protected memory scheme is a choice made by software architects and adopted voluntarily; it's not a Law of the Universe you'll find described in a book by Stephen Hawking.

If you're a coder developing a product designed to work on a particular platform, the rules of the operating system are equivalent to a physical law like gravity. In the case of a software project, creating a new operating system just to run your code would probably be too costly and limit the appeal of your product. Therefore, the rules of the OS are something you'll just have to accept.

One way, among many, to intervene…
(image: Library of Congress)

The Infinite Varieties of Preventative Measures

> The archetype of causality research was: where and how must I interfere in order to divert the course of events from the way it would go in the absence of my interference in a direction which better suits my wishes? In this sense man raises the question: who or what is at the bottom of things?
>
> **Human Action, Ludwig von Mises**

Preventative measures can't alter the laws of physics or logic; change can't be effected at this level of reality. Therefore, the aim of prevention is to intervene *before* an inevitable process can get rolling. That is, we want to keep our ship's

hull from ever making contact with that rocky reef. We desire to have the brake pads be the only surface that touches the rotor on a set of brakes. We want our computer program to stay within the bounds of the memory allocated to it, thus ensuring that the operating system will let it run in peace.

Once we've identified the thing we want to avoid, prevention enters the realm of infinite possibilities. Take our ship and rocky reef, for example. There are countless measures we could take to prevent a shipwreck: always maintain a certain distance from the shore, install a sonar warning system that detects obstacles, vow to navigate only in good weather or during daylight, consistently maintain the steering and propulsion systems so they are always ready to make course corrections, create a detailed map of hazards, etc. We could even keep the ship safely anchored in the harbor at all times!

That's a long list of options for this particular problem, and you can probably think of many more. But, you're probably wondering if preventative measures are really spread over an infinite realm of possibilities. You might think that I'm exaggerating when I say that preventative measures are endless. However, merely by adding time as a variable to the mix, you can easily see that the opportunities to intervene are boundless.

Imagine a timeline, ending at the point where an accident was inevitable ($\mathbf{t_0}$), stretching back forever in time:

$t_{-\infty} \ldots t_{-5} \rightarrow t_{-4} \rightarrow t_{-3} \rightarrow t_{-2} \rightarrow t_{-1} \rightarrow \mathbf{t_0}$

Each mark on our timeline represents the passage of one unit of time. I've used the NASA T-minus countdown style, which shows the amount of time remaining before an event happens. The units on your incident timeline could be seconds, hours, days, weeks, or even fortnights! The scale doesn't matter, because any division of time we can pick is infinitely divisible. To prevent an accident, all you need to do is **change the course of events before they become a certainty.** By definition, that means you could have intervened *anywhere* on the timeline before the end. You could have chosen to interfere at t_{-1}, t_{-5}, t_{-20}, t_{-35}, $t_{-35.1}$, $t_{-35.2}$, $t_{-35.3}$, etc. to infinity.

When you combine the *when* (the timing of an intervention) with the power of our imaginations to choose the *how*, it's easy to see that the raw number of possibilities for preventing any accident are without end. That might make you feel better, knowing that you've got a lot of choice for prevention. It might also make you feel overwhelmed! Take comfort in the fact that, while preventative measures are indeed infinite, there will only be a small subset of options that simultaneously meet your particular situation's constraints (practical, legal, economic, etc.).

Absent Space

> A ship in harbor is safe, but that is not what ships are built for.
>
> **John A. Shedd**

In the news snippets above, the reporters suggested that not driving in fog, better communication, and paying attention while towing a barge would have prevented those accidents. However, because options for prevention are without bound, you could easily have suggested alternative "causes" with headlines like:

- "Operating a car causes car accident."
- "Humans present at a mine result in deaths."
- "Two boats using the same river, at the same time, causes a collision."

In other words, you could solve the problem of mine accidents killing people by—not having people near the mine at all (maybe one day robots will do this dangerous work…). Driving accidents could be prevented by—wait for it—not driving! Boats running into each other on a river could be solved by only allowing one boat at a time to use the waterway (that is, have other boats be *absent* as a matter of policy).

Within the range of alternative prevention methods, many will be impractical, unpopular, unenforceable, costly, unacceptable to the prevailing culture, etc. However, it doesn't make them any less effective. As a troubleshooter, it's important to recognize when value judgements have been injected into an analysis. If you understand that prevention methods are infinite, you can ask "Out of the many, why are we focusing on *these* few?"

When considering how to stop the next accident from occurring, what is promoted as acceptable will be intertwined with the current social, cultural, and political context. Given that setting, there will often be a debate among competing interests over *whose* vision for prevention will prevail. Nuclear power is a good example: if you believe that electricity from this source cannot be generated safely, then preventative methods will focus on prohibition (i.e., the *absence* of a nuclear power plant is the best way to ensure a meltdown never happens). Likewise, the owners, operators, employees, and consumers of nuclear power will each have their own interests and stances on prevention.

I re-watched *Jaws* recently, that venerable first blockbuster, and was reminded how it dramatizes the competition of interests with differing visions of prevention. Of course, I remembered there was a big shark involved somehow and the famous ad-lib "You're gonna need a bigger boat." What I had forgotten about was the tension between the stewards of the beach town over the correct means to stop the next shark attack. At first, Amity Island's police chief Martin Brody and mayor Larry Vaughn have very different ideas, both about the nature of the threat and the remedy. In one heated argument, Mayor Vaughn defends the economic interests of the town, saying "Look, we depend on the summer people for our very lives…"; Brody counters with "Larry, we're going to have to close the beaches!" You don't have to go full-on X-Files or Alex Jones, but when a particular line of prevention is being promoted or closed off (particularly within a political context), it's always a good idea to ask "Cui bono?"

When it comes to conceptualizing prevention, an easy place to start is with absence. A real world example of this principle in action are traffic patterns: whether it's a road, airport, or harbor, it's desirable to have vehicles travel in the same direction when in close proximity. After enough head-on collisions and rush hour jams, our distant ancestors figured out that a simple social convention could create an absence of vehicle-on-oncoming-vehicle hazards. The origin of one-way traffic patterns goes back to at least ancient Roman civilization ("…a key part of the traffic system at Pompeii is the use of one-way streets…"). I visited the Panama Canal, and it was even in use at this feat of "modern" engineering. Traffic along the waterway flows one-way at a time, because the canal has some narrow parts that make bi-directional movement dangerous ("ships move in one direction at a time due to safety constraints to cross the Culebra Cut.").

My guess is that a uni-directional flow for movement is likely as old as humanity itself, arising organically from the problems of sharing a common route of travel. Even at Thanksgiving, we pass the food around the table in only one direction. (Oh, how it's torture when the mashed potatoes start with the person next to you—in the opposite direction!) These schemes are so wide-spread because the principle is easy to understand: let protocol lead to the absence of hazards.

Positives and Negatives

When it comes to troubleshooting, a "cause" can have a positive or negative meaning. I'm not going to fight the common usage, but merely want you to think about both sides. The first sense features the word as a positive concept, that "something" that is *present* and directly brings about an effect. The other meaning is about what was *missing*: the policies or procedures that weren't followed, the lack of situational awareness by an operator, the ignorance of a better way, etc.

Understanding the physical and logical processes that underlie a bad event is the key to stop it from recurring in the future. Once the precise mechanisms become clear (hull hitting rock, restricted memory being accessed, etc.), they become the focal point of your prevention efforts. You can then brainstorm the myriad ways to tweak reality *before* the inevitable happens, choosing both the means and the timing.

Another conclusion we can draw from a low-level analysis of how failures occur is that accidents are never accidental. Mishaps, large or small, emanate from predictable and ultimately knowable facts of reality; the only unexpected thing about them is the feeling of being caught unawares (I touched on this in Failure Most Foul: "Isn't it interesting that we live in a world where it's certain that every machine will eventually break down, and yet our *experience* of those failures is one of surprise?").

Accidents involve long chains of causation, going as far back as you care to look. Understanding that logical progression and then deciding how to interfere are two related, but ultimately different, things. As your awareness of the individual links grows, so too will your ability to choose prevention methods that merit your precious time and scarce intervention resources.

References:

- Header image: *Railroad Wreck*. 1922. [Photograph] Retrieved from the Library of Congress, https://www.loc.gov/item/2016833122/.
- [1] Lao Tze, translated by James Legge. *The Sacred Books of China, The Texts of Taoism* (Oxford: The Clarendon Press, 1891). Retrieved from the Internet Archive, https://archive.org/details/wg939/page/106/mode/2up.

Accident Causes ≠ Preventative Measures was originally published February 27, 2019.

Notes:

Acknowledgements

From the First Edition (May 8, 2014).

I am grateful for:

- The love and support of my parents. The stable foundation they have provided since day one has made everything I do possible.
- My aunt, Joan Anne Maxham, who faithfully edited every word of this text as I serially published *The Art Of Troubleshooting* to my blog over the course of two years. "Aunt Persnickety" gave great feedback on both the small details and the big picture. Considering that she loves to find typos in the *New York Times*, I can only assume that fixing my prose was even more satisfying.
- The Troubleshooters I interviewed, for their deep insights and help polishing my ideas: Alex Chaffee, Ken Fechner, Jamie Karrick, Rich Kral, Karl Kuehn, Dan McCormick, Mike McCormick, Austin Quade, Gerald Quade, and Jeremy Sheetz.
- The following people, who gave generously of their time to provide feedback or aid in numerous other ways: Ivan Batanov, Ben Bayer, John Brogan, Sam Carter, Alex Chaffee, Brandon Clow, Darrell Clow, Elana Connor, Dennis Crall, Lance Fuller, Jason Gollan, Otto Grajeda, Dave Hoffer, Micah Joel, Jamie Karrick, Eric Lujan, Phil Lukish, Stan Mars, Amanda Maxham, Cynthia Maxham, John Maxham, Mike McCormick, Charlie Miller, Antonis Papatsaras, Austin Quade, Steve Renaker, Lou Rosenfeld, Brad Ross, George Rothdrake, Adam Rutland, Greg Schwendinger, Damian Spain, Sean Tario, James Thomassen, Jimmy Tobias, Matt West, Harold Woo, Jacob Woolcutt, and Matt Work.
- My fellow co-founders of Discovery Mining: Matt Work, Leslie Brennan, and Andy Jenks. What a blessing it was to fall in with these guys! Later on, I would hear horror stories of bad matches among business partners, so this realization was even stronger after-the-fact. One of the rare teams I have worked on that truly meets the definition of "synergistic," with a result much greater than the already amazing component parts. We struggled and overcame, while maintaining our civility and integrity.
- The entire Discovery Mining engineering team, especially Antonis Papatsaras, Steve Renaker, and Damian Spain. Every day, it was a pleasure to work with you and the amazing talent we had the fortune to assemble. The mind-bending challenges we encountered (and mostly bested) were the genesis of the material in this book.
- Brewster Kahle, who offered me an internship at the Internet Archive while in college. That formative experience led to my first "real" job at his company Alexa, and a move to San Francisco. Alexa is where I would meet the people with whom I would eventually start Discovery Mining.
- The Wednesday Night Wrenchers, which first met at Matt Work's garage and now

at the Piston & Chain motorcycle club in San Francisco. The principles described in this book are seen in action every Wednesday night!
- Inspiring teachers, who kindled and nurtured a passion for learning and discovery: Ms. Stern, Mr. Bremer, Mr. Heiks, and Dr. Hester.

About The Author

The author at the top of Mt. Fuji. Getting here required troubleshooting—his body's desire to turn back!

Jason Maxham's eclectic background eventually led him to think and write about troubleshooting in a general way. He's an aviator, audiophile, choral singer, foodie, motorcyclist, programmer, world traveler, and entrepreneur who's been involved in multiple startups—most successfully as co-founder and CTO of Discovery Mining.

He's very interested in your thoughts about this book, so feel free to drop him a line at: bookfeedback@artoftroubleshooting.com