AN ENGINEER'S GUIDE TO SOLVING PROBLEMS

"This conversational style book is something like I would write. You are definitely going to enjoy giving this one a read"
— David L. Jones, EEVBlog.com

BOB SCHMIDT

An Engineer's Guide to Solving Problems

Bob Schmidt

Second Edition; Creative Commons Licensed Release

Publisher's Cataloging-in-Publication data

Schmidt, Robert Warren.
An Engineer's guide to solving problems / Bob Schmidt.
p. cm.
ISBN 978-0-9887476-2-3
Includes bibliographical references and index.
Second edition.

1. Engineering. 2. Problem solving. 3. Engineering systems. 4. Mechanical engineering. 5. Electrical engineering. 6. Computer engineering. I. Title.

TA147 .S2 2014
620 --dc23
2013921921

Cover Design by Luke Fletcher
www.fletcherdesigns.com

The First Edition was previously titled: *The Dog Barks When the Phone Rings: An Engineer's Guide to Solving Problems.*

Table of Contents

Introduction

"Listen," he said. It was an invitation, not a command.

The storyteller dropped his voice a little below its usual quiet calm. Without even realizing it, we all leaned in a bit more to make sure we could hear.

This was going to be another classic. It would be one of those tales of puzzle and personality; a story that would have a solid moral if we could dig deep enough. The tale would save us from some weakness or mistake of our own if we could just listen closely enough, and learn from the follies and misadventures of others.

I have come to believe that all professions share these nuggets of wisdom, passed from generation to generation. They are part myth, part fact, and a bit of fancy, all woven together in sturdy tapestry by the wisdom and experience of the teller.

We laugh at the humanity, recognize our own traits in the stories, and carry them with us to each new job. Sharing the stories and jokes, we spread them like a virus, mutating and molding them to the new audience.

Always there is at the heart of each story some inner meaning, some lesson or moral that we might not even recognize in the laughter and delight of the story itself. But eventually (hopefully) the wisdom of the tale leaks into our own efforts. If experience is what we get just after we needed it, then these stories "pay it forward," giving us the benefit of somebody else's experience, without all of the pain.

Someone once told me that a parable is an earthly story with a heavenly meaning. It is from that idea that I have come to refer to these tales as *parables in problem solving*.

This kind of spirit and passion (or is it an affliction?) infuses certain practitioners of any profession or hobby.

Because I am condemned to a lifelong love of things electrical and electronic, that is where many of my stories live. But if you are quiet and contemplate the stories for a while, you may find echoes of your own particular passion in these little mysteries, comedies, and tragedies. There may be a moral in here that resonates with your spirit. I hope that you will be as fortunate as I have been to have heard and learned from some of these storytellers.

"Listen..."

Executive Summary

I once had a boss who found my emails to be long and detailed. Perhaps they were even painful. He requested that I put an executive summary at the beginning of any long emails.

He did not want to wade through all the details just to know whether he needed to take an immediate action.

This was a very good idea. It forced me to think hard about the shortest possible message I needed to convey and to quickly discuss the main point I needed to communicate.

So here is the short story: I want you to master a basic problem-solving method, which could also be called "the five questions":

1. What do you know? (Describe the problem.)
2. What are the rules? (Know the basic science behind the system.)
3. What don't you know? (Outline the missing information.)
4. How can you find out the stuff you don't know? (Do research and experiments.)
5. How do you know when you are ready to solve; or have already solved the problem? (Evaluate and verify your solution.)

Wow. That looks pretty basic, eh? If you are an engineer, you have already gone through at least four years of education doing these same steps over and over again. If you are an engineering student, you are in the middle of a lot of classes covering these same techniques. The names of the variables and constants change, but the methods are pretty consistent. For lots of engineering and science problems, you are really just writing down equations (rules) that you have learned, filling in the constants and variables (stuff you know) then doing the math to solve for an unknown quantity (stuff you don't know).

Design is very much a human activity. Like all human activities the process is subject to error. When we design systems, we are solving problems and also creating new problems. Some problems will be the result of old mistakes that have been made before, while other problems will be new.

Even very good designs might have components that eventually fail. This creates new problems that we should solve.

The key idea you need to take from here is that debugging (or troubleshooting, or "fixing stuff," or whatever you want to call problem-solving) is an intrinsic part of the design process.

When I was managing an electronic hardware design team, I told them I wanted the team to make lots of mistakes, but I wanted them to make those mistakes quickly. Then I added that I wanted those engineers to find the mistakes and fix them—faster.

Asking the five questions is great for solving technical problems. It works extremely well on complex systems of machinery and electronics. This method can produce spectacular success in diagnosing the most subtle bugs in computerized (virtual plus physical) systems that confound many projects. Occasionally, these methods can be used to diagnose very difficult health care issues.

(Disclaimer: Unfortunately, these methods become unreliable in dealing with human type "people" problems. Sorry about that boss. Nothing in this world is "one-size-fits-all.")

If you are a non-technical manager, you might not have enough time (or enough interest) in all the details to read this entire book. Let me save you some time and effort.

You could possibly get maximum entertainment and minimum learning by reading this executive summary, then skimming through the remainder of the book.

If nothing else, please read chapters 1, 12, and 18-24 to get the central story. Also, as you flip through, look for the universal Fast Forward symbol: ⏩ . This symbol is there to help you know to stop scanning.

Most of the stories that follow this symbol start with the words “once upon a time" (although this first joke does not.)

⏩ There is an old military joke: what is the difference between a fairy tale and a war story?

A fairy tale starts with “*Once upon a time…*”

A war story starts with “Now this ain’t no shit son, I was there, and I saw it myself…”

What is this book *really* about?

This book is about solving problems in complex technical systems. This is not a book about “issues.” It seems that somewhere in the past few decades, it became popular management-speak to talk about “issues” instead of “problems.” Apparently “issues” are not something negative, just something we have to resolve, whereas “problems” imply that we made some mistakes and might get fired. So somebody toned down the language to keep upper layers of management calm.

That seems crazy to me. Products and new designs have problems. Engineers fix problems. So for the purposes of this book, I will talk about problems and problem solving. I will not talk about “issues.”

One of the ways you can learn to solve problems is by listening to other people share stories of how a problem was solved in the past. These stories are best when they are short, include some important details, and end up teaching you something general about problem solving that you can use in the future. These stories are even better if they have some drama or they make you laugh. Sometimes such stories make you laugh nervously because you can recognize yourself in them.

Another way we learn to solve problems is by applying a process or method such as our five questions. This book has four basic sections.

In the first section, I will confront the question of how to describe and document a problem. For some problems, you only need a clear description to understand what needs to be fixed. In short, you try to answer the three questions “What do you know?”, “What are the Rules?” and “What Don’t You Know?”

In the second section, you will think about how to find out the stuff you don’t know—also known as *debugging, experimenting, or researching*. This can take a lot of time and demands great patience. You probably are already sure you know how to do this part. You are wrong. Like everybody, you take shortcuts, and the more complex the problem, the worse the results when you cheat.

Even if you already have some good thoughts about how to go about doing debug or experiments or research, you probably still cheat a little by not doing them in an organized and clearly documented way.

Debugging is like playing golf or having sex. Everybody does it wrong sometimes; some folks lie and some folks cheat. Few are as good as they think they are, and we all look ridiculous doing it.

There are some differences between debugging new systems and troubleshooting existing (previously working) systems. There is a lot to talk about here and we will return to this topic later.

In the third section, you get to the good stuff and maybe even solve some problems. There is a villain lurking in the shadows here, and sometimes he looks just like the hero of your story. In fact, the villain and the hero look just like you and me. The villain constantly leaps to cause in a single bound; the hero gets there only when the time is right. The villain runs away based on flimsy evidence; the hero goes back to the beginning and proves his conclusion both backward and forward.

There are giant craters in your Yellow Brick Road. You are going to need help from lots of other people to get to good solutions. You will need some inspiration to get to brilliant solutions.

In the fourth section, you will reflect on what you have learned. You get to look back and add some perspective to the business of problem solving. Maybe you can even learn something about the business of business.

As I’ve said, a parable can be seen as an earthly story with a heavenly meaning. I have always liked that description, but I will warn you that the parables you find here are more like just a hell of a story—and some of them might even be true.

No, wait. Only *pieces* of some stories might be true. All of the names have been changed to protect both the innocent and the guilty. Facts might be swapped between stories to make them more readable or understandable. Maybe I made stuff up in some places. The idea is for these parables to serve as conduits to your understanding of problem solving.

One more thing: I will apologize to all of you in advance.

Sometimes in my enthusiasm to make a point, I fall into rather crude or exaggerated words. My editors might suggest gentler words or tactics to obscure profanity. But sometimes, one just needs to use the full range of a rich and emotional vocabulary to get the appropriate emphasis.

Sorry about that.

Why should you learn these methods? Why is this important?

Let's do a simple exercise.

Stand up, and raise your hand if you think there is a risk that we will ever run out of problems to solve.

Hands up! Anybody? Anybody?

But why don't we run out of problems? Shouldn't this stuff get easier and easier until we can do it all with unskilled labor or machines?

No, I don't think so. As long as there are end-users, customers, salesmen, bosses, or humans, we will always have somebody who wants a product or service to be one or all of the following:

- Easier to use
- Cheaper
- Smarter
- Faster
- Smaller
- Lighter
- Less expensive
- Cheaper
- Cheaper
- Cheaper … and better!

This means that you have two kinds of problems to solve for any project. The first are the problems posed by the project specification: you need to make the system/product cheaper or better in some way. The second kind of problem in every project are the mistakes or limitations that you introduce when solving the first kind of problem. Think about it this way: If debugging is the process of removing errors from a design, then designing must be the process of putting errors into a design!

One thing this means is you need better testing to uncover problems. Becoming a better tester helps make you a better debugger, and understanding what makes you a better debugger helps make you a better designer. You will always make mistakes, but hopefully you will fix them before your customer can see them.

Why Should I Listen To You Bob? You ask, "What Do You know about My Special Skills?"

That is a good question. I very well might be the dumbest guy in the room. I assuredly don't know much about whatever specialty skills you have. Yes, I have done a lot of electronic system designs over that past 40 years, but there is no assurance that any specific debug experience would match some problem you have today.

I started doing electronics design in high school. Nearly everything I built for several years did not work very well, if at all. But I learned to study my mistakes and failures. Once I got the hang of it, I found that the differences between the successes and failures usually involved some really small details.

I spent a lot of time designing industrial single-board-computers and way too much time writing code in assembly language. At one time, I was in a group of engineers designing DVD players at a consumer electronics company. I then moved to a group designing satellite set-top-box receivers. There were two of us named "Bob" in that particular group and we sat next to each other. Having two Bob's led to occasional confusion over who was being referenced in a conversation or meeting. Soon I became known as "Audio Bob" and the other fellow was dubbed "Video

Bob" due to the skill areas we were covering at that time.

I have also been "HDD-Bob" (hard-disk-drive), and "Manager-Bob" and many other "Bobs" in my career. I have owned and run several businesses. It is likely that none of these things qualifies me to work on your projects.

But I have spent a lot of years listening to really smart people tell their stories and explain how they solved problems. I have also solved some difficult problems by myself and in large global teams. Over time, I have become convinced that there are some really simple, basic ideas that make all the difference between rapid, successful debugging and disastrously slow or failed debugging. I want to share these simple concepts with you and I really hope they help you.

This book talks about solving problems as an intrinsic part of the design process. Occasionally, I use the word "fix" instead of "solve." The word "fix" has an implied meaning of repair or service. Some folks might see this as somehow less noble than problem solving, debugging, or troubleshooting.

For these people, the word "fix" pulls up an image of a plumber instead of, say, a fluid dynamics expert. Nothing could be further from the truth. In this book, *fixing* a problem is assumed to be the same as solving, diagnosing, implementing, or any other high-brow words you might choose.

If you have perused the table of contents of this book, you have probably noticed that there are several chapters about communicating. No doubt, you are probably wondering why.

Let me answer that with an appropriate joke: How can you identify an extroverted engineer?

He is the one who stares at *your* shoes while talking to you.

Engineers and scientists are distributed all along the autism to normal spectrum, but there is some kernel of truth in the stereotype of the nerdy, introverted, highly focused and slightly anti-social geek. Truly extroverted engineers might end up in management or sales, simply because they can overcome the communication barriers that would otherwise keep them confined to purely technical pursuits.

For that reason, much of this book talks about communication. Just as I have listed five Questions as summarizing the method in this book, we could talk about the five C-words of a successful debug:

1. Communication
2. Contemplation
3. Concentration
4. Confirmation
5. Communication

Hey wait a minute, isn't "communication" in there twice? Yes it is—and for good reason. A good solution to a problem always begins and ends with clear communication.

Let's dive in.

A Really Simple Problem Solving Method

1. **What do you know? (Describe the problem.)**
2. **What are the rules? (Know the basic science behind the system.)**
3. **What don't you know? (Outline the missing information.)**
4. **How can you find out the stuff you don't know? (Do research and experiments.)**
5. **How do you know when you are ready to solve; or have already solved the problem? (Evaluate and verify your solution.)**

Chapter 1: The Dog Barks When the Phone Rings

The phone jangled in the sleepy Kansas garage. For some reason, telephone-company repair shops have always been called "garages." Yes, the well-stocked trucks came and went from this place, but in truth it looked more like any small office, perhaps just a little more organized. Joe liked to keep the desks clear and the files in order.

It just made things a bit smoother when an all-out battle ensued. Like last month when they had called in all hands to recover from the twister damage near Lawrence, restringing and tracing circuits around the clock. Joe felt good that his crews had been so cheerful and efficient. People depend on their phone service and Joe's guys were the best.

The phone jangled a second time as Joe lifted it from the cradle.

"Repair service! How may I help you?" Joe nearly sang his standard greeting.

"Yes sir. I need you to come fix my phone." The customer's voice was clear but somewhat thin. Joe concluded she wasn't a very young lady, but not elderly to the point of infirm. He liked to try to guess what the callers looked like from the sound of their voices. It was the same mental quirk that made him work so hard on how he sounded on the phone.

Joe began the standard litany:

"Are you calling from the phone that has the problem?"

"Yes," she replied.

"And have you been able to receive calls from the outside?"

"Oh yes!" she answered without hesitation.

Joe wrinkled his nose. This wasn't going down the normal diagnostic path. Calls going-out are okay, calls coming-in are okay, and he could tell the connection sounded pretty good.

"What seems to be the problem, ma'am?" Joe asked.

"My dog barks when the phone rings," she replied.

Joe's mouth opened and closed soundlessly three times. The fish–out–of–water act finally ended when he composed himself and bought a little time with the standard "Excuse me, I didn't quite hear you" line. But he had heard her all too well.

"My dog barks when the phone rings," she said.

"Yes ma'am. I'm sorry, this is telephone service. Perhaps you need to call a veterinarian." Joe was smiling now. He could safely park this one in the category of those confused folks who wanted Telco repair to fix the refrigerator or the TV or some other "e-lektrick" thing.

"No sir. This is definitely your problem. I want you to come fix the phone." She was surprisingly firm and didn't seem confused. Joe tried again.

"Maybe your dog just doesn't like the sound of the phone. If you turn the phone over, there is a little silver wheel on the bottom that will make the ringer bell much quieter if you turn it all the way to the other end."

"No sir. My dog is outside. If I let him in the house, he doesn't bark when the phone rings. He just runs behind the sofa and stays there. I want you to come fix the phone." She sounded insistent. Joe paused to collect his thoughts. He didn't want to insult her, but she was obviously terribly confused.

"Ma'am, I'm sorry. We can only fix the telephone. You said that you can place calls and receive calls and the line quality sounds pretty good. I don't see what we can do for you."

She firmly replied, "The dog barks when the phone rings. Ever since you people put that new extension in my bedroom last week, he kicks up a terrible fuss when my friends call me. I want you to come fix my phone."

The hairs on the back of Joe's neck stood on end. He was ready to fight, wanted to tell this lady to kindly go stick it, but something was nagging at the back of his mind. His predecessor Fred had always run Joe through a quick checklist:

1. Did it ever work before?
2. What changed?

Joe was always surprised at how quickly this helped sort problems. But it didn't really fit here, since nothing was actually broken! There was something in her voice though, and something tickling inside his brain. But he couldn't put his finger on it.

Sighing deeply, Joe asked the insistent customer, "What is your address?" It turned out to be fairly close to the garage.

"I can come out at noon." He wouldn't dare send one of his guys on this one. They would never let him hear the end of it if he asked them to go fix a dog—a *real* dog, not a difficult problem. (Technical folks sometimes call tough problems "dogs.")

"Thank you," she said. As they both hung up their phones, he couldn't believe he had gotten himself talked into this one.

The rest of the morning was a blur. Joe's assistant covered the phones while Joe checked the service records for his dog-bark customer. Nothing unusual there, although he was able to see that she had indeed had a new jack put into a bedroom the previous week.

When lunch time rolled around, Joe was rolling up to the customer's house. A nice, modest, ranch-style home, it looked well maintained. There was a small front yard and a fairly large backyard with no fence either front or back.

A sturdy woman of about 50 (Joe couldn't help but smile; he had gotten that part right) answered his ring of the doorbell. After introductions and a few brief pleasantries, she showed him the main kitchen phone and then led him to the new extension phone in the back bedroom.

From there, he could see out a window into the backyard. A medium-size dog was leashed to a dog-run cable that ran from the back of the house to a large maple tree that graced the rear yard. A nicely painted dog house and a bowl of water sat under the tree. The customer told him that the dog was a Humane Society adoption, mostly collie, but some other breed mixed in as well.

Joe dialed a special number for an automatic central office callback. He re-cradled the handset and waited. On the first ring, the circus began. The dog appeared to be making a serious attempt at powered flight. Yelping and barking and twisting, it cleared a good 36 inches between itself and the ground before gravity won the battle. The second ring induced a leap and barking fit that made the first appear small. A thin trail of urine appeared to follow the poor creature's arc through the air like some perverse fountain statuary.

Joe shook off his astonishment long enough to slam down the handset and stop the rings. The dog continued its plaintive yelps for several minutes.

"You see," the customer repeated. "The dog barks when the phone rings."

Joe saw all right, and now he was pretty sure he knew what the problem was. He went out the back door of the house and checked the dog-run wire. As he expected, it was a length of aluminum aircraft-control cable, joined neatly at each end to an eye hook. At the house end, the eye hook was screwed deep into the clean siding. Joe walked up and tapped the siding. "Aluminum siding," he said to no one in particular.

Now he was sure he knew what was happening. He walked around the back of the house, pausing to fidget for a moment at one place near the back bedroom window. He returned to the customer, explained to her that he had fixed the phone and the dog should not bark when the phone rings in the future. A quick re-test confirmed his statement.

When Joe returned to the garage, he related his newest parable to all the repairmen. He told them that his repair had only required a bit of electrical tape and some non-conductive goop.

The conclusion to his story went something like this:

When the new extension phone was installed, the new wire was simply run along the back of the house where it went into the bedroom through a small hole drilled in the exterior siding. To keep the wire straight and neat, it had been stapled in place at several points along the outside back wall. But the sharp edge of one of the staples had nicked the telephone wire insulation, and that wire was therefore shorted to the aluminum siding.

The conductive metal siding was shorted to the dog-run cable, which in turn connected directly through the dog's chain to his metal-studded collar. The dog was of course grounded.

When the phone rang, the 90-volt ringing energy was being routed right through his poor little body, including any part touching the ground! Wouldn't you bark too if that happened to you?

Chapter 2: What Do You Know?

The phone rings at *your* desk. A person at the other end says, “It does not work.” You say, “What do you mean, it does not work?” They reply with something utterly uninformative, such as, “I mean it is broken. It does not work.”

I have seen and heard written and verbal exchanges go on like this for days or even weeks. Such conversations would actually be funny if they were not so frustrating, stupid, anger-inducing, wasteful, and oh, did I mention *stupid*? (Important note here: It is not the people who are stupid. In fact, I find this happens far more with very smart people. It is the conversation that is stupid. Somehow, these very smart people get locked into goofy word exchanges that contain no useful information!)

Let’s break it down. The first word in the statement “It does not work” is the word “it.” You don’t know what “it” might be from that sentence. Of course, from the point of view of the complainer, the “it” is obvious. “It” could only refer to the last thing on which the complainer was working. “It” is very fresh in this person’s mind. He wonders how stupid the person he is talking to could possibly be if that person does not understand what “it” is. He might think, “There is only one thing that is shared between the two of us, and that is the project that includes ‘it.’” The complainant simply cannot conceive that you don’t already know what he is talking about.

The next words, “does not,” are usually straightforward. That phrase expresses an active negative behavior. Fred *does not* go to the grocery store. The car *does not* start. This painting *does not* look realistic.

Every one of these preceding statements is more useful than the complainant’s, however, because they include two additional facts. First, they identify a specific entity (Fred, car, painting) instead of an ambiguous “it.” Second, they state a much clearer action than just “work” (go to the grocery store, start, look realistic). So you have already found two problems in the original four-word statement. The only thing you clearly understand up to this point is “does not.”

Eventually, however, you realize that even these slightly better short statements leave out a lot of information. Fred may not go to the grocery store during the week, but does he go on weekends? Or does he *never* go to the grocery store? When one says that the car does not start, does this mean that the starter motor does not try to spin the engine, or does it mean that the starter motor actually spins the engine but that the engine does not fire and stops spinning as soon as the starting key is released?

I could go through endless examples to illustrate, but let me get to the point. A good statement of a problem, or *problem statement* will address all of the following:

- What?
- Which?
- How many?
- When?
- Where?
- Who?
- How?

Let’s talk about these in order. Every one of these questions is critical to our method of problem solving. Some of them might even seem obvious when presented in a nice, neat order like this. You will rarely receive all of the answers to these questions in the initial contact about a problem, however. An extremely mature development organization will incorporate these steps into their problem-solving process, but for most companies, you will be lucky to receive more than a few of these facts.

What?

In the contact described previously, the “what” was represented by a completely ambiguous word: “it.” No matter how painful or how tiresome you find the exchange, you absolutely must force the person reporting the problem to clearly identify the "what" of the problem. In the case of a released product, you need a model name or number and a part number. Model names tend to identify a general type of product (Ford Mustang, Apple iPad), but they do not identify a particular model year, revision, production lot, date code, or other information. For these things, you often need a specific model part number, a serial number, or both.

Well run organizations have methods in place to identify and track these things. Badly run organizations do not have such methods and might not even understand why such identification and tracking is important.

Which?

During development, many organizations use different names to identify product samples that come from various steps in the development process. In some companies, part numbers are assigned that use digits to indicate development versions and letters to indicate production versions after the product has been released.

For example, the power delivery assembly (PDA) of a Ferndoggle 500 might carry a part number of 1234567-001, -002, or -003 for early development samples. The production (released) versions might be 1234567-A, 1234567-B, or 1234567-C. Assuming these PDAs are marked with their part number, a person trying to report a problem with a Ferndoggle 500 will be able to instantly tell you enough information for you to know whether that person is working on a development or production sample and which version he is using.

> Accurate software version labelling is equally important to letting you understand *which* system you are debugging.

There are many ways for parts to be marked with identifiers. Part markings can be placed on physical labels (paper, plastic, metal, cloth), which are attached to the part or assembly. Part markings can also be made with ink stamps. Those inks can be dye or pigment based. Metal components might use an embossed or stamped (impact displacement or indenting of a small part of the metal surface) part marking. Plastic parts or molded metal parts can carry markings from the mold tool on production parts.

Parts that are very small often carry no marking at all. In this instance, the production process must be extremely well controlled to ensure that the wrong part is never used. Otherwise, there is no simple method to determine that the correct part has been used in a given sample. You could remove and measure the physical performance of every component in an assembly, but that is extremely time-consuming and inefficient.

Imagine the pain of removing 600 capacitors and 300 resistors from a circuit-board sample and then measuring the basic performance of each part to try to find which component position was loaded with an incorrect value. It is far better to take many extra steps to ensure that the correct parts have been loaded into the correct positions for all samples.

A good general rule of thumb is to clearly mark significant assemblies with identifying part numbers and version numbers that can be traced back to the design documents for that version. This implies that you have a good version control (and/or revision control) and an engineering change notice (ECN) system in place that lets you associate the appropriate design documents (block diagrams, schematics, bills-of-materials, source-code versions, and tool versions) with any given sample.

Engineering change control systems might not seem to be terribly important to you when you are a young, eager designer.

Aren’t these things just extra paperwork to keep the managers happy? No. They are critical steps to helping you debug your design problems. Many different systems can be made to work well, but be sure that your organization has some kind of system in place.

How Many?

This is one of the most important questions you can answer about a problem you are trying to solve. Does the problem happen on all samples? Does it happen on 99.9% of your samples? Those two are not the same answer! One of the worst situations you can have is that your first development sample had the terrible luck to be the one assembly that worked. You know it worked, you can see it worked, but none of the other samples have worked (or might ever work)!

This kind of failure can happen if your circuit needs a given transistor to have gain greater than 50, but the distribution of transistors coming from the factory is such that most of the production has a gain between 20 and 40. A few units have gains above 40, and you were just lucky (or unlucky) enough to have gotten a high-gain sample.

The same kind of failure can occur with mechanical designs. Tolerances can add up in tricky ways. A design that had excellent fit in the first prototypes might show terrible slop in early production.

Likewise, you might find a problem that occurs in only 1% or 2% of development or production units. At this point, you don’t care what the numbers might be, but you need to know: How many are good and how many are bad? Also, are there any units that are somewhere in-between?

When?

Some problems are completely independent of time. For example, “This bucket leaks” is a problem that might seem unlikely to have a time associated with it. If the hole is in the bottom of the bucket, that bucket will probably always leak. But if the hole is exactly half-way up the side of the bucket, an accurate problem statement would be, "This bucket leaks when it is more than half full."

The problem statement that includes a time reference (when) gives you some good information. You are now very likely to look up along the middle of the side of the bucket for the hole.

Does the problem happen only on Tuesdays? Does the problem happen only when the door to the room is open?

You want to be specific and clear about when a problem occurs, but you also have to be very careful about correlating the problem to external factors. A problem that happens very frequently (always, perhaps after a few seconds of operation) might appear to be related to some other event simply because you are also causing that external event at the same time.

You can easily get into superstition about "when."

You might say something like, "I waved the chicken bone just after starting the system and it worked okay." So maybe you need to wave the chicken bone over this system for it to work well! Does that really sound like clear thinking? Often, you simply did not try enough cycles to know whether there was really a correlation between chicken-bone waving and success.

Occasionally, you might actually find a crazy correlation between something like waving a chicken bone and success. In the absence of a good explanation, you start to believe in a bit of magic. Maybe the CPU really likes chicken? I call this magical thinking. But eventually, somebody comes along and explains that the system is showing exquisite sensitivity to temperature. The act of waving *anything* nearby was enough to cool the CPU and keep it working for a little longer.

Where?

Some problems seem to happen only in one place. Conversely, some problems happen everywhere *except* one specific location.

This kind of clue is critical to debugging this type of problem. It can mean that the problem is truly caused by the location, or it can mean that the sample is sensitive to some condition of the location that was previously not understood.

If your product fails in Alaska during January or in Arizona during July, you might want to think about doing some extra temperature-range testing for extreme cold or hot.

Sometimes, the location-sensitivity is not so obvious. I once heard a description of some video monitors that were failing in a particular installation in a large city. The monitors would work okay for a while, but then would suddenly show drastic image distortion.

It turned out that a subway train ran just behind the wall where the monitors had been installed. The giant electric traction motors used to move the trains created a huge magnetic field, which disrupted the electro-magnetic deflection of the monitor electron beam. An unusual amount of magnetic shielding would have been required to make the product work in that location. A simpler solution was to move the monitors away from that location. In summary, one critical clue to the cause of this problem was *where* the problem happened.

Who?

One important bit of information you need to collect is whether the failure is specific to a single person or a specific category of operator. Sudden, unintended acceleration of automobiles has been shown to correlate strongly to certain age groups. This does not mean that a specific person or category of user is somehow to blame for the failure. But you might find that some previously unobserved characteristic of that user (or type of user) directly contributes to the problem.

I have noticed that some product testers have a knack for “breaking” what seemed to be working products. They often enter rapid or nonsensical sequences of commands, just to see how the system responds. A well-designed system will detect the incoming overflow or meaningless command sequence and will recover gracefully and quickly. A mediocre system might take a long time to recover. A poorly designed system might lose information or respond in unexpected ways. A badly designed system will crash.

Often, young designers will say “Stop! You are using an unfair input! You have to wait longer between keystrokes!” or something similar. But the best designers welcome and greatly appreciate such test sequences. In fact, they will try to define and incorporate such tests into their product testing.

How?

Answering this question is probably the most important step in the problem-statement sequence. It is here that you describe the exact sequential steps necessary to reproduce the problem. If your “how” statement is accurate, then anybody using the same steps with the same samples should be able to reproduce the problem.

There is a bit of artistry and a lot of science to writing a good “how” statement. You must convey to a skilled practitioner the steps he needs to take, but you don’t need to teach him every single aspect or the use of every single tool.

You don’t need to explain how to hold a pencil when you ask an experimenter to write something down. But you might need to give him a list of the values you want him to record (such as date, time, temperature, or velocity).

Why I didn’t include “Why”

Shouldn’t you immediately include “Why?” in your list of questions?

Wait! Don’t answer that one. I was checking to see if you were paying attention. You are only allowed to answer the question “why” after the problem is actually solved. “Why” is never part of a good problem statement. The problem statement comes first; then you can work on solutions.

"Why" tells you the cause of the problem. If you have enough information, enough tests, enough proof, you can finally make a statement like, "The bucket leaks because there is a hole in the bucket, five centimeters above the bottom of the container. This has happened because we found a flaw in the plastic mold that created a weak spot in the sidewall."

The worst thing you can do is guess ahead and be wrong. When you leap to cause early, you typically eliminate many experiments and avenues of investigation. If you are not willing to consider those possibilities (and when you are wrong in those first guesses), you will find yourself frozen. You cannot move forward because you have found that your first guess kept you from collecting the information you needed to make a better analysis.

Politically, you have made a mess by including a guess at cause. The boss says, "I thought you said the problem was *X*." He probably then passed bad information up the chain of management, maybe even to a customer. Suddenly, everybody is scrambling to undo misinformation. Maybe 20% of that management group will get the message that the first "why" was wrong. Do you know who now has to spend a lot of time re-educating the other 80%? *You* do—and your next guess had better be a lot stronger. But if you had not made a first guess, nobody would be expecting you to have a second guess—or a good explanation at all—until the investigation was complete.

"Now wait a minute," you say. "The way we always find the solution to our problems is to do a brainstorming session where we list all of the possible causes and then try to figure out good experiments to prove or disprove each of them. So we end up with a whole list of possible 'why' answers, and then we eliminate them until we find the one true answer."

Yes, I understand your argument. There is nothing really terribly wrong with your problem-solving method. However, you must separate that list of possible causes from the problem statement. Everything in the problem statement should be a verifiable fact.

You must keep the beliefs, theories, superstition, and voodoo separated from the facts. Otherwise, you end up muddying up the waters.

You sometimes can't see so clearly because you are trying to look through a filter of belief and non-belief. How many times have you heard somebody say "I cannot believe the problem is *X*!" and yet you eventually find that the problem was, indeed, X?

False Assumptions, or "What You Know That Just Ain't So..."

You must always be vigilant about what you know. Every so often, something you assume to be true, is simply false. This kind of mistake can be incredibly costly to a debug effort and can lead you down long roads of wasted effort. You might try to establish the cause of something that is not happening. You might spend a lot of effort, time, and money chasing an idea that never was accurate in the first place. If you have doubts about a "fact," be sure to note that clearly in your list of *things you know*. This can save a lot of time and embarrassment later.

The Old Man

I first read this tale of "The Old Man" by George Rostky in the pages of *Electronic Design* magazine. I consider it a privilege to have learned so much by reading the late Mr. Rostky's editorials, week after week and year after year. His wisdom lives on in his writings. It is my fervent hope that someday his collected works will be published as a single book. My retelling of this brief story is a weak mirror of the original.

⏩ *Once upon a time,* a group of businessmen gathered in a local bar to decompress a little after a particularly difficult week.

The first one said, "I wish I could relax like that old codger hunched over at the end of the bar. Look at the line of empty glasses in front of him. I'll bet that is what keeps him young, all of the drinking! Maybe all the alcohol has pickled him from the inside."

The next man said, "No, way. That guy has been chain-smoking since we came in here. The barkeep has emptied that ashtray three times and he already has it half full again. The nicotine high must be keeping him going."

The third businessman said, "Nah, it is the bad food. He has been pounding down greasy burgers and fries like there is no tomorrow."

A fourth said "Hey, I was in here last month and also last week. He is always here, and there is always a beautiful 40-something woman with him. It has to be the women. That is what is keeping him so youthful."

Finally, the argument got the better of them. They approached the old guy. "Hey old-timer," one started. "We were just debating what might be the secret to your longevity. Is it the cigarettes, the booze, the food, or the women?"

"Well, perhaps I owe it to simply living so long," he replied with a bad-toothed grin.

"No, really, we want to know because we can't agree. But first, maybe you could tell us just how old you really are?"

The wrinkles on the man's forehead deepened and he squinted at them through narrowed eyes, obviously digging through his mental haze to pull up the number of his last birthday. Then nodding as it came back to him, he replied with surprising clarity and obvious truthfulness.

"I'm 24" he said.

Looking ahead, there is something else that you know—or that you *should* know: a set of rules that tells you how the known and unknown parts of your problem relate to each other. Mathematically, these rules form equations or relationships.

But before I discuss those rules, I need to dive into a really critical part of solving problems: clear communication.

Writing a problem statement

A good problem statement collects together this critical information:

1. **What?**
2. **Which?**
3. **How Many?**
4. **When?**
5. **Where?**
6. **Who?**
7. **How?**

When writing a problem statement, you should clearly note items that are suspect. At the same time, watch out for false assumptions and avoid jumping to any conclusions this early in your investigation.

Chapter 3: Communicating Clearly

⏩ *Once upon a time,* a senior-management visitor from Europe, whom we shall call Mr. Big, came to an engineering facility in the eastern United States. His company had purchased an American organization including several technical development centers. There had been several setbacks in a recent project. Corporate technical planning had been too optimistic, and now the project faced cancellation. In brief, Mr. Big was there to meet the team and assess their chances for success—or perhaps to kick asses and take names.

Walt was a local American manager who had worked in the European headquarters for some time before coming back to the United States. He had an ear for the language and could usually convert somewhat fractured English to American-speak without much trouble.

Walt's team included some key engineers, including Mickey, a brilliant electrical engineer with a quick wit and irrepressible good humor, and Ralph, an experienced mechanical engineer with very little sense of humor and a provincial, if not downright xenophobic, attitude toward the European owners. The third key team member was Kevin, a very sharp printed circuit board layout engineer who preferred fishing to most everything else in life. Kevin was even-tempered and quiet: a good listener who always did his job and rarely complained.

Walt led the visitor into the lab, where the trio of key engineers waited.

"It eeze nice to meet you," Mr. Big stated and hands were shaken around the room with brief introductions. "I understand zat you will be ze team to fix our big project problem?"

"Yes, these are our best engineers," Walt quickly reassured Mr. Big. Walt gave a brief summary of how the project had been handed to his team without their participation in any of the scheduling.

Walt then mentioned the three specific problems they had encountered while trying to finish the design. Two of these had been solved quickly, but the third was still in the process of being unraveled.

Mr. Big listened carefully, but his shoulders hunched up in a pose that conveyed unhappiness. His eyes stayed on Walt, but then shifted to the trio of engineers. Walt glanced at Ralph and could see that his face was tense.

Mr. Big said, "We must fuck youz to reach our goal."

Ralph's face turned bright red and the veins in his neck popped out.

The corners of Mickey's mouth twitched. Ralph leaned forward slightly, and Walt rotated his own left shoulder forward a tiny amount, placing it between Ralph and Mr. Big, just in case.

"The shower doors are counting on this project to improve the value of their chairs," Mr. Big continued, "but they will not wait forever."

Kevin scanned the room, looking at the desks and chairs. "What the heck," he thought. "Who cares about the chairs?"

Mickey fairly bounced on the balls of his feet while Ralph's jaw tensed further. Walt leaned in quickly and guided Mr. Big toward the test facility down the next hall.

The trio volleyed their confusion once the visitor was out of earshot.

"Well that was special," giggled Mickey.

"I should have punched that jerk," growled Ralph.

"What was that all about?" wondered Kevin.

Shortly, Walt rejoined them. He had passed Mr. Big to the next local manager who would demonstrate some of their new software-development tools.

"Okay guys," said Walt as he came back into the room. "I could see we had some trouble with that little speech he gave."

"Some trouble!" exclaimed Ralph. "Did you hear what he said they were going to do to us? What a butthead!"

"Wait, calm down," Walt replied. "I think you heard what you expected to hear. But that was pronunciation, not denunciation."

They chewed on the meaning of that for a few seconds. Walt had a way with words.

"He said we need to *focus* to reach our goal," Walt explained. "It's just that his pronunciation made it sound like something completely rude."

Ralph's posture relaxed a bit. Mickey was already laughing out loud.

Kevin was still perplexed, "What was that nonsense about shower doors and chairs?"

Walt grinned, realizing how Mr. Big's foreign accent had mangled his meaning. "He was saying that the *shareholders* (not shower doors,) are impatient. They are worried about the price of their *shares*, not chairs."

Communicating Clearly: Verbal Discussion

One of the first barriers you will encounter to problem solving will be poor communication.

Remember that example of a lousy problem report? "It does not work." Yuck! This makes me physically sick to think about! Yet, it is one of the most common headaches in solving problems. You cannot get many people to communicate a clear problem statement. They use indefinite articles (it) and nearly meaningless generalizations (does not work).

For so many statements that I have heard or read in early problem-solving reports, the appropriate response should really be, "What the hell was that supposed to mean?" Unfortunately, such a response will only make later communication even more difficult. You need to collectively pause and then find a better way.

You are probably going to find yourself in situations in which you have to conduct an interview, either face to face or on the phone. You might have to ask many questions to extract a clear meaning. People will wander off into unimportant discussions that mean something to them but don't help you solve the problem. You can politely ask them to try to limit their responses to very short, factual statements. It won't help much, but you should still try.

You also need to remember that when you first learn about a problem, you will be full of many misconceptions (bad ideas) about what the other person is trying to tell you. You probably don't get what he is saying. It might take several tries before you understand any of his statements.

This is why breaking the discussion down into those little pieces is so important and can be so helpful. First, ask the person to identify the "what." Collect all of those facts in short little pieces. You can be friendly and encouraging. Psychologists would say that you are getting the other person into an affirmative mode. You don't want to fight; you just want to help the other party help you. Everybody wins because you are making progress, even if it seems slow at the beginning.

One of the barriers I have encountered is a dependence on verbal communication. We all live inside our own heads, so the words we say sound just fine to us when we say them. I know what I mean by "it" and by "work," so the statement "It does not work" makes perfect sense to me when I make it. But when it is written down, maybe I realize that there is not so much useful information there.

A significant difficulty in multi-location (sometimes international) problem solving is that even if every person in the conversation is capable of speaking and understanding the same language, they probably don't share the same collection of conversational idioms. These are those short collections of words that might have a very clear meaning within a small, local group of speakers, but are impenetrable to remote persons.

In English, we might talk about "pulling somebody's leg," but we really mean we are teasing them or making a joke by telling them something that is not true.

Or we might say it is "raining cats and dogs," but we are not being literal. There are no small animals involved. (The original source of this expression did indeed involve animals, but today we just mean it is raining very hard.)

It can be fun to try to list all of the common idioms that get used in a five-minute stretch of conversation during a meeting. We rarely speak as clearly as we think we speak.

Communicating Clearly: Conference Calls

Big, multi-location corporations seem to be in love with conference calls. It is true that from time to time, a conference call can lead to that "aha!" moment, where some previous statements in an email or relayed phone message suddenly make sense. That being said, conference calls often pull in multiple layers of management, which results in so much management-speak that any real ability to communicate is quickly lost. Hours of time and many person-hours of effort are wasted during endless regurgitation of useless information.

A few rules must be in place if you want to get anything useful from a conference call. First, have one person at each location or endpoint do a quick introduction of all the people on the call.

Also, unless managers are also taking on specific problem-solving roles, they should mostly stay quiet—very, very quiet. They are welcome to speak up if it is to volunteer a specific resource to help solve the problem.

One person must be assigned to take notes and publish a summary of the call. Be sure to include contact information in the summary of the call. A good conference-call summary will always list specific action items coming out of the call. These action items must include the following details:

- What is the action?
- Who is assigned to perform the action?
- When should that action be completed?

Don’t worry, there is plenty of flexibility in such lists. If the action is, “Measure the output DC bias and signal voltage seen at the collector of transistor Q101,” and you find that you should have been talking about Q201, then it is easy to publish a correction or explanation. If the action is assigned to Fred, but he passes that responsibility to Sam (who agrees to perform the action), such reassignment should be okay. If the action is due by Thursday, but you don’t finish until Monday, then maybe you are in trouble with your boss—but hopefully you gained enough information over the weekend to help solve the problem.

Communicating Clearly: Email

At one time in my life, I believed that email should be the universal answer to good communication. Just write it down! Type a few words into the computer, hit *Send,* and there should be no problem with clarity. Boy, was I wrong.

One of the main problems with email is that it does not convey the sender’s body language or the tone of voice behind the words. Sarcasm, satire, irony, and humor all depend on everybody in the conversation having a clear idea of the true intention of each statement. But simply receiving a neutral email can lead to a surprising number of misunderstandings.

If you capitalize SOME SPECIFIC WORDS to add emphasis, people think you are shouting at them. If you use **bold characters** or *italics* for the same reason, some readers will assume that you are being snide or sarcastic. If you copy some words from their previous email and put those words in quotes, they might feel you are attacking them, using their own words as a weapon. (And to be fair, maybe you were.)

I continue to be amazed at how many ways people can misunderstand a few simple words. I am also always amazed at how something I wrote one day can suddenly look and sound completely different—even to me—on the next day.

The best advice I can give is to keep email and written reports as short, simple, and factual as possible. Delete any editorial comments from your written summaries. You don't need to say "Obviously, this product is not ready for the customer." Unless you have been asked for a direct opinion as to the readiness of the product, you should stick to the facts of the failure.

Let's pretend that your boss directly asked you, "Is this product ready for the customer to see it?" You can respond, "In my opinion, this product is not ready for the customer." Just dropping the word "obviously" and qualifying the statement as being your opinion (not fact) can make a huge difference in how that statement is viewed later by a wider audience. In the first example, you might be seen as an arrogant pile of excrement, in the second example you are simply a careful, thoughtful employee responding to a management query.

I am frequently surprised by how often engineers fail to use basic tools that enhance communications. Simple block diagrams can quickly clarify where in a system a problem exists or is being observed. Flow charts, schematics, timelines, graphs, instrument captures, and even little cartoons can help folks at the other end of the communication understand what you are trying to convey.

That being said, stay away from jokes in written communication about a problem. People from other countries and other cultures often simply do not understand a clever play on words.

Also, if the problem turns out to be extremely serious (for example, people are injured or killed by your design), those jokes can come back to haunt you in court or in the general perception of the public.

Communicating Clearly: Photographs

When you have a test setup, take a photograph of the basic environment. Very often, this can immediately lead to solutions to problems. For example, the person at the other end might realize that although you have been referring to an output at connection *X*, you actually have the cable plugged into connection *Y*.

Photographs are also great at conveying some specific details, especially of things that are too small to see easily unless you have a magnifier or microscope.

Communicating Clearly: Videos

One of the quickest ways to simulate having the person with whom you are communicating sitting next to you is to record some video of the test or demonstration. You might need to restrict the length of the video or keep the resolution fairly low if you have a bandwidth limit for the transfer of the video. More resolution is better, but longer video may not be—especially if it just shows you sitting there thinking or talking about what you wanted to demonstrate. Keep it short, keep it sharp, and keep it smart.

Critical Question: Has it Ever Worked Before?

One of the most straightforward questions you can ask is whether the system in question has ever worked correctly before now. Although the question is straightforward, finding the answer often is difficult and complicated.

For one thing, people working on a new design simply may not know or might not recognize that the system never has worked correctly. In the excitement of bringing up a new video player, they might not have ever actually listened to the audio! In a high-pressure, quick-paced development environment, it is not uncommon for thorough testing to be delayed or pushed toward the end of the project.

The thinking becomes "We have seen that part work, so let's move on to the functions or features we have not implemented yet." Complete testing checklists are skipped over due to pressure to reach design-complete status.

Likewise, even very mature products can contain surprising bugs. If the system depends on an outside device that is not complete or does not even exist at the time of the initial development, everybody within the design team may simply assume that the interface works (or *will* work), and the product goes out the door having never passed a functional test with an external device or simulator.

Critical Question: How often has it worked before?

Even if a feature or interface has had some amount of testing, we all have experience with designs that turn out to not play well with large numbers of real-world devices that have been developed by other design teams. As design and interface complexity grows, this kind of situation arises more and more often. One way designers attempt to cope with this is to test their product against as many examples of third-party implementations as possible.

For example, certification of devices with USB interfaces typically requires participation in a "plug-fest," where manufacturers are challenged to demonstrate that their device can work correctly with (appropriate) devices from all other manufacturers. So an inkjet or laser printer might need to show that it can talk to USB ports on a variety of different motherboard implementations. A set-top-box design might need to demonstrate that its HDMI port can successfully work with a large array of TV set brands in various modes.

Such interoperability testing places ever-growing demands on designers and their companies. A library ranging from poor to excellent implementations of "the other end" must be maintained. The costs of such collections can quickly spiral out of control.

Listening to the Customer.

One of the most difficult things we are asked to do as designers is simply to listen to the customer. As I will discuss later, there is a built-in assumption that we must be smarter than the customer. Otherwise, the customer would not be asking us to design or make the product, would he? If he were as smart as us, wouldn't he just design this product himself?

Nope, not true. There are many reasons a customer might choose to purchase instead of build his own product or system. It is critical to the success of any enterprise that its representatives learn to listen to their customers and to value the feedback they are getting. The customer is trying to tell you what he wants. That message is absolutely critical to you, because the customer is the person who pays you real money for what you are selling. If you don't listen and respond, he will quickly find somebody else who will.

When I ran some electronic businesses in the past, I found that the customers I liked the best were the ones who truly *did* know more than I did about what products they wanted and needed. Their advice was smarter and more useful. They also recognized the value of using my company to provide products that they might have been able to build themselves. If nothing else, buying from my company assured them that they would not receive bad units, which we had carefully filtered out in production testing.

Active Listening

Short definitions of active listening typically include concepts like feedback, summarizing, and openness. You work hard at absorbing and embracing what the other person says. Many classes, books, and online references teach the skill of active listening. You should find one and learn this skill. It is an integral part of putting together a problem statement.

In some way or another, the person reporting the problem is your customer—or maybe they are an internal stand-in for your customer. You need to collect this information quickly and accurately. You might need to work out a common vocabulary, but you must get this human-to-human interface working to get a clear description of the technical problem.

How Was the System Supposed to Work?

The best descriptions of a problem must include some discussion of the expected system behavior. More importantly, you need to write these statements down, not just think such wording to yourself or say it in a meeting somewhere. Write, "When I do *this*, I expect the (*system, object, or person*) to do *that*. Instead, I find that it is doing *this other thing*." (Hopefully, you will get more specific identifiers than *this, that,* and *this other thing* from a good problem description!)

Often, I find that I am the person reporting the problem—and sometimes I find that I am reporting the problem to myself. Oddly, that makes it all the more important that I use clear, unambiguous language to describe the expected behavior. It can be too easy to take shortcuts when you are just talking to yourself. You already know what you mean and already know what you were thinking, so you don't take the time to write it down clearly. That is a mistake. Eventually we all need to communicate our problems and solutions to other people.

Maybe you will need to write a patent application to describe how you overcame a problem. Or maybe you will need to warn the customer not to make a common mistake in your user's guide. Or maybe you just need to keep your managers aware that you are earning your paycheck. If you cheat here, you are just cheating yourself. These descriptions of expected behavior are absolutely critical to problem solving.

Your clear problem description might also become part of marketing literature later in the life of a product. Your sales or marketing departments might be able to promote your solution as a key feature that your competitors do not include. In some cases, clear problem descriptions can make their way into user manuals, in “How to Use It” sections or maybe in “How NOT to Use It” sections.

Different settings create different communication problems.

1. Verbal Discussions
2. Conference Calls
3. Emails

Use these tools to improve clarity and understanding:

1. Photographs
2. Videos
3. Block Diagrams

Key Question: Has the system ever worked before? (What evidence do you have?)

Hints:

1. Listen to your customers.
2. Practice active listening.
3. Be sure you can write a clear description of how the system is supposed to work.

Chapter 4:

Barriers to Good Communication

Customer-support groups are often filled with stories of end-user stupidity. Like the guy who couldn't figure why his printer would not work, but had never plugged it into a power source. Or the tale of the lady who inserted disk after disk into a floppy drive during a software setup, but never realized she had to take the previous disk out before inserting the next one. Her complaint was that she could not put disk 4 into the drive.

Sometimes, customer-support groups have special codes for these users, such as "PEBKAC," meaning "Problem Exists Between Keyboard and Chair."

The "People Are Dumb" Automatic Bias

Given these stories, it probably comes as no surprise that one of the biggest barriers to good communication about any problem is that we all have a tendency to assume that the guy at the other end of the conversation is just not smart enough. By this, I mean we think the other guy is not so bright. Dumb. Stupid. Idiot. Moron. He has a head full of rocks. This is so obvious to me, why can't he see it clearly?

But the following is true no matter which side of the support desk you are sitting on: I am sorry to say, the reason the person on the other side cannot understand what you are saying is almost certainly because you are not saying it very well. You probably have left out several of the items in the where/what/when/who/how list. You probably have not used all the great tools at your disposal, such as photographs, videos, charts, and drawings, to explain how to use the product. Or, if you're the one with the problem, maybe you have not used similar tools to report it.

As much as anything else, you probably do not share exactly the same vocabulary for your description. You might be talking about processor exceptions, but the person on the other side is thinking about hardware interrupts. You are talking about fasteners, nuts, and bolts, but he only understands that he is getting screwed. You are talking about boards, but he became bored and angry a long time ago.

When you find that your conversations (whether face to face, via conference call, or written) are not getting you to a place of common understanding, stop and think hard about the ways you are communicating the various aspects of the problem. You might even need to write a glossary—a dictionary of the technical terms specific to this project and this problem. Make sure that everybody agrees on the terms. If other parties are using words or concepts that don't make sense to you, make sure you stop them and ask for explanations—preferably as well-written as you are providing to them.

Look, if they are uninformed, it is because you are not doing a good job of educating them. "Hey, that's not my job," you retort angrily!

"Get over it and get used to it," I reply—and your boss is with me on this one. It is always your job to teach and to elevate the knowledge and understanding of those around you. It is what makes a successful enterprise. If you are the only one who understands something, then you are actively failing. If you hide away knowledge, then you might think you cannot be replaced. But if you cannot be replaced, then you cannot be promoted. You will be stuck in one place—and maybe not for as much time as you hope.

> There are occasionally organizations that cultivate chains of incompetence or islands of restricted knowledge. They rarely survive for long.

Jargon

Jargon is a double-edged sword in most businesses. It can work to your advantage when everybody is carefully trained to share the same meaning for obscure acronyms. In this case, jargon speeds up your daily workflow. “Hey Ralph, Bring me another 247,” is fast and efficient if Ralph understands that what you want is part number 123456789-**247**.

But it can lead you into trouble if Ralph is the new guy and brings you part number **247**456789-001 (which looks nothing like the part you need).

I once sat in a project meeting at a company where I was the new guy. It became clear to me that our competitors could have sat in this same meeting and learned almost nothing about the state of our projects. The discussion was carried out almost entirely in jargon and acronyms that were specific to internal company operations. Every company location had an obscure multi-letter code, and every project milestone had a similarly obscure acronym. After the meeting, I sat down with a list of codes and acronyms I had captured and asked anybody who would give me a few minutes of their time what each bit of jargon represented.

Pretty quickly, I became an “insider,” and could decode the flow of the conversation in real time. But that first meeting was brutal, and could have led to a lot of misunderstandings.

“It Works Fine for Me” Automatic Bias.

Have you ever found yourself saying something like this: “I have never seen *this thing* do what you say it is doing, so you must be doing something wrong.”

At some time or another, we all fall into this trap. It is an absolutely guaranteed method to infuriate your customer, your spouse, your friend, or any other living, breathing, human being. People just don’t like to be told that they don’t know what they are talking about. They don’t like to hear that you think *they* must be the problem—that you think they are not smart enough. Let me tell you a little story about that.

The Magical Taxi Trunk

⏩ *Once upon a time*, Tony drove his taxi into the service bay of the taxi company where he worked. It was the early 1950s and guys in the Big Apple were happy to have some steady work. Tony was middle-aged, thickening a bit around the middle, and had an easy laugh that made people like to be with him.

On this day, however, Tony was visibly upset. "Hey! Which one of you jokers is messing with me today?" he exclaimed. All of the mechanics looked at each other with slightly confused expressions. If they had conjured up a good prank, they would have shared it with each other long before this.

Interpreting the blank looks as poker-face bluffing, Tony launched into a diatribe. "Look, I can take a joke with the best of them, but this is not funny. You clowns have rigged up my taxi, and it is costing me a lot of time and that time is representing real money to me!"

Sam, the senior mechanic in the shop, stepped forward. "Just what is it that you are having a problem with Tony? You know I don't let anybody mess with our vehicles."

"One of you guys is trying to have a big laugh at my expense. Most of the time when I press the Talk button on this new radio, my trunk opens up. This started as soon as I got about two blocks from here and kept happening every 10 minutes, all morning long. I get out and close the trunk every time, but it happened again and again. I also think you broke my exhaust system, 'cause I can smell it every time I get out to close the stupid trunk."

Sam tried to reason with the irate driver, "Now just a minute. You know that we don't have any kind of automatic trunk release on these cabs, Tony. What you are telling me *is just not possible*. Maybe you aren't latching it right or slammed it so hard one time that you damaged the latch."

Tony was getting red in the face. "Don't tell me I don't know how to close a trunk! I have been doing this for a long time, and the car never did this stupid thing before! And it only happens when I press the Talk button on my radio, but it didn't happen when

I drove over the big bump over on 38th Street, so don't start telling me I don't know what I'm talking about!" Tony's voice was now louder and higher pitched than when he had started, and his face was shifting from red toward a hint of purple.

Sam made another try, moving to within arm's length of Tony. "Now calm down, buddy. Nobody here is messing with you. But you have to understand that what you are telling us is impossible. It can't happen. I'm sorry, but you must be wrong."

Big mistake. Tony hated to be told he was wrong. Eyes bulging, Tony shouted, "IMPOSSIBLE? I'LL SHOW YOU IMPOSSIBLE!" He turned sharply and jumped back into the front seat of the taxi. He closed the door, picked up the relatively large microphone, and turned back toward the semi-circle of mechanics facing him, stretching the coiled cord across his chest. He smiled a bit and then depressed the Talk button.

BANG!

It took a few seconds for the echoes of the shockingly loud sound to stop reverberating through the shop. Every mechanic stared open-mouthed, for they not only had seen the trunk pop open, but from their viewpoint, had also seen the enormous fireball that had exited the trunk.

Tony was already coming back out of the driver's seat, "Impossible? I got your impossible here! What do you think of that, smart guys?! How come you got nothing to say all of a sudden?" Tony's view of the fireball had been blocked by the trunk lid. Now his nose crinkled up and he suddenly exclaimed, "Hey, now it stinks like that exhaust smell in here too!"

Sam reached out, his composure only partially recovered. His voice was still a bit shaky. "To-Tony, you were right. And you were very lucky. Let me show the guys what went wrong here, and you can listen in too."

Pulling the rest of the mechanics to the back of the taxi, Sam got out a screwdriver and a flashlight. It only took a few minutes to remove the screws holding the new two-way radio box that had been installed the day before.

"Look here. When the guys went to install the radio yesterday, they drilled some pilot holes for the screws to hold the mounting brackets. Unfortunately, somebody let the drill go too far. They drilled right down through the top of the gas tank, which sits under the trunk. Gasoline fumes were leaking slowly up through that new hole and filling the trunk. Each time you pressed the Talk button, the relay contacts on the transmitter created a little spark. If the fuel-air mixture was rich enough, a significant explosion was the result."

"Blasting open the trunk was the smallest part," Sam continued. "You were lucky you didn't end up incinerated. Come to think of it, I heard about an unexplained taxi fire over in Boston. It was burned to a cinder. They thought it might be arson, but I have a feeling they might have had a similar problem."

Tony's face had now transitioned back from purple through its normal color and was headed toward a sickly greenish gray. "I coulda been toasted!" was all he could whisper.

Overcome common barriers to clear communication

1. **Start with an assumption that the person reporting the problem is intelligent and sincere.**
2. **Listen to what they are telling you without jumping to any conclusions.**
3. **Be very careful about jargon. Make sure you are both using the same words to mean the same things.**
4. **Just because you have never seen the problem does not mean that problem is not real.**

Chapter 5: Methods for Better Communication

Having discussed some barriers to clear communication in the previous chapter, let's talk about more methods that can help or hurt your ability to reach a shared understanding of a problem.

Whiteboard Abuse

Are you familiar with the television drama *House, M.D.*? This entertainment program revolves around the adventures and misadventures of a medical doctor, one Gregory House, M.D. He is addicted to painkillers, is abrasive, and is abusive, torturing his associates with juvenile and self-centered behavior. The concept of the program is that they all put up with his horrible behavior because they desperately need his brilliant diagnostic skills.

In other words, he is supposed to be a wonderful and brilliant debugger. In his world, the systems are the human body and various diseases.

It is kind of strange, but every medical worker I talk to about the show thinks that Dr. House is actually one of the *worst* diagnosticians they have ever seen. The actor playing House is fantastic; the cast surrounding him is young, attractive, and interesting; but it can be painful to watch the group stumble and fumble their way to solutions.

Ostensibly, this is because House is solving the really difficult cases. In fairness to the writers and the viewers, however, the clues are generally out in plain sight. This really means that House is just like the rest of us. He flails about, ignoring red flags and chasing hunches when he should be spending a lot more time on his problem descriptions and methods.

This brings me to my least favorite part of the show: Dr. House (at least in the early seasons) is too fond of using a dry-erase board (also known as a "whiteboard") to write down the key clues and possible diagnoses.

There is nothing wrong with using a whiteboard to quickly gather a list of facts or to scribble out simple block diagrams that show some relationship or sequential function. I have done this myself many, many times. But the correct use of such a brainstorming tool is to then collect the information and neaten it up a bit into some kind of electronic format. Type that list up in a document file, copy it to a spreadsheet, or at least take a photograph of it! Remember, we live in team environments, and we need to be able to communicate these ideas quickly to other folks who might be far away. A clear history of your thinking and a clear description of the problem will be critical later in the debug process.

Unfortunately, the fictional Dr. House is perfectly willing to work an entire case from his whiteboard, erasing and crossing off ideas when he has done a preliminary test that seems to eliminate some particular diagnosis. So my next objection is that he exhibits classic jumping-to-conclusions behavior with his brainstormed list of possible causes. A whiteboard is not such a great method for recording multiple tests, complex interactions, detailed step-by-step analyses, or statistical representations. In fairness, the team of doctors sometimes is shown looking at multi-page file folders, but the portrayal is that these are just stacks of "fact papers."

A slightly better example of whiteboard use is found in the "Lincoln Rhyme" mystery novels written by Jeffrey Deaver. Forensic scientist Rhyme's assistant meticulously records key facts on large whiteboards. In this case, additional documents and photographs or charts may be posted on the whiteboard. The novelist often summarizes the running contents of the whiteboard at the end of a chapter or section. In this way, he is playing fair with the reader, keeping all the clues in front of you, while still working magic in the words that eventually reveal the true nature of the crime.

Another TV show mystery named *Castle* extensively uses a whiteboard, but in this entertainment, the characters all seem able to keep long lists of additional facts in their heads. They point to simple relationships between a few photographs on the whiteboard and then recite complex deductive reasoning using many extra facts that leads them to final understanding of who-done-it. It is neat that they are all that smart and have such good memories, but I'm not sure I find such debug skill plausible.

Now you sigh and say, "Okay Mr. writer-guy, so what are better methods?"

Use Whiteboards (Yes, Whiteboards!)

Oddly, the first recommended method on my list is still to use whiteboards. But please, use them only in a very limited way. They are good for small, local groups and are great for brainstorming. This is where you throw out a bunch of ideas to build a list of questions, possible causes, or possible avenues to investigate. Or, you can scribble a picture or schematic diagram of how you think the system should be working or how you think it is actually working. But this presentation is only useful to the people in the room at that moment.

As soon as you have your list or your sketches, take a photograph of the whiteboard. If all else fails, you can always transmit a copy of your photograph to other participants. This is probably one of the weakest ways to communicate your thinking, but it is better than a big empty nothing.

Some whiteboards have built-in printing capabilities, but that feature has faded in popularity in recent years. There are also computer-based whiteboarding applications, which enable you to project a computer display onto a larger screen for group viewing. The drawback to this method is that you can usually have only one person "driving" the computer. This might require some training and skill, whereas almost anybody can "drive" a marker and a whiteboard. However, software whiteboards capture the image digitally and make it immediately available for distribution.

Convert Your Brainstorm to an Electronic Format

Quickly commit the information from your whiteboard exercise to an appropriate electronic format. This might be a word-processing file or a spreadsheet file. You might embed drawings, sketches, charts, or diagrams. Sometimes, you can simply use links to external documents.

The point here is that sooner or later, you are going to need to communicate the investigation to somebody who will be outside of the room that contained your whiteboard. Those people cannot see inside your head and they cannot see your whiteboard—and yes, whiteboards get erased (by intention or accident).

Writing allows us to transfer our ideas across space and time. Bigger teams, global development, and global customers push us toward modern methods of electronic communication. If you store your information this way early, you will have a great starting point to edit your notes into documents that are appropriate to your next audience. These documents might be transmitted by email, Web access, FTP sharing, or even FedEx envelopes. Your physical whiteboard does not fit any of these transmission methods very well.

Simplified Block Diagrams

One of the best tools I have encountered for dealing with complex systems—especially in debugging mixed hardware/software systems—is a simplified block diagram. Sometimes, you will use a block diagram to represent the entire system, with emphasis on the major blocks and the interfaces between those blocks. You need a list of the "goezintas" and "goezoutas." These silly words mean you need to know what "goes into" a block and what "goes out of" a block in that diagram.

Often, when creating and discussing these block diagrams, you will suddenly realize that there is an obvious hole in your system design—a completely missing interface or signal. You also might find that the interfaces are there, but you have not really done any good testing of one specific interface. When you go look, you suddenly find that the interface is not working in the way you expect.

Complex systems can incorporate feedback. This says that signals into one block depend on signals coming out of that same block or a later block. Closed loops like this can increase stability and performance of a system or can lead to wild, unstable oscillation of the entire system. Feedback systems and safety-limiting systems can also cause a system to simply shut down and stop normal operation. Because these controls are expected to help the system, it can be confusing and difficult to understand that they also might be keeping the system from working at all.

This book is not meant to cover control-system theory or stability. I must assume (and hope) that you have a good grasp of that subject when your system includes feedback. This can be a good area for further research and for advice from true experts.

You use the block diagram to complete two different tasks:

- To record and collect some important things you know
- To quickly highlight some things you don't know

A later chapter discusses things you don't know, but it should already be obvious that in the process of capturing the things you know, you will inevitably expose some things you don't know (and will eventually need to find out).

Bug-Tracking Software

Software folks have been dealing with a specific methodology for this kind of problem solving for a long time. They use a computer database to do bug tracking. A good bug-tracking database enables you to store lots of who/what/when/where/how information in one place. Most bug-tracking software allows a manager to assign a specific bug to various parties and record their progress. Sometimes, these databases are even visible to customers or partners.

Well-run organizations use bug-tracking databases to also follow hardware or system problems. This is ideal for following up on problems that are not immediately assignable to just hardware or just software.

Note that all of these methods work well, whether you are the person reporting a problem or the person solving the problem. Both of you will need to communicate clearly.

It is with great trepidation that I have included any mention of recent television shows. Nothing dates a work so quickly and badly as referencing some popular culture item. Please forgive me, I will attempt to avoid it elsewhere in this text.

Methods for Better Communication

1. **Use whiteboards (but in a limited way).**
2. **Convert your brainstorming to an electronic format.**
3. **Share photographs, video, or documents; any files you can easily store and transmit electronically.**
4. **Use simplified block diagrams.**
5. **Use Bug Tracking Software to organize your efforts.**

Chapter 6: What are the rules?

When you ask the question "What are the rules?" you might be asking if you know the rules and have successfully applied them in your design. If this is somebody else's design, then you need to understand the rules in order to know how the designer intended the product to work in the first place. Once in a while, you will find a product that *never* could have worked the way the vendor describes. We all hope that is a very rare occurrence.

Sometimes, rules might be very simple. Let's think about fixing a leaky roof:

- Water runs downhill (gravity).
- Water can wick up into a narrow cavity (capillary action).
- Water will flow along a surface when it can (surface tension).
- Water running across a surface often leaves behind sediments and residue.

Knowing these rules might help you to recognize where water is getting into an attic space and why the water might come out into an interior space that does not exactly match the location of a defect in the roof or maybe an overflowing gutter.

Using the fourth rule, you might find some residue or evidence of the path of the water once it is below the roof. If you can see where the water comes in, you have a much better chance of figuring out where to put some patch material or where you need to replace a shingle or board on the roof. The other rules give you ideas for where to look. Using the first three rules, if there are no available surfaces for the water to run along (or run downhill), then the water might be dripping directly to the spot you find it. You simply look straight up.

You might think this is too simple of an example, but it is a common enough problem. Many people have already tried (or will someday experience trying) to locate the source of a leak.

When debugging complex electronic circuit boards, I am always astonished at how many problems come back to the simplest possible rule of electricity: Ohm's Law. This relation states that the voltage drop across a circuit is equal to the current flow through the circuit multiplied by the resistance in the circuit. In formula form:

$E = I * R$

This is so simple, how can people get it wrong? Yet over and over again, I find printed circuit-board layouts that use too small of a conductor to carry a given current. I find systems wired to a remote power supply using too small of a wire diameter. I find high-current connections that depend on tiny mating forces to keep them in contact. The result of such mistakes will be too large of a voltage drop, meaning that less voltage (energy) is delivered to a key spot on the circuit board.

Sometimes, this problem shows up only with high-frequency circuits. In this case, the characteristics of the physical circuit change as the frequency goes up. The equivalent of resistance in alternating current circuits is called *reactance*. There are other simple equations that represent the effect of various circuit components (capacitors and inductors); you need to understand the effect of those components if you are going to see how the rules apply.

No matter how simple the rule, you will find that you and other designers make mistakes in applying these rules. Engineering is a human behavior, and we all make mistakes. Sometimes we simply forget to check our work or to check what the rule is telling us.

We forget to check whether the strength of the material used in a part matches the expected stress applied to that material by a normal human being. Then we forget to check whether some users might be on the outer edge, capable of super-human strength (football players, body-builders, teenagers, or angry spouses).

There are some good methods to keep you from forgetting the rules when you are doing designs: You can develop simple checklists and you can hold design reviews.

Simply caring about your work and about the needs of other people can make a big difference in the quality of an outgoing product. If you design a brilliant product but it is just too difficult for the folks on the factory floor to assemble, then your customer will never get to experience the smart and creative product you intended. Instead, they are likely to receive a pile of parts that don't really go together right. This is how they will think of your product and ultimately how they will see you as its designer. Nice idea, but it was a pile of junk. Likewise, if you design a car that attempts to pull kilowatts of electricity from a 1.5V AAA carbon-zinc dry cell, then you are breaking the rules and your product was a bad idea from the start.

You study the rules in school. You teach yourself the rules when you take on a new subject outside of formal training. The rules for your special skill might have been developed over tens or hundreds or thousands of years, or might be new and changing at the speed of the Internet. You have to determine the special requirements of your own special skill. Do you need to be constantly learning new rules? Or can you stick with some basics that are unlikely to change during your career? I cannot answer those questions for you, but I think everybody has a sense of the pace of change in their own field of interest.

Can you write down a short list of the rules for your skill? My high-school physics teacher taught the "blank sheet" method of studying. Take out a blank sheet of paper. Write down the things you should know at this point in time. Go back and compare them to the important things in your textbooks and notes. Did you miss anything? Repeat the process until the answer is no!

Okay, you are going to scream at me and say, "Hey, I can't write down a complete summary of 10 to 20 years of education and experience on a single sheet of paper!" True, but I'll bet that the best engineers you know can do a pretty decent job of it, without looking up a single reference. Maybe you don't remember the derating formula for a given brand of electrolytic capacitors, but you should have a pretty good idea of what terms it includes and where to find that formula in the literature.

And that is the big secret here: You don't always need to know every last detail, but you must have a pretty good idea of what the rules mean and generally how those details will fit together. You need an intuitive grasp of the rules: "When *this thing* gets bigger, then *this other thing* gets bigger too."

Constraints

There is a very special set of rules that you must apply to your problem. Unfortunately, these rules often have almost nothing to do with the problem itself. Even so you are commanded to treat them as if they were primary relationships between your known and unknown items. We call these special rules *constraints*.

A dictionary might define a constraint as a limitation or restriction. That definition works pretty well in this context. Your choices for solution are limited; your options for improvement are restricted by various constraints imposed on you. Some of these constraints might come directly from customers; some are imposed by management as part of doing business. Sometimes, people have constraints imposed by their own beliefs.

In any case, it is important that you list the constraints in your problem-solving documentation. If management says, "We must have a solution by the 15th of the month," or "We must have a solution by tomorrow," you cannot ignore that input. That fact (or belief) will limit the choices you can make in pursuing solutions. You cannot start a run of experiments that will take two weeks if the answer is needed by tomorrow.

Sometimes, the constraint is economic: "Your fix to this problem must not create more than 5% increase in direct component costs." Or maybe, "You can have only three engineers work on this problem." Alternatively, the constraint might be cosmetic, such as dictating the color, appearance, or surface finish of a product.

Constraints can also be completely arbitrary: “The product must be smaller than *X* by *Y* by *Z* millimeters.” Just because a constraint is arbitrary does not mean there is not a good reason for the constraint, however. For example, a mobile phone needs to fit in a pocket or be held by a normal human hand.

Occasionally, you have hidden constraints. There is nothing more damaging to a product’s development than finding out late in the project that nobody told you about some key critical requirement.

The Three Factors

There is an old joke (or maybe I should say truth) about project management. All projects involve three factors—time, money, and results—*but you can only control two*. (Some people use other words for the same concepts—for example, schedule, cost, and quality.)

Projects are like balloons. As you squeeze on one end, the balloon swells out in the other direction. Businesses involve the perpetual effort to control all three factors. No business can ever completely succeed at this effort, but businesses can try to minimize the bulging in one factor by releasing a little pressure in the other factors.

We can represent this graphically in a figure of a triangle with balls at the vertices. Each ball represents one of the three factors (time, money, results). You can think of them as balloons connected together by hollow tubes. As you squeeze one ball to shrink its size, the air is forced into the other ones, causing them to increase their size.

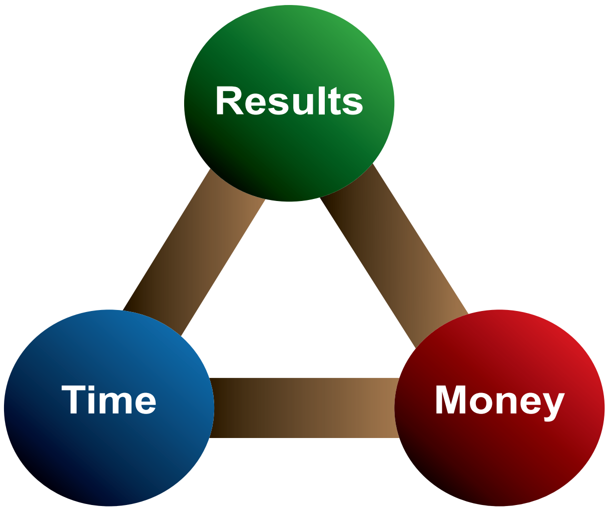


Figure 6-1: The Three Factors of an Project

Your boss has two of these balls firmly in his grasp and is squeezing as hard as he can. You know this because they are your balls he is squeezing.

A given constraint typically applies to only one of the three factors. It is often your job to minimize the damage to the other factors while applying the constraint to your project or product. For example, the boss might ask you to add a feature (improve the results) without adding too much cost or delaying the schedule. Alternatively, the boss might ask you to speed up the development, but still include all the features without adding cost. Or the boss might ask you to reduce the product cost, but still meet all the requirements and schedule.

After the first edition of this book was released, I was staring at this figure and pondering the constraints we all face in various designs. Suddenly a strange idea jumped into my thoughts and simply would not go away. There are ***four*** primary constraints in every project, not three. That makes the constraint triangle into a constraint pyramid—with triangular sides on each face. What is this new constraint, you ask? ***People***.

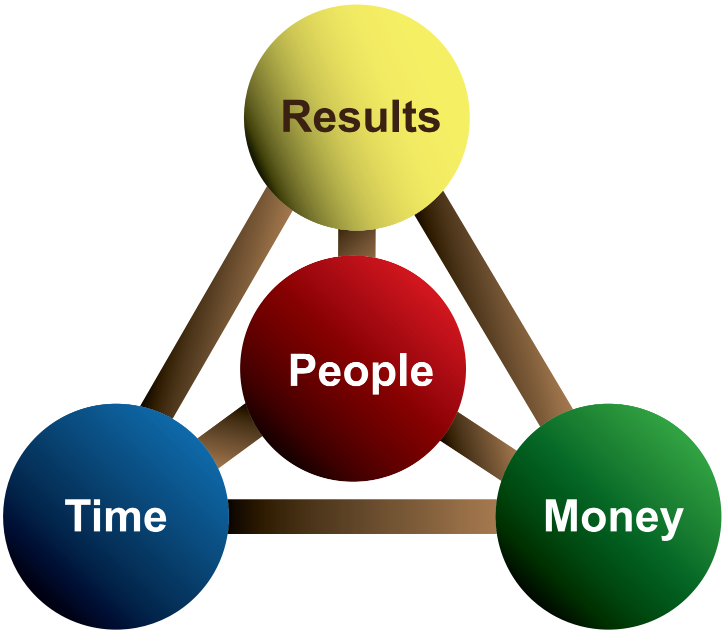


Figure 6-2: The *Four* Factors of an Project

I put People at the top of my constraint pyramid for a very specific reason. I believe that this is the most important constraint among the otherwise equal limits. Of course you can view the pyramid from any angle, so People might not be at the top of your pyramid—but I believe they should be there.

There might be some of you now saying, “Oh ‘people’ is just something you get by spending money, so you really don’t need a separate constraint there.” I cannot accept that, and let me share why I believe that *People* are an independent and overwhelmingly important factor.

Is there any amount of money that would have induced Steve Jobs to leave Apple and go to work for Microsoft? No, of course that would not have happened, even though there was a point in time when Steve Jobs had just been pushed out of Apple and Microsoft had plenty of money to offer. Steve Jobs working at Microsoft just was never going to happen.

Likewise, there are many, many people out there looking for work (or not looking for work) that you would never hire to work on your project. They might be perfectly suited for a given job, but they are not the right fit for your team.

This is why so many management books and articles make a big deal out of hiring and firing. You must have the right people for *your* situation.

Having spent a lot of time studying this analogy and diagram, I realized that there is one more fairly subtle note that makes the concept work. The balloon I call "results" should perhaps be called "bad results" and is really 1 divided by the value of Quality. Mathematically, it might be written:

$$Bad\ Results = \frac{1}{Quality}$$

This gives you some interesting consequences. Bad Results can grow to a huge value as Quality approaches zero. As Quality gets very, very big, the value of Bad Results approaches zero, but never quite gets there.

A wise Japanese manager once defined Quality as "anything that irritates the customer, other than price." I have often said that this definition is really saying the *absence of quality* is anything that irritates the customer, other than price, but today I think my reciprocal expression is really more appropriate.

$$Bad\ Results = \frac{1}{Quality} = Anything\ that\ irritates\ the\ customer, other\ than\ price.$$

Now our balloon and air-tube analogy really starts to work. If we put the squeeze on money, our (bad) results swell up and maybe our schedule (time) starts to grow. The same thing is true if we squeeze down on the Results (bad results) to get better quality in our project. Squeezing bad results to get better quality might require additional time, money, or more (or different) people.

You can run through lots of different scenarios like this, all with similar outcomes. If you squeeze on the People side, you often have fewer (or the wrong) people trying to accomplish the project. Schedules can expand and more bad results (lower quality) will be the consequence.

There is an underlying assumption that is exposed by this discussion. A given project with a defined set of goals will have a fixed "size" for its People-Time-Money-Results constraint system. It is certainly true that large organizations will sometimes have additional capacity available to throw at a project. This means they can endure rapid, short-term increases in People and Money without much impact. However, most customers have limited flexibility in their project goals (schedule, price, quality) and therefore the pressure is ultimately applied to *people* to cure problems in the other constraints.

Yep, you are getting squeezed.

Maybe you can meet the challenges handed you. When a real technical problem is overlaid by a web of constraints, a project is at its greatest risk of failure. Experienced engineers will tell you that the one thing most designs cannot withstand is the late arrival of new constraints.

You cannot ignore constraints. They must be added to the list of requirements for your product or project. They will be part of your response to "*what are the rules*" that apply to your problem.

Changing the Rules Changes the Result

⏩ Imagine asking various people a simple question: "What is 2 plus 2?"

A child will quickly and confidently reply, "4."

You ask an engineer, and he will say something like, "Well, assuming some small delta in the measurement value of our inputs, the answer will be approximately 4, plus or minus some variation based on the accuracy of those inputs and the tolerances in the addition engine."

You ask a Wall Street banker and he will say, "Are you buying or selling? Minus our commission, the answer is definitely a lot less than 4. Of course you could build that markup into the transaction."

You ask an accountant, and she will ask, "Is that on a cash or accrual basis? How soon do you have to pay the invoice? Is this a taxable transaction?"

Finally, you ask a lawyer. He will look left, then look right, then close the door and look at you through narrowed eyes and ask, "What do you *want* the answer to be?"

What are the rules?

1. **The basic science behind a system provides the rules.**
2. **Many problems come from violating the most basic rules.**
3. **Constraints are special rules imposed by the goals of each project.**
4. **All projects include the four basic constraints of People, Time, Money, and Results.**

Chapter 7: What Don't You Know?

"*...there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns -- the ones we don't know we don't know.*" Donald Rumsfeld, 2002 Press Briefing[1]

When you first start working on a particular problem or on something you have never seen before, you might find that almost everything falls into the general heading of "stuff you don't know." This is partly the reason so few people seem to be good troubleshooters. They quickly become overwhelmed with the stuff they don't know and despair of ever being able learn enough to fix this problem, let alone master a new subject. There often is a lot of pressure (from management, from customers, from family, from friends) to deliver a solution quickly. "I need it now!"

The thing you absolutely must do at this stage is make a short list of the relevant things you already know you don't know. Note the word "relevant." If you are investigating why a moving plastic part is breaking in all of your samples of a new product, you probably don't care if one of the things you don't know is how to bake chocolate-chip cookies. On the other hand, chocolate-chip cookies are delicious and can sometimes be used to convince other people to share their knowledge and wisdom with you. If you think you will need to bake some chocolate-chip cookies to get some key piece of information, then that can go on your list of things you don't know. Otherwise, it stays off of the list.

Brainstorming without Jumping to Conclusions

At this point, I need to talk some about the difference between brainstorming and jumping to conclusions.

Brainstorming is when you make some educated guesses about the kinds of things that *might* be causing the bad (undesired) behavior in the system you are trying to debug. *Jumping to a conclusion* is when you say in advance that *X* is the cause of

[1] http://www.defense.gov/transcripts/transcript.aspx?transcriptid=2636

problem *Y*. By "in advance," I mean you are making such a statement without reliable, repeatable, and reasonable proof of this cause-and-effect statement. In other words, jumping to a conclusion is when you suddenly and arbitrarily decide that one of your brainstorming ideas *must* be the correct answer.

In a brainstorming exercise, you can write down a list of things that you think might be the cause or might at least be closely related to the cause of the problem.

Jumping to a conclusion is when you pick a single possible source of the problem and fixate on that while excluding investigation of other things you don't know.

So if you are trying to fix a basic gasoline internal combustion engine, you might brainstorm that the problem *could* be in one of these areas:

- Fuel flow
- Air flow
- No spark
- Clogged exhaust

Each of these general areas will lead you to a list of things you do not know (yet) about the problem. For example, is fuel getting into the combustion chamber? Is the air filter blocked? Is too much air leaking in from unexpected places? Is the electric source for the spark functioning? Is the spark plug shorted to ground? Is the spark plug worn out? Is the spark-plug wire broken? Is the exhaust system clogged? Are the intake or exhaust valves stuck open or stuck closed or damaged in some way?

Jumping to a conclusion would be if you suddenly said, "It has to be the fuel flow. Forget all of that other stuff; just look at the fuel flow." You could just as easily have become fixated on the spark system. "It must be the spark plugs. Let's only work on the spark plugs." It sounds pretty dumb when you write it that way, but we all fall victim to jumping to a conclusion sooner or later. Unfortunately it is usually sooner.

Don't worry if you don't really know much about internal-combustion engines or the terms I have used here. This is just an example. I could make up a huge list of other examples from nearly any field. But I like this example because most people have at least some exposure to cars and (like me) a vague understanding of how engines work.

The best part of this example is that it shows how a list of general functionality (things you know) can lead to a bunch of questions (things you don't know) that are actually pretty easy to find out. You can check the cylinders for unburned fuel (wetness). You can simply look at the air filter or the valves or the spark plug to see if they meet your understanding of how they should appear. Any unusual appearance might give you a clue for further investigation or might tell you enough to lead directly to a fix.

I really just want to convey the concept of brainstorming a list of things you don't know and then turning that into an action list. The investigation of each item in that action list should eventually lead you to enough information to actually solve the problem. However, this part of the process might not get you to success yet. You might not be good enough at listing all of the aspects of the problem to see that there is something else you don't know.

So here is a question: Do you need to find out or solve every single thing in your "things I don't know" list? My answer is absolutely, definitely, maybe yes, maybe no. The goal of your problem-solving exercise will vary with the problem itself.

Let's say the problem you are trying to solve is extremely urgent. For example, the airplane you are piloting is diving toward the ground and the propeller has stopped spinning. Perhaps the highest priority of the debug process will be to achieve some kind of semi-level glide path and attempt to survive the intersection of plane and earth.

There will (or won't) be plenty of time later to tear down the engines, fuel-delivery path, and control systems.

If you don't fix that first issue (the dive), the remaining root causes and solutions will be somewhat unimportant to you and your passengers.

On the other hand, if you are designing a high-volume consumer-electronic product, you might ask yourself, "Do I really want to be shipping a product with circuit sections that nobody ever got around to testing and debugging?" In other words, a well-designed product should not have any unresolved "things you don't know."

What Don't You Know?

- **Brainstorm to create lists of things you don't know (yet).**
- **Prioritize these items to make efficient use of your time.**
- **Do not fixate on any one item. This is jumping-to-a-conclusion. Such behavior is immensely damaging to a good debug effort.**
- **Include this list in your reports. As you discover the answers to things you don't know, you can move them into your list of things you *do* know.**
- **Don't be afraid to add new items to the *Things You Know* and *Things You Don't Know* lists as you gain knowledge about the problem.**

Chapter 8: Finding the stuff you don't know

Let's do a quick brainstorming list of some of the ways you can find out the things you don't know about some problem that you are trying to solve:

- Derive some knowledge from first principles.
- Ask an expert.
- Look it up on the Web.
- Look it up in a book or magazine.
- Use your basic senses to: see, hear, smell, touch, and taste.
- Conduct some experiments or tests to prove an idea.
- Collect some data from tests or experiments.

That's a pretty good list. You have probably done some problem-solving using all of these methods, except perhaps the first one in the list. Keep in mind there might still be other great methods that I have not listed. I love hearing stories about other ways that people have found to uncover the "stuff they do not know (yet)." Let's work our way through this list.

Derive Some Knowledge from First Principles

Although we all wish we were Einstein-level smart, there are not so many of us who can solve a problem simply by retreating to a quiet room and applying massive brain power and mathematics. That is not to say that you should not try, nor does it mean that you should ignore the idea of working your way up from basic principles toward an understanding of a more complex problem. Remember that those basic principles should already be in your list of things you know.

Ask an Expert

This is often the most expedient solution, especially when you are less experienced and work in a larger team environment. You simply find somebody with more experience or knowledge than you and ask him for an opinion. I have no doubt that every single person can think of a situation in which they have solved a problem (big or small) very quickly just by asking the right person a few questions about the topic. (Of course, instead of a quick answer, you might get told a story, like "The Dog Barks When the Phone Rings." The story might help or it might just leave you scratching your head.)

The expert could give you a few comments and then say, "Let me think about that for a while." Sometimes, that means he sincerely wants a little space to focus and think about your problem. Other times, saying this is just an excuse to make you go away and leave him alone. (Or maybe it is a little of both.)

Sometimes, the expert will admit that he does not have the right answer at his fingertips, but will give you several pointers to resources that he would check first. Indeed, the answer might be nothing more than hints or weak suggestions. The expert might also suggest a list of questions or experiments that you should try. This behavior is very much an exercise in brainstorming. They are providing educated guesses, but leaving the hard part (investigation) to you.

Larger companies are filled with internal experts. Smaller companies sometimes have to reach outside to get expert help. Obvious candidates for experts are people whose knowledge and wisdom you respect: former coworkers or bosses, former classmates, experts at your current component vendors, and finally, outside consultants.

There are some cautions to be applied here. If you ask multiple experts, you might find yourself in the position of needing to ignore the advice of one or more. This can lead to awkward or difficult conversations if one of your experts follows up later and asks you if his advice solved the problem. The safe answer should always be, "Your suggestions were extremely helpful! Thank you!" If he tries to pin you down on specifics, you can say that while his comments helped send you in the right direction, the solution ultimately turned out to be something else.

A really good expert is not asking you for the outcome just because he wants to stroke his ego. ("I was right!") The best experts relentlessly pursue new knowledge and especially new problems and solutions. They see this as preparation for their next problem. The good experts will come back and ask you what you found. Of course, they hope that their advice helped, but mostly they just seem to get a good feeling when problems get solved.

If all else fails, you can spend some hard cash to hire an outside consultant.

A consultant is someone who borrows your watch to tell you the time. (And then sends you an invoice.)

There is a lot of truth in that saying, but you should not be afraid to reach out for help when it is absolutely necessary. Be sure to carefully define what you expect to receive from an outside consultant, when you expect to receive it, what it will cost you per hour, and the maximum amount you are willing to spend on this consultation.

I have another joke that applies here too: An expert is anybody more than 25 miles from home.

Even though asking an expert is fast and easy, it has one big drawback if that expert is in the same field as you: It means you are not exercising the mental muscles that you need to develop to become an expert problem solver yourself.

Please use the asking-the-expert approach with care. If you have no other choice due to time or other constraints, challenge yourself by writing in advance a private list of the things you think the expert might tell you. When you find that you are consistently predicting experts' answers, you probably don't need to use their help as your primary starting point.

Now I am going to immediately contradict myself—sort of. If the expert is in a different field from you, it is always a good idea to consult and listen closely to an expert's advice. Nobody can cover all subjects. As much time as you have spent learning your topics, other experts have spent learning their topics. Spend some of your time figuring out whose advice you trust in other areas. Use their advice, but don't abuse their time and generosity.

Look It Up on the Web

Of all the sources of information, this one can be the most hazardous. There are times that I am convinced that the Web has become the "source of all knowledge" for too many people. I am also convinced that at least 80% of the raw information commonly available on the Internet is absolutely wrong. When you are collecting information from the Web, be sure to always keep track of the source of information and your general confidence in that source.

I generally use the Web for information gathering in an extremely cautious mode. Vendor datasheets and application notes are typically pretty good and usually reliable when published on the vendor's own pages. Some secondary reference sites claim to offer extensive libraries of datasheets from other vendors. I am always quite cautious about such secondary sites. Authorized distributors typically maintain reasonably good datasheet libraries, but it is best to gather your information from primary sources.

My BS detector goes off the scale when reading user forums and enthusiast Web sites. Anyone, including the bottom 80% on an intelligence scale, can contribute to public forums. Of the remaining 20%, on any given day, some will make silly mistakes and others will be mentally absent or just in a bad mood. That translates into about 90+% of the information being wrong, bad, poor, or sadly misguided. Can you find useful information on the Internet? Yes, but it takes a very careful eye and extra effort confirming any information from sources like this.

Some engineers use Wikipedia as a primary source. I generally find it a good place to get a very rough overview of less-familiar topics, but then I must do much more confirmation research to be sure that I have not been completely misled by some amateur posing as an authority.

The World Wide Web brings us information at nearly the speed of light. It can make us appear brilliant and knowledgeable incredibly quickly. Unfortunately, the Web can make us stupid at exactly the same speed.

Look It Up in a Book or Magazine

The Web is an easy place to plant misinformation by ignorance, malice, or accident. Publishing "bad" books or magazines represents much more wasted effort and cost, especially if that information is subsequently disproven or becomes generally accepted as unreliable. For this reason, your local university, public, or corporate library is often one of the best places to find answers to what you don't know.

The best problem solvers with whom I have worked all made an effort to maintain a high-quality personal library of books and magazine articles. Often, these are in neatly organized file systems and bookshelves. These problem solvers will religiously skim through trade magazines and then copy or download electronic copies of key articles they find interesting. Therein we find another clue about good problem solvers: They are consistently curious about a wide range of topics and relentlessly collect and sort knowledge that meets some arbitrary standard of "good to know."

Use Your Basic Senses

The next method you can use to find out the things you don't know about a problem is to simply use your normal powers of observation. In other words, you can look at the problem. Arthur

Conan Doyle's character Sherlock Holmes solved most of his mysteries by using keen observation and encyclopedic knowledge and then by applying extraordinary deductive reasoning. Holmes is astonishing because most people are incapable of matching his skills in any of these areas, let alone all three.

Even without emulating the ultimate detective, you can apply some of the same techniques. Ask yourself these questions:

- What did you **see** just before, during, and just after the failure?
- What did you **hear** just before, during, and just after the failure?
- What did you **smell** just before, during, and just after the failure?
- What did you **taste** just before, during, and just after the failure?
- What did you **feel** just before, during, and just after the failure?

I will discuss each of these, but it is important to note that some or all of these might not apply to your problem. On the other hand, you might be surprised at how often several of these eventually *are* found to be significant to your problem. Keep an open mind and open your senses—but don't jump to any quick conclusions without lots of evidence.

No disrespect is intended here for people who have a specific limitation (for example, vision or hearing impairment). In such cases, I suggest you engage in teamwork with someone who could supplement your observations in those specific areas to be sure that no observations are missed as you collect further information.

What Did You See?

I don't know if there is a whole lot of explanation needed here. Obviously, you probably don't need to record the presence of every item in your field of vision. You need to pay attention to the relevant things you see but ignore those things that clearly do not have anything to do with your problem. If there is always a bright flash of light a few milliseconds before a failure, then I believe it would be worth noting that fact. However, if the system is constantly flashing bright lights and they only occasionally coincide with the failure, then at best you might note the apparent lack of coincidence between the two.

What Did You Hear?

Surprisingly, many people do not find this observation as easy or intuitive as the previous one—what they saw. Sometimes this is because they were too busy talking during the event. Sometimes they focus on watching the system and maybe forget to listen as well. Perhaps a lack of training plays an important role.

Whatever the reason, you will want to avoid this pitfall and start listening to your systems. It is astonishing how much listening can reveal about the workings or failure of a mechanical or electromechanical system. My wife claims that I have several times accurately predicted the failure of our furnace and air-conditioning system when I have asked her, "Hey, does that sound different to you?"

⏩ *Once upon a time,* an employee we will call Brad, at a company I will cleverly call "The Vendor," received a phone call from a customer. This customer was having a problem with an industrial control computer that was failing to boot up properly from a floppy disk. The Vendor had provided some of the circuit boards, but had not built the entire system.

After collecting some basic information, Brad finally asked the customer to describe the sound the drive made during boot. The customer struggled to describe the sounds accurately, so Brad suggested that he hold the telephone handset up close to the floppy disk drive and then power up the system.

This is what Brad heard: ZZZZzzzT! zzzZZZZT!
Then, <medium pause> THUNK! <medium pause> THUNK! <medium pause> THUNK!

Brad knew what the floppy driver software normally did during boot. He had previously written some of that driver code. Those noises told him a very complete story.

The first falling note was the driver causing the heads to seek back to track 0. During boot, this was an automatic behavior to ensure that the firmware knew the current location of the disk drive read/write heads. Then, after some setup and housekeeping, the drive would be told to seek out to the last four cylinders of the disk, where the operating-system code was stored. This was represented by the rising tone of rapid steps. Both sides of the disk were then read for each of the last four cylinders. The three THUNK sounds represented the heads stepping from one cylinder (track) to the next.

If the floppy-disk controller integrated circuit (IC) detected errors, the driver had a pattern of relocating the correct track that ran something like this:

1. Step to the track and read sector(s).
2. If there are location or data errors, step in and then step out by one track. Try reading again.
3. If location or data errors, step out and then step in by one track. Try reading again.
4. If there are still errors, seek to track zero to establish a known head position.
5. Repeat step 1, seeking and reading, two more times.
6. If there are still errors, report them.

This error-recovery pattern made a very distinctive clicking sound as the heads moved in and out over the desired data track. But Brad was not hearing this repetitive clicking sound of retries.

The sounds were telling Brad that the floppy-disk controller was doing everything normally. No errors were reported on the console terminal. Yet the operating system was not booting up. In this case, the answer was surprisingly straightforward. Because the floppy-disk controller was not detecting errors, Brad had good evidence that the diskette was okay and that the controller was successfully reading the boot image.

For some reason, however, the operating system boot image was not showing up in the memory. Brad speculated that the memory might be in write-protect mode. Write protect was one of the few reasons that memory write cycles might fail. The memory could be write protected if the main +5 V rail on the system backplane were set too low. The customer adjusted the power-supply voltage to a proper value and the problem disappeared.

The critical clue in solving this problem was provided by those noises made by the floppy-disk drive. They were the normal noises, which eliminated almost all of the rest of the system as candidates for the failure.

What Did You Smell?

Occasionally (though not often), a system failure is associated with a smell. Most typically, this might be a "hot" smell or even a burning odor. Such smells might represent frictional heating in a mechanical system or a circuit overload in an electrical system. In extreme cases, these smells can be overwhelming, but the point of failure will usually become obvious due to associated smoke, melting, color change, or some other sensory input (e.g., the system feels hot).

What Did You Taste?

I have to admit that taste is an unusual sense to include in this list. Most of the time, we don't try to taste mechanical or electrical systems or devices to see if they show some kind of change.

My wife Debbie once walked into a service shop after we had major repairs on the transmission of our car. While she was paying the bill, the mechanic stopped by and explained the various repairs that had been required. As he was finishing his explanation, he commented that he knew we had one specific problem after tasting the transmission fluid. She stared at him, open-mouthed, trying to find words to express her astonishment.

He quickly realized what had caught her notice and said, "Hey, I don't *drink* it. I just put a little drop on my tongue and then spit it out. It is amazing how much you can tell by the taste." I guess I am glad that I am an electrical guy. We generally try to keep the gear out of our mouths. But I did actually trust that auto repairman, and he took care of many vehicles for us until he finally retired.

He had a good point, too. Our tongues and noses are amazingly sensitive chemical detectors. We should not discount or fail to document the evidence they provide. However, I insist that you first consider your personal safety in any experiments or observations. You don't need to lick an electric fence to find out if it is turned on.

How Did the System Feel to Your Touch?

Was it hotter than normal, colder than normal, or just normal? Do all of the failing samples "feel" different in some way? Some surfaces might be rough or smooth or textured. Feeling unusual temperatures or a wet or greasy area might also be clues.

In his book *Troubleshooting Analog Circuits*, Robert A. Pease attributed some brilliant advice to Tom Milligan, his former coworker: "When you are taking data, if you see something funny, record amount of funny."[2] Do not forget this important advice!

[2]*Troubleshooting Analog Circuits*, Robert A. Pease, page 5, ©1991 by Butterworth-Heinemann. Reprinted by permission of Elsevier Limited.

Encountering the Unknown Unknowns

Remember that you are collecting clues, and any odd or unexpected behavior of a system is most definitely a clue. You might not understand the clue immediately, and maybe you will never perfectly understand it, but put it in your notes.

This is such an important concept, I want to restate it in another way. Many times, when you are doing experiments (observations, measurements, tests) while trying to answer something in your “things you don’t know” list, you will find that there is something else going on. There is something you previously did not know that you did not know! For that reason, it did not appear in your list of things you know, your list of rules, or your list of things you don’t know. If you are not absolutely meticulous about recording your test results, this critical information will be lost.

Experiments to Prove an Idea

You sometimes set up an experiment to prove an idea that you have already formed about the behavior of a system. You do a test run, collect some information, and then make one specific change to the system. If you understand the system very well, you are able to predict the outcome of the experiment.

Sometimes you will be surprised by the outcome; sometimes you will confirm your previous belief. Both results are equally valid. In either case, you must write down what you did and what you found.

Experiments to Collect Information

Sometimes you will run some tests simply to collect information about the behavior of the system. In such cases, you are not trying to prove or disprove a theory. You are just taking a look and trying to see if something surprising, unexpected, or strange jumps out of the data.

Is There a Difference Between a Test and An Experiment?

I sometimes fall into the habit of interchanging the words test and experiment in casual communication. A more careful speaker would be sure to separate these concepts. My best attempt at differentiating the two would be this:

A *test* is an action you take where you *think* you know what result will be observed.

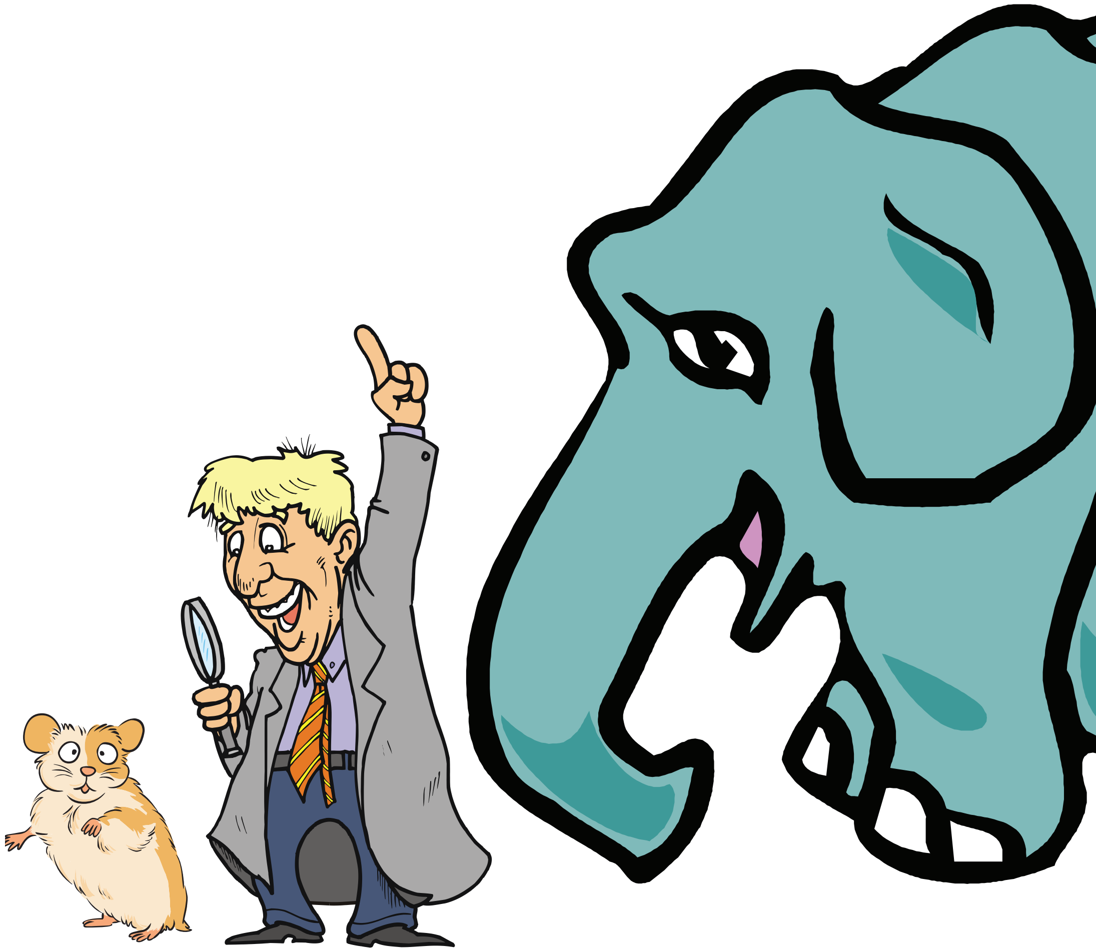
An *experiment* is a test where you are not really sure what result will be observed.

A surprising or unexpected result for a test or an experiment can be extremely helpful in understanding the behavior of the system and thus help you solve a problem with the system. Likewise, test results within your nominal expectation can reassure you that some parts of your system are working as expected. Your attention can then go to other areas.

In the next chapter we will journey more deeply into the subject of tests and experiments.

How Do you Find out the Stuff you don't Know?

1. Derive knowledge from First Principles
2. Ask an Expert
3. Look it up on the web (DANGER!)
4. Look it up in a book or magazine
5. Observe the problem with your own senses
    a. See it
    b. Hear it
    c. Smell it
    d. Touch it
    e. Taste it
6. Do some experiments or tests to prove an idea
7. Do some experiments to collect information



It is easy to miss something obvious.

Chapter 9: Experiments

There are several reasons we do experiments: to prove an idea, to find a specific unknown value, or to simply collect more information. I am always surprised at how easy it is to confuse these reasons for experiments and suddenly branch off in the wrong direction.

Do Experiments or Tests to Prove an Idea

In this first case, you conduct one or more experiments or tests to specifically prove an idea or to fill in one specific unknown value. You will probably change one parameter during the experiment to see the effect on the system. This kind of testing is often extremely focused: You have a clue or a hunch that you are following. In some cases, you could consider each observation to be a separate experiment. In other words, you make the system do something and then you look (measure, monitor, or capture) at point A. You then repeat the experiment looking at points B, C, D, and so on. You might call these "debug experiments" or "troubleshooting experiments."

When doing *any* sequence of experiments, be absolutely sure that you change only one thing at a time. It is incredibly tempting to throw a bunch of changes at a problem. You might be getting tired. You might be under a lot of pressure from management. But do not make this mistake! (Sadly, we all sometimes get impatient and fail to follow this advice.)

If you modify more than one thing between experiments, you do not know which modification caused any change in the results! Indeed, you *cannot* know this. Yes, it is probably one of the things you changed, but which one?

Of all of the mistakes of problem solvers, I think changing more than one thing between experiments might be the second most common. The first most common mistake in any debug is poor record keeping, which also creates poor communication.

The first and most basic kind of debug experiment is one you do simply to reproduce the problem report. Even if you are the source of the problem report, you might find yourself repeating some experiments (tests) just to be sure that the problem is repeatable and consistent with the setup you have documented.

Determine a Specific Value from Tests or Experiments

The next reason you might do a test is when you are trying to establish one specific value from your system. Maybe you want to know the temperature of one component, or the voltage or current at one node within a circuit. Perhaps you want to know some physical characteristic of a material.

Collecting Data from Tests or Experiments

The other reason people do experiments is simply to collect more information. In this case, you probably don't know where the data is going to take you. You don't know if this data will prove anything specific. For some problems, the data might not even be useful when you are finished. But you often need to perform data-taking exercises like this simply to have enough statistical information to draw some conclusion(s) about the behavior of the system or problem.

You might take extra measurements or observations during this kind of experiment. Sometimes, having a wide variety of measurements before, during, and after a problem can give you enough clues to see what is going wrong. Some problems involve design tolerances. Until you have enough samples, you cannot tell that the problem exists or how often the problem occurs.

> Not having a better name, I could call this type of exploration "statistical experiments."

Let's look at an example of a problem that required many experiments and lots of data collection. Eventually, all the indications led to a single experiment, which failed to fix the problem. But the *failure* of that experiment told the investigator he was looking at the right idea. We will call our investigator “Barry.”

⏩ *Once upon a time,* there was a long and painful investigation of some drop-test failures of a sophisticated communications device. The product manufacturer had internal standards for the height from which a packaged or unpackaged receiver should be able to withstand a direct drop onto a hard surface without damage that might cause the device to stop functioning.

Drop tests work something like this: A large and heavy steel platform is mounted such that it can be lifted to an arbitrary height. A trip release then allows the platform to be pulled straight down by gravity toward a solid concrete floor. These drops are generally done at heights of 1 to 2 meters.

These tests are not for the faint at heart. The impact produces a resounding BANG that can usually be heard and felt in every corner of the test building. A typical test cycle might involve dropping each sample one or more times on each face of the product (six faces for a rectangular box). The failure criteria might be that no more than zero or one product is allowed to fail from a batch of 100 samples. Doing even the minimum 600 test drops (6 faces × 1 drop × 100 samples) for one test cycle takes a lot of time. Some test setups allow dropping multiple products at one time, but some do not.

Drop tests are not silly or capricious. Package handling in transportation hubs often incorporates conveyor systems with drop heights that can equal such test values. Let me say that again in a different way: Shipping your product with delivery company X can guarantee that your product will be dropped at least once from a significant height. That drop is built into the company’s conveyor system. The farther you ship a product, the higher the chance of hitting multiple or many different drops.

A few good bumps in the back of a truck can approximate similar shock impacts. Human handlers sometimes toss packages over great distances and heights, not worrying about the potential damage or liability they are creating. Along a similar vein, have you ever dropped your cell phone? A well-designed product can withstand these stresses.

You can easily see that such a product test can be completed only after you have manufactured significant quantities of the product. Preliminary testing can be done on smaller quantities of product, but that does not guarantee success for the final product. Unfortunately, testing that happens late in the development occurs when there is the least amount of time available for change or recovery from a failure.

Barry's involvement in this project had been mostly advisory, with the electrical engineers reporting to him making minor changes to adapt this model to the needs of a new customer and then testing against any new requirements. Suddenly, the project meetings became dominated by increasingly long discussions between the mechanical and process engineers of some drop-test failures in the factory. Production start was imminent, but the failure rates were slightly beyond the allowable limit. International conference calls and continued debate began to fill up the team's calendars.

It did not help that the design had been started in one design center and then finished in a second design center halfway around the world. Surprisingly, though, there was no finger-pointing—nobody saying, "Hey, this is *your* fault." There were several basic ideas floating around, yet no measurable progress was being made toward a solution.

Barry casually suggested to a senior manager that the program manager was not getting much traction on this problem because he continued to listen to multiple conflicting inputs. The program manager was frozen because he could not decide which interpretation of the problem was correct and therefore which possible solution should be pursued. "At some point, you just need to assign a drop-test czar," Barry said. "You need somebody to sort out the options and find a good debug path to chase."

"Good idea, Barry," the manager replied. "You are it."

"Wait, what?" Barry protested that he was absolutely the wrong guy to take this on because he knew less than nothing about mechanical design or the manufacturing processes. Barry was the electrical guy. He had no dog in this fight. A short flurry of emails later, Barry unhappily found himself officially designated as the drop-test czar.

During an extended conference call, the team collected a short list of possible causes. These were published back to the team and several specific experiments were to be conducted. By the next meeting, most of the work had been completed, but the team still did not have absolute evidence of a specific cause.

The first surprise for Barry was how responsibly and professionally every person on the team behaved. He initially had feared that a finger-pointing war might erupt, with each skill area blaming a different group for the problem. The opposite was the case. Each group seemed to feel the best explanation for the failure was attributable to their own skill.

The mechanical engineers were focused strongly on the amount of support for the printed circuit board (PCB) and how much the PCB would flex at the moment of impact. The PCB expert was worried about the fabrication of the PCB material. The manufacturing-process expert was concerned that the solder process was to blame. Nobody was even remotely interested in blaming a different group for the problem.

Many significant experiments were run. Materials and strengths were tested and verified. Failed units were X-rayed and disassembled. High-speed videography captured the flexing of the PCB at the instant of impact.

The net result was that everything seemed to be within normal parameters. Yes, the mechanical design and PCB could have been a little more rigid, but they were well within the expected variation.

The system-chip vendor was convinced that the device should withstand the expected stresses at the solder balls, even though this was a smaller ball size than in the previous device. Metallurgical testing confirmed the PCB and integrated circuit had the expected materials and no sign of oxidation. The manufacturing location was sure that the solder process was optimized. Nonetheless, a small but statistically significant number of samples had failed. The failure analysis (not surprisingly) showed that the connections that were failing were at the locations where they would expect the most stress on the system-chip connections (solder balls) during a drop.

All of this information was again collected and reported back to the entire investigating team. And again, no strong consensus was reached. Barry was not showing much success as a drop-test czar. Like the program manager, he had many inputs, but no conclusive proof.

One engineer shared an article written about a different design in which the team had ended up gluing the system chip to the main board after experiencing a somewhat similar shock/vibration failure. Barry was intrigued by this solution. A small number of samples with glue were tested to extreme conditions with no observed failures.

There were many barriers to such a solution, however. For one thing, the circuit board became irreparable after the glue was applied. This was not very appealing, because it was not unknown to have system IC failures after the first solder cycle or later, in the field. The economics of simply discarding all of those assemblies were not good. Also, the manufacturing line would quickly bog down waiting for the glue to set properly. Fast-cure or thermally cured glues were an option, but they introduced new manufacturing steps that were extremely difficult to control in production. The glue itself added cost, and new production machines would be needed to apply the glue. Schedule delays and added cost are never welcome near the end of a product's development.

With the production start date looming close on the calendar, Barry traveled to the factory with some mechanical and industrial engineers. The team reviewed everything they could find in the factory and had one more conference call with their process expert in the United States. The remote process expert carefully discussed the solder profiles and test charts. He then explained for Barry's benefit some of the behavior that these solder processes can exhibit as the factory tuned temperature and duration parameters during the time the assembly spent in the reflow soldering oven.

There were at least two bad things that could happen to the solder balls as the factory pushed or pulled the process in one direction or the other. In one case, the solder could fail to make a good connection if the heating were not complete enough. In the other direction, too much heat could create an undesirable crystallization or granularity in the material.

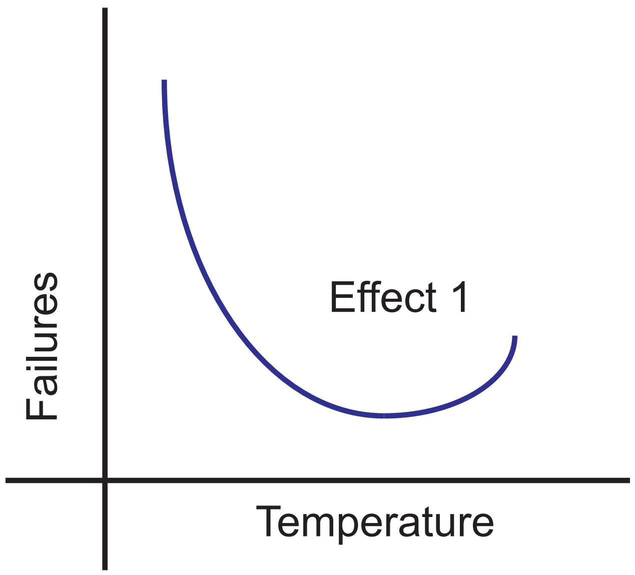


Figure 15-1: One Effect of Increasing Temperature

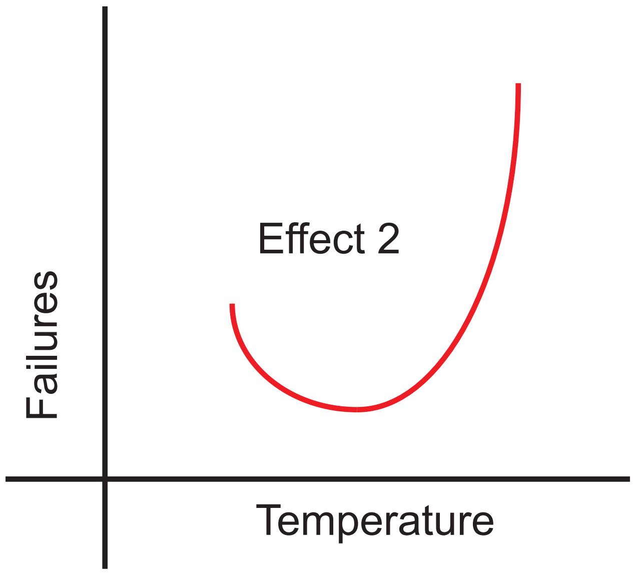


Figure 15-2: A Different Effect of Increasing Temperature

There was no easy way to tell if the failure was happening due to one mode or the other. The two effects might have looked like a big letter U on a graph if you were able to chart failures from each effect on the vertical axis versus process parameter across the horizontal axis. It is best for the process to sit right at the intersection of the two effects, where the production should have minimum failures, but there was no easy way to know where they were sitting on that curve when they started.

Metallurgical and microscopic examination were ambiguous. The team was sure that certain solder joints were fracturing during the drop tests, but the root cause was still unclear.

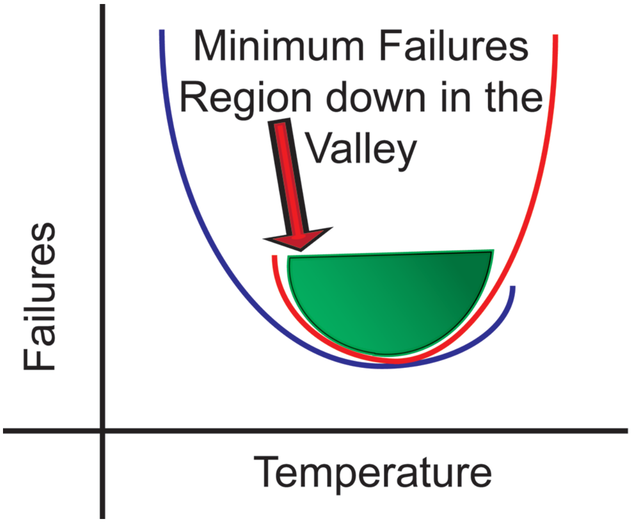


Figure 15-3: Combined Effects of Increasing Temperature

There was something the team could try. They could build a statistically significant quantity of boards with the process pushed a little toward one side. If the drop-test results showed significant improvement, Barry would know that this was the direction he needed to push the process. If, on the other hand, the results became worse, he would know that they needed to push the process the other direction to get farther down in the failure curve.

There was only one catch: If he picked wrong the first time, those samples would have even poorer solder strength. A larger number of samples would be destroyed by the drop test and the entire batch would need to be discarded.

There were some tense moments with the team considering the economic risks. Once again, no clear consensus was reached.

This became the second surprise in the investigation: It suddenly was clear to Barry why it was so beneficial for somebody—anybody—to be the czar: The czar could make a decision that had an economic impact, which would have been very difficult for a person with normal direct-line budget responsibility.

Barry made a completely arbitrary decision. He assumed that the process was sitting to *this* side, so they would push it *that* direction. If they were wrong, the failure rates would go up. If they had guessed correctly, the failure rates should go down and they would be done. The cost would not be small, but the team needed to know where they were and ultimately they needed a solution.

Tense hours passed as the production line was run through to final assembly. The next morning, the factory team started drop testing these new samples. Bad news traveled fast: The failure rate had doubled. But this was actually *good* news, because it meant that Barry's experiment had told them something useful. He was just unlucky at guessing the right direction to push.

With some further conversation with the process expert, the factory pushed the process parameters the other direction and built another large batch. This time, the results were not only better than the previous batch, they were now well within the drop-test requirements. Barry finally had a solution.

Barry shared all of the thinking and results with the factory team. They agreed that he was not pushing the process outside of the allowable envelope, and they had some good ideas as to why the process parameters might be offset a little bit from where they expected. They did raise some concerns about the soldering performance of an unrelated part, but quickly agreed with Barry's process expert that this specific new issue could be addressed by a small change in the solder-paste stencil. It was far more important to get the critical large system chip soldered correctly.

Long after this investigation concluded, it occurred to Barry that he actually had not collected enough evidence based on the first failed test to be absolutely sure that the team had completely understood the problem. What if their curve had been relatively flat in the region of the process setting? And what if they had not moved very far along the X axis? They might have seen no improvement!

Worse yet, if they had jumped completely across the failure valley, they could have landed very far up the opposite failure mode curve, which equally would have explained the sudden increase in failures.

It became clear to Barry that he was relying on the experience and wisdom of his process expert and factory team. Both had a good sense of how far they could push the soldering process and see only a small change. They had the confidence in their knowledge of the difference between a small change, which would reveal the direction of the curve, and a bigger change, which might have leapt over the valley onto the second effect. It was important for Barry to trust the experts and equally important to finally settle on the expert whose advice he trusted most.

This story includes both types of debugging experiments. In the beginning, the team tried lots of different tests and used a wide variety of brainstormed guesses to get clues about what failure modes to investigate. Later, they did a carefully controlled experiment in which a single (and small) change was made to verify their understanding. The change was applied to a statistically significant number of samples. Most importantly, the failure of that experiment did not automatically mean that Barry was on the wrong track. It did mean that he had made one wrong guess about where they were starting, but that failed result gave the team a good idea to then push a little in the opposite direction.

Finally, this story demonstrates the importance of thinking through an experiment and anticipating the results before you get them. If you can predict the possible outcomes and assign meaning to each possibility, then you probably understand the problem well enough to begin solving.

Are Experiments Easy? No.

The information presented in this chapter represents the most challenging part of problem solving, yet it is the part that most engineers believe they already know how to do. Let's hope that is they are correct. Either way, everybody can work to improve their skills.

Note that I am again using the terms tests and experiments pretty much interchangeably here. I previously suggested that a "test" is an experiment where you believe you know the outcome in advance. I am not sure it makes much difference. In either case, you might undertake a test (or an experiment) to try to disprove some idea. You also might do one or more tests (experiments) to simply collect a lot more information about a problem. Any test (or experiment) could involve making one or many measurements.

If you are taking several different measurements, you do not want the conditions under which those measurements are made to change. In other words, you want *no* variation in the system between measurements. One way to achieve this is to have enough equipment to take all measurements simultaneously. If you don't have that much equipment, or such measurements cannot be captured at exactly the same time, you should make an effort to understand how your system might be changing during the time between different measurements.

You will also do experiments in which you intentionally make one (and only one) change in the system to try to understand the effect of that change. Repeating the obvious, if you change more than one thing at a time, you will never know which thing you changed was the cause of any resulting change in behavior or measurement.

Finally, as you prepare your next experiment (or next test), you should be able to predict the outcome as your insight into the system improves. When you have excellent agreement between your predictions and results, you probably have enough understanding to begin thinking about the root causes of your problem and the solutions you could apply.

Experiments and Tests

1. Experiments can help disprove a specific idea.
2. Experiments rarely prove a theory, but can give you good evidence to support a theory.
3. Experiments can be done to collect lots of raw information.
4. Only change one variable at a time during any experiment.
5. Try to predict the outcome of changes. When you have excellent agreement between prediction and outcome, you probably are gaining some insight into the problem.

Chapter 10: Debug by Division

There is a special strategy that is usually applied to locating problems in complex systems. It goes by many names: debug by division, divide and conquer, and binary search. Whatever you call it, the bottom line is that you want to organize your debug efforts to get to the root cause (i.e., the real problem) as quickly as possible—or maybe I should say as *efficiently* as possible. Sometimes, going faster is not the only measure of success; indeed, it could mean that important steps have been skipped.

The key strength of debug by division is the same as any binary sort or binary search. You reduce the number of steps required to the nearest power of 2 required to represent your universe of steps or stages in a design. For example, if you have a design that encompasses 1,000 blocks, debug by division should let you zero in on the problem area in only 10 steps ($2^{10} = 1{,}024$).

Let's look at an electrical circuit design example. Assume the problem has been reported that given a correct input signal, the system is not producing any valid output signal. If you have a design with only three sequential signal-processing blocks and the output does not match your expectation, you can use debug by division to guide your probing and measuring of the signal quality at various points in the circuit.

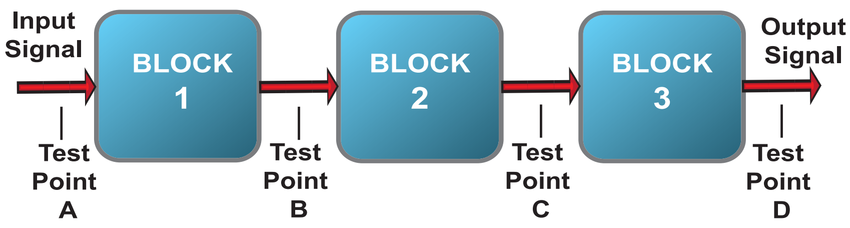


Figure 13-1: General System Block Diagram

First, verify that the problem report is accurate. In other words, with a proper input signal at test point A, make sure the signal at test point D really is an invalid output signal as reported. Occasionally, you might find that the output signal at test point D is actually just fine. You would then need to determine why you are not reproducing the problem.

Assuming that you can reproduce the problem, you should then verify that the signal input is as you expect. You measure at point A. If that incoming signal is incorrect, you probably do not need to debug anything in your circuit. In other words, if the input signal is bad, then it would not matter what design you use after that point; the result would still not match your expectation.

The one exception to this is if your circuit is causing the otherwise good input signal to suddenly become bad. An example would be if you had too much input capacitance, which instantly filtered and rolled off high frequencies on the incoming signal. Likewise, a circuit with improper input impedance might cause signal distortion that cannot be fixed later in the design. When the input impedance does not match the feed circuit, some amount of the incoming energy is reflected. This varies at different frequencies for the input signal. You can verify that the input signal is correct with an ideal load in place of your questionable circuit.

Now let’s assume that the input signal is fine, even with the sample circuit connected. In that case, you divide your system in half or, in the case of an odd number of blocks (for example, three blocks), you can choose to put your division point on either side of the center block. (It does not matter which side.) You then check the operation there.

- If the signal is okay at that point, you again divide the remaining circuit toward the output where the signal has been declared “bad” in half and measure again.
- If the signal is not okay at that point, you divide the incoming path in half and measure there.
- If you are down to a single block between good and bad

operation, you know that the failure is taking place at or inside that block.

If your design "reads" from left to right, and you have not yet found the problem, then you step to the right in halves until you observe the problem in your next divide-by-two measurement step. After you have found the problem, you step to the left in successively smaller halves until you no longer see the error.

A binary search is incredibly powerful in this way. You can isolate a defect in a very large system in relatively few steps.

Well…maybe.

There are always exceptions. Take the example of five cascaded amplifiers (each stage feeds an amplified signal to the next stage), all running from a common power source. If that power supply fails, then the signal will not meet expectations at *any* of the intermediate steps. The problem in this case is not in any of those amplifier stages. If you had checked the power supply first, you would find the problem even faster than locating the trouble to the first amplifier stage and then debugging that individual amplifier.

Similar problems occur in digital circuits when a common clock signal is used to step each piece of sequential logic. If that clock is not running or is running at the wrong frequency, the circuit will never operate as expected.

For this reason, the best digital-circuit developers practice a mantra when debugging designs:

1. Power
2. Clock
3. Reset
4. Memory
5. I/O

If the power is bad, then nothing works. If the clock is missing, then nothing will work. If the clock is too fast or too slow, then maybe nothing will work or maybe some things will act strangely. If the reset is held active, then nothing will work, although the power and clock will measure correctly. If there is noise on the reset, then the system will behave badly. If the non-volatile memory is bad, then the system will not boot. If the volatile memory is bad, then the system might only partially boot. If the I/O is bad, the system will not be able to fully function, even if it appears to boot.

Can you see why the mantra commands you to verify each of those items in the correct order? You can waste a lot of time debugging a later item if an earlier entry in that list is bad.

Say it with me now, "Power, clock, reset." That simple phrase can save you hours of time and effort. Designers working in other skill areas will no doubt have similar mantras for their expertise. Learn these ultra-quick checklists from your mentors.

Divide and Conquer (Debug by Division)

1. **Verify the problem (output is bad).**
2. **Verify the input is good.**
3. **Start testing near the middle of the system. Keep a record of every test.**
4. **If the result is good, set that point as your new start and then pick a new middle.**
5. **If the result is bad, divide the system from the start point to your current point.**
6. **When no more division is possible: diagnose that specific node (stage or block) of the system.**
7. **Check for common subsystems that affect all nodes or stages.**
8. **Learn the first-check mantras for your skills.**

Chapter 11: Stimulus-Response testing

There is a special subset of debug by division called stimulus-response testing. In its most basic form, stimulus-response testing asks the question, "What happens *here* when I do *this*?"

Stimulus-response testing can be used as the specific tactic of debug by division. For example, if you have a guitar amplifier, you might feed a fixed signal at 1 kHz into the input jack. You can then listen to or measure the output (or at any point in between input and output) to try to observe what has happened to that original signal.

One of the simplest examples of stimulus-response testing comes in the form of a push-button switch, used to activate some function on a product. The stimulus is the action of your finger, and the response is everything that happens from the time you start pressing until you completely release that push button.

I like to use this push-button example as an interactive brainstorming exercise to create a list of all of the ways you can vary this test or describe the behaviors you expect to observe.

- What do you expect to happen when the button is pressed?
- What do you expect to happen when the button is released?
- What do you observe or capture when you press the button?
- What does it feel like when you press or release the button?
    - Does the button have a nice surface?
    - Is the button motion (travel) smooth?
    - How much force does it take to press the button?
    - Does the button catch or rub on the edges of its cavity?
    - What happens if you use a fingernail to press the button?
    - What happens when you press off-center on the button?
- What does it sound like when you press or release the button?
    - Is there a noticeable click? Or more like a thud?
- What do you see when you press (or release) the button?

- How many times can you push the button before it fails?
- Is there any marking on the button?
    - How does the marking look visually?
    - Is the marking easy to read?
    - Is the marking clear in low-light?
    - Does the button use consistent graphics, fonts, and naming?
    - Is the button illuminated?
        - How long will the illumination source last?
        - Does the button get hot?
    - How many times can you push the button before the marking wears away?
- Does the electrical switch suffer any “bounce” creating multiple activations?
- What happens if you press two buttons at once?
- What happens if you don’t stop pressing the button?
- Are there any special times that pressing the button does not work?
- Are there any special times that the user should never press the button?
- Can the user find the button? (Don’t laugh, there are real-world examples of buttons placed so poorly that most users could not find them.)

Wow! This has generated a huge list of questions—but has barely scratched the surface of a real system design. I am just talking about one button and one finger!

Whether we are designing a new product or investigating a problem with an existing product, there will be many times that we use stimulus-response testing. We want to poke the system in a controlled way and see how it responds.

> There is a secondary question, which I will skip for the moment: What happens when you poke the system in an *unexpected* way?

When the system responds to your stimulus as you expect, you probably can look elsewhere for the cause of a problem. As my mentor George says, "When everything seems okay, it probably is." But when a piece of the system (block or component) responds in an unexpected way, you must investigate that section until you completely understand the behavior. You are trying to answer the question, "Why did the system do something we did not expect when the stimulus was *this*?"

Normal Environment Testing

In an electrical circuit design, you might need to know the response of an amplifier to various input amplitudes, frequencies, transients, and noise. In addition to those direct signal input conditions, you will also need to understand the impact of environmental conditions such as power-source variation, temperature variation, air-pressure variation, output load, driver-source impedance, and on and on. That means you need ways to vary each of these parameters while measuring the output behavior. Sometimes you might need to vary several parameters at the same time. Yikes—that sounds like a lot of work!

Yes, it can be a lot of work, but fortunately you can find many good ways to reduce the effort. Before you start varying all of the environmental parameters, make sure that your design is stable and working well in a normal environment. For example, this could mean nominal power, room temperature, normal pressure and expected loads.

Once you have built confidence that the system works well with all conditions at a nominal operating point, you can start to vary the operating conditions. You are getting to the fun part where you get to torture your system.

Three specific ideas to keep testing within reasonable limits are as follows:

1. Prioritize your testing
2. Automate your test sequences
3. Select key points and do point-testing in those areas

Method 1: Prioritize Your Testing

The first thing you need to do is prioritize your efforts. That simply means you do the most important tests first. It might also mean that you commit more of your total test time to the higher-priority tests.

For example, an electronic-circuit design absolutely must perform well over the entire range of allowable power-supply voltage(s). Likewise, a mechanical or electrical design absolutely must perform well over the specified operating temperature range for the product. Materials are selected and tested to show that they have the proper strength (or flexibility or transparency or some other characteristic) over the range of normal raw materials and processing. As a third example, a building darn-well should not fall down under normal wind, rain, and earthquakes.

What stress parameters are most important to your skill? Is testing against those stresses part of your normal design checklist? Which stresses fall below your normal checklist cutoff line? Do you have good justification for skipping those tests?

Most folks will try to vary only one parameter during a test and observe the direct effects of that variation. As with any experiment, if you vary two parameters, you can never be absolutely sure which change caused the behavior you observed. This does not mean that you cannot have lots of testing with multiple variations, however. It just means that you probably want to try to step through a range of variations for a first parameter and then adjust the second parameter and repeat the previous range of variation for the first parameter.

Here is an example: Suppose you have a problem report that a particular IC misbehaves when it starts up cold. You have some suspicion that the power-supply voltage is involved in the sensitivity because all the failing units happen to measure with a particular power-supply rail in the low end (but still allowable) of the specified range for that IC. But because the failure happens only when the system is cold, you also want to vary temperature.

For this case, you might try startup cycles with various voltages. You place the product in a temperature chamber and run a series of tests at one temperature with a series of incremental power-rail voltages. Then you repeat this series of tests, incrementing the temperature by some reasonable amount. If you find an inflection point where the system changes from bad to good or good to bad behavior, you can then fine-tune the steps to get a more accurate indication of the failure conditions.

You have to use your own common sense to know when you have data that is good enough. For example, a statement like "The system fails when the PSU voltage is between 3.00 and 3.3 volts DC and the temperature is between 25° and 50° C" might not have enough precision to enable others to reproduce your inflection point. But statements like "The system fails when the PSU voltage is between 3.215697 VDC and 3.227437 VDC volts DC and the temperature is between 27.6579° and 27.6601° C" might imply a false accuracy. Other folks might be able to reproduce your results by knowing that the "failure has been observed between 3.21 and 3.22 VDC at temperatures slightly above 27.6° C." Not all samples might fail at exactly the same place, so you just need to convey the right area.

Method 2: Automate Your Testing

One good way to reduce the test effort is to automate your test tools. This lets you repeat your tests under a wide variety of test conditions. Automation lets you collect test data at far more points than you could do by hand. There are as many good approaches to automation as there are good engineers. This is an area where you should avoid religious wars about programming languages (tools) and focus on getting the job done.

In larger organizations, you can often turn to interns or co-op students who will have the time, interest, and focus to attack these automation tasks. The downside is that you are handing off a critical effort to somebody who is less experienced. The upside is that as the more-experienced person, you can focus on the big picture of perfecting your product design while the intern is learning about the product (or process or system) by discovering how it is tested and helping you speed up that testing.

When you set up automatic tests, it is really important to understand the characteristics of the measurement system. Your automation must include the concept of settling time. This is where you allow an appropriate delay between the time your stimulus pokes the system and the time when you measure the effect. That delay might need to be zero (if you are looking for transient effects) or it might need to be longer if you have a long path between the stimulus and the expected response.

Method 3: Key Point Testing

Another good method to reduce the total effort in testing your products is to select key points along a continuous curve and simply test near or at those points. For example, if you are designing an audio amplifier, you might test the amplitude-versus-frequency response against certain environmental factors by choosing a relatively small number of meaningful frequencies. For example, you might see 20 Hz, 200 Hz, 500 Hz, 1 kHz, 2 kHz, 5 kHz, 10 kHz, and 20 kHz as representing low, middle, and high ends of the normal human hearing range. You could then run specific point tests at those frequencies over a range of tests that vary something else, such as the temperature. The curves (measurements) are captured and can be plotted to quickly identify any unwelcome dependencies.

Somewhere during your product testing, you want to do a really exhaustive and extensive test of certain parameters (like the aforementioned amplitude-versus-frequency audio test).

But you don't have to repeat that massive data collection for every single test cycle against every other conceivable stress or environmental factor.

If you find inflection points in your design, be sure to include some additional nearby point-test steps to expose the characteristics of your system in that region.

Overstress Testing

In addition to tests you do for normal operation, there are many fields in which you need to test the system or product with stresses that greatly exceed the conditions expected during normal operation. You can generally lump these into a category called overstress testing.

In electronics, you might do tests for sensitivity to electrostatic discharge (ESD) or electromagnetic interference (EMI). Many products are tested for mechanical shock and vibration. Drop tests are done on most products that are shipped in some way to their final user.

The truth is that as engineers, we should always anticipate conditions that might cause our designs to suddenly experience stresses far beyond their normal operating conditions. You want to know what happens then, when the system gets hit either from the outside or from a failure within the system.

Does this plane fall out of the sky when an unexpected wind gust hits a wing? Does a helpful medication turn into a poisonous substance when it sits without refrigeration in a truck for three days during summer in Phoenix, AZ? Does your house fall down when 24 inches of snow sits on the roof? Does your TV set catch fire when a two-cent diode short circuits, or does the protective fuse blow?

Nobody can protect against every conceivable stress. For example, most TV sets will not survive being tossed a mile or two by a tornado. The trick, then, is to understand what failures and stresses might be foreseeable and reasonable. The long list of safety warnings on a simple stepladder can be seen as a clue that such standards do change over time.

Regulatory Compliance Testing

Nearly every product or design eventually runs into some kind of governmental or industry-standard compliance requirement. Either you or some agency tries to verify that your product meets the rules and requirements of a particular standard. This might be for the safety and protection of the user, or it might be simply to protect the good name of the licensor of some intellectual property. You might have to fill out some simple or complex paperwork and minimal or extensive compliance test data might be required. The agency might do testing itself or simply review your paperwork.

Larger companies will have entire groups dedicated to these testing and reporting functions. Smaller organizations will often subcontract some or all of this work. In any case, you are never completely relieved of the need to understand why and how these tests are put in place.

There is a natural tendency for new engineers to see these tests and requirements as unwanted signals (noise) and lots of irritating paperwork. This is a common mistake—one I hope you will avoid.

The best designers want to understand all of the requirements of their design in advance. This enables those excellent designers to balance those needs during system design, subsection design, and component selection. They find the *center* of the design, which does not tip over too far in any direction.

The Nothing Test

One final note of interest: Often, the appropriate stimulus for a stimulus-response test is nothing. What happens to the system when you give it an input of zero, nothing, nada? Does the system stay stable or does it get all twitchy, hunting for a different input? In electronics, a nothing signal is often used for signal-to-noise measurements.

Stimulus-Response Testing

1. Do normal environment testing *first*. Build some confidence that your system works.
2. Next stress-test your system over the expected operating ranges.
3. You can reduce your test burden if you prioritize your testing, automate your testing, and use specific point-testing instead of continuous-sweep testing.
4. Don’t forget to do appropriate Overstress Testing.
5. Always do all regulatory compliance testing.
6. Test early and test often.
7. View testing as part of design.

Chapter 12:
"If I Could See it, I Could Fix it!"

One of the most common things I have heard people say about a problem is, "If I could see it, I could fix it!" If you carry any understanding from this book, I hope it is that saying.

Engineers are remarkably clever people. Something that has always impressed me is the ingenious ways that engineers find to "see" a problem. In some cases, I am talking about literally finding a method to see something happen with their eyes. In other cases, they try to visualize something in their mind's eye.

For example, mechanical engineers often build or modify an assembly to be a cut away where hidden moving parts are exposed. When some specific motion or action takes place, they can then observe the behavior of the individual parts.

Suddenly, a behavior that seemed strange or impossible becomes an obvious consequence of some small attribute. You find yourself saying something like, "Oh, I see it now. That little end piece on the plastic lever is folding up from too much pressure. We just need to make the actuator surface bigger to spread the force over a larger area. We could also use a stronger material, but that would cost more."

Or, "Oh, I see it now. That little piece that holds these other two pieces together has fallen out of the assembly."

Or, "Oh, I see it now. The oscilloscope shows a lot of random noise riding on the reset line DC average. Goodness! There is no leakage path to get rid of this unwanted signal. We have a very long wire on a very high impedance input."

Or, "Oh, I see it now. The water is coming in here through this little crack, and then running across this board until it comes out way over here."

Or, "Oh, I see it now. We forgot to include a label on this material stating 'HIGHLY EXPLOSIVE—Keep Away from Heat or Flame.'"

Figure 16-1 shows a photograph of a padlock with certain regions intentionally milled away. This cutaway design allows a locksmith to visualize what is happening inside the lock as the mechanism is actuated by the proper key or by an improper key.

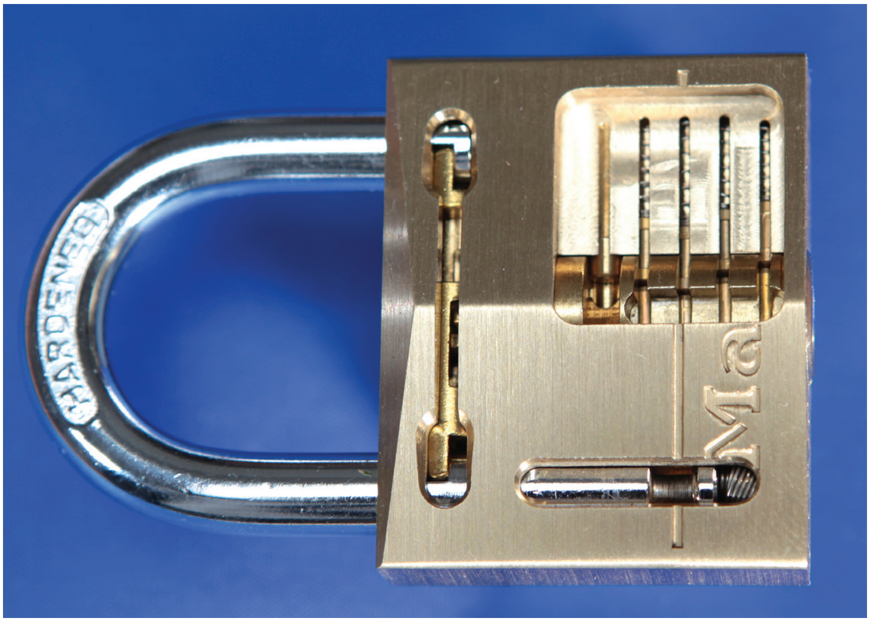


Figure 16-1: A cut-away lock allows us to clearly see and understand the workings of the cylinder and pins.

Figure 16-2 and Figure 16-3 show a close-up of the mechanism, first with no key inserted and then with a proper key inserted.

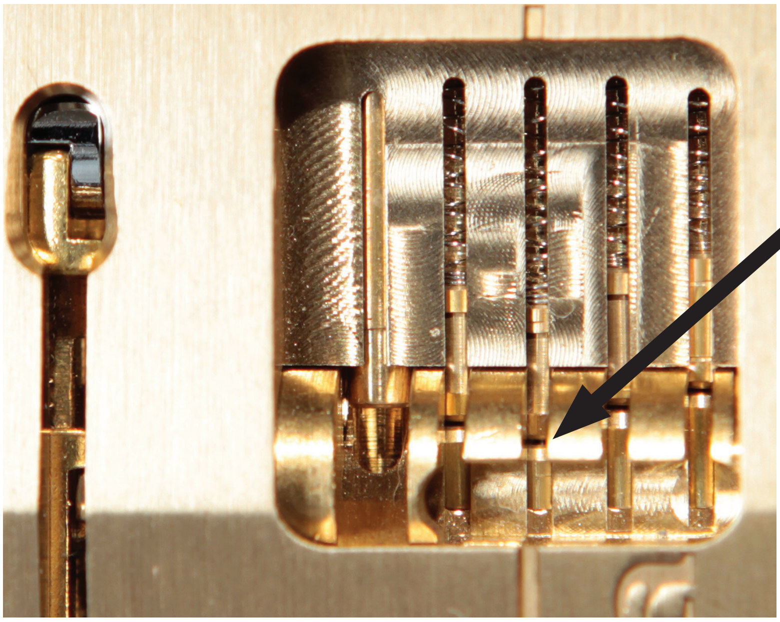


Figure 16-2: With no key inserted, the pins do not line up at the cylinder boundary.

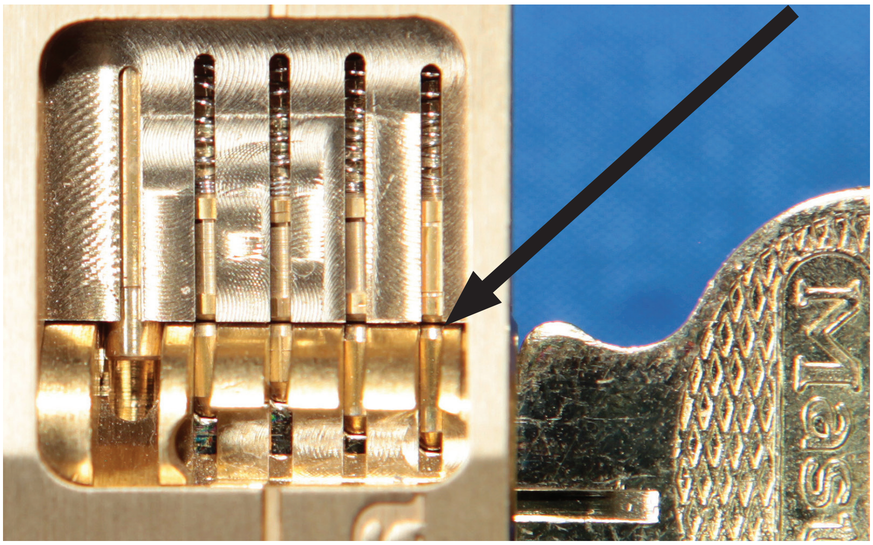


Figure 16-3: Correct key inserted, the pins line up; allowing the cylinder to turn, releasing the hasp.

Sometimes, a mechanism cannot work when it is open to the outside, such as the combustion chamber of an engine. In that case, a mechanical engineer might invent a viewing port made of glass, quartz, or plastic. Or the engineer might find a way to embed a tiny camera into a device. Sometimes, the engineer might build an entire sample from transparent materials.

Maybe the thing you are trying to observe is really, really small. The tools that help biologists see tiny micro-organisms can be used to study man-made devices, too. Magnifiers and microscopes are helpful. The trend to ever-smaller electronic devices has moved much of the electrical engineer's domain from unaided vision down to magnified, then down to extremely magnified viewing. Indeed, seeing the structures on modern integrated circuits requires leading-edge scanning electron microscopes.

When I started in electronics, we could find a visible defect on a circuit board with our own eyes. Eventually, we needed a magnifier to see small connections. Today, a good microscope is essential—and this is just to view the parts on an assembled circuit board, let alone to see the features of a semiconductor like an IC. X-ray microscopes let us see the shape of hidden ball grid array (BGA) solder balls.

A couple of decades ago, I owned a business that designed and tested industrial control computers. Like many businesses (more and more in today's environment), we did not do the assembly and soldering of our circuit boards. Instead, we contracted that function out. We designed the product and the test fixtures and did 100% of the product testing ourselves.

One of the things we noticed during testing was that certain problems repeated themselves, especially when our contract manufacturer added new workers. The most common of these was a simple mechanical assembly defect. At the time, most integrated circuits were not very dense and were housed in dual-inline packages (commonly called DIP ICs). Sometimes, one pin at the end of a row would fold under the device instead of going into the device socket or PCB hole. This also applied to the sockets that were commonly used to hold the ICs. Occasionally, an IC pin would miss the target and go to the outside of the socket or get bent under the socket.

This was what we termed a "purely mechanical defect." The device would have worked if the IC or socket pin had not bent out or under. The tricky part of this defect was that it was often very difficult to detect visually because a pin in the hole looked a lot like a pin that was bent under. A quick look almost always missed this defect. To address this, we developed some visual-inspection methods with bright lights and magnifiers to help us find this kind of defect.

The most insidious part was that sometimes the IC pin would actually make contact with the desired connection point on the socket. A given board might appear to work well and then might intermittently stop working.

In today's electronics with higher density packaging, you are not likely to encounter this exact problem. But you will encounter some defects due to poor mechanical handling of components during assembly. Good factories will be able to self-identify and correct such defects if they have proper in-house test fixtures. You might need to help them during factory start up to recognize and locate similar kinds of unintended component damage.

A Different Kind of Vision

In some cases, you don't need to see what is happening optically. Instead, you need some way to see something in your mind's eye. In other words, you want to mentally visualize what is happening, even if that mental picture or mental movie does not match the real physical process. Sometimes, the answer is to create a simulation. You substitute familiar objects and processes for those you cannot see to get your brain to absorb the meaning of what is happening.

Don't forget that you can substitute any of your senses for "seeing" a problem:

- If I could see it, I could fix it.
- If I could hear it, I could fix it.
- If I could smell it, I could fix it.
- If I could touch it, I could fix it.
- If I could taste it, I could fix it.

> Please be careful with these approaches so that you do not get hurt in an attempt to directly sense the behavior of a system. Let your tools do the work for you and stay safe.

Of course, sometimes we just need to look and the problem becomes obvious.

⏩ *Once upon a time,* Bart owned a small machine driven by a gasoline engine. (The exact function of the machine is not important to this story.) The machine was mounted on wheels and sat very close to the ground. The engine did not seem to run very well. It could be started, but ran very roughly and would run only a little more smoothly if the choke was fully applied at all times—even if the engine was already warm.

Bart was mystified. He knew that his list of "things I don't know" about engines was very long. He did remember that the engine needed fuel and air to mix in the cylinder before it ignited. This made it seem very strange to him that the engine wanted to have its choke set, because he knew that meant that the engine was acting like it was getting too much air instead of too little. "How in the world could it get too much air?" he wondered.

Finally, in desperation, he got down on his hands and knees and ducked his head down so that he could see the lower part of the engine. He could follow the path for the fuel from the tank into a small metal object. The air filter was to the side, so he could guess that the fuel and air were mixing in this small metal object. Bart seemed to recall that this part was called a "carburetor."

"Okay, so the fuel and air go in *here* and *here*, and then they go out—hey, wait a minute!"

Bart realized that the carburetor was supposed to be attached to the cylinder part of the engine with two bolts. But he saw only *one* bolt—and that attachment had backed away from the engine body by more than a centimeter. There was a huge gap such that extra, unfiltered air could mix into the fuel-air mixture before it entered the cylinder.

Bart decided that he could easily find an extra bolt and tighten the remaining bolt to hold the carburetor to the engine. It was obvious that the engine had shaken enough during normal operation to completely unscrew one of the bolts and almost fully unscrew the second bolt.

He got a bolt with the right size threads and then tightened everything back together. The engine started up and ran with no problem.

The interesting part of this story is that with almost no knowledge of the parts or operation, Bart was able to pinpoint the problem with the engine—just by looking at it. The answer was to get down on the ground and get a little dirty. The problem was so obvious that once Bart could see it, he could fix it.

A Wonderful and Inexpensive Tool for Seeing

In the past few years, a new tool has become available to help people see some problems: digital cameras. Digital cameras often come with lenses capable of producing outstanding macro photography. Small objects can be made to fill the entire photograph. When viewed on a large monitor or as a large print, tiny details are greatly magnified.

Thanks to this tool, it has become possible to take photographs of objects and systems that previously would have been very difficult to fit under a classic microscope and illuminator arrangement. There is a bit of skill required to take good macro photographs; fortunately, plenty of guidance is available in books and (gasp!)

even on the Web.

In general, you need to shoot at higher aperture numbers (i.e., smaller aperture settings), and you almost certainly will need to attach the camera to a tripod or some other fixed-position mount to prevent motion blur. External flash or high-intensity continuous light sources can help illuminate small features in macro photographs.

Remote Vision

⏩ *Once upon a time,* Roger's mother had gotten herself into quite a problem with her personal computer. Like many technical workers, Roger provided problem-solving support for family members who got into trouble with their modern widgets. In this case, Roger's mother was struggling more than usual—which was not really surprising considering she was 90 years old and trying to decipher the mysteries of Microsoft Windows.

Roger had intended to install remote-access software on her computer the last time he had visited his mother, but that task had been deemed unnecessary. At the time, things were going well. But that meant that on this day, Roger was left trying to solve his mother's problem without being able to see the problem.

The more Roger's mother tried to describe the problem, the more confused Roger became. The words she used did not match the image in his mind of how her screen should appear. This most recent problem was urgent: She could not reply to an important email until the problem was solved. But Roger could not figure out what she was describing.

To fix his mother's problem in person would require a five-hour drive there and another five-hour drive back home. He was confident the fix itself might only take a few minutes, so the 10 hours in the car sounded like a low-productivity use of his time. If only he could see her screen!

Suddenly, an idea popped into Roger's head. "Mom," he said. "Do you still do a Skype call every week with your good friend in California?" Roger knew that friend was battling cancer and there was no assurance that these calls had not come to an unfortunate end.

"Yes," she said, "Every Saturday without fail."

"Excellent!" Roger finally had a way to attack this problem. "Start up Skype now. I am going to call you." In less time than it takes to describe it, they had established a Skype video connection. "Keep listening on the phone," Roger said. "I want to be sure that our verbal communication does not get interrupted."

"Here is what I want you to do," he explained. "Pick up the video camera from on top of your monitor and point it at the screen instead of at yourself. You may have to wrestle with the wire a little bit, and the focus might not be so good, but hopefully I will be able to see something."

The video feed in Roger's Skype window looked a bit like the view out a porthole of a tiny ship caught in a hurricane. Images flashed and smeared back and forth. The scene was a roller-coaster ride of snippets from his mother's computer.

Then suddenly, in the corner of the image, Roger could see for just an instant the email application window. In that instant, Roger understood what she had been trying to describe and why she was confused. A collection of display rows had been collapsed into a single line, where normally they appeared as a neat vertical listing.

Roger's mother returned the camera to its normal position. After he walked her through the process of expanding the display rows, she could see the expected email listings where before there had only been a single row.

Roger was thrilled and disappointed at the same time. He had avoided driving 10 hours to fix a simple problem, yet he was missing out on a chance to visit his mother.

He congratulated himself for coming up with such a clever debug tool, but eventually it seemed obvious to him. The best ideas always seem obvious—after you have them.

The next time Roger visited his mother, he installed remote-access software. It was a bit difficult to use—he had to walk her through some settings each time he wanted to remotely access her machine—but the tool gave him a much easier way to see exactly what she was seeing on the screen.

They later fixed many problems together using the remote access software, but none were quite as much fun as the roller-coaster camera debug.

If I Could See It, I Could Fix It

1. **Your brain helps you convert images to understanding.**
2. **Sometimes, you just need to see the problem, and then it becomes obvious.**
3. **Use every tool available to help you see.**
4. **Sometimes we need to "see" with our mind's eye, not our physical eyes.**
5. **Digital cameras with macro capability open up new possibilities for viewing small devices.**
6. **You can see things remotely using inexpensive cameras and software.**

Chapter 13: Measurements

Remember, you are trying to find out the stuff you don't know. To achieve this, it may be that you need to extend the concept of "if I could see it, I could fix it" to things you cannot observe using the literal meaning of "seeing."

One of the best debug methods you will ever use is to apply some instrumentation to your system. This is where you connect (or insert or apply) some kind of measurement device to your system that is not part of the normal system. That measurement device converts a property of the system to a display or value that you *can* see.

Sometimes, you won't have a clue about where to look (measure) in the system. Instead, you will start with a broad sweep of measurements that are mostly to verify your basic assumptions. Often, you will have some kind of working theory about the failure so you have a better idea of where to look in the system. Hopefully, as your debug progresses, you are targeting specific areas more closely.

If you find yourself looping back to broad-sweep measurements, it might be a sign of one of the following:

- You don't have much confidence in what you know.
- Your system design is not very stable yet.
- Underlying assumptions might be false.
- The high-level design is changing without good documentation and tracking.

Choosing the Right Tools

I often ask electrical-engineering job candidates, "If you were sent to a remote location to fix some piece of equipment that you had never seen before, what single item of measurement gear would you take with you?"

The answer I am hoping to hear is "an oscilloscope" or perhaps "a digital storage oscilloscope." (For RF folks, a spectrum analyzer is an equally good answer.) The reason I look for that answer is that while both a voltmeter and an oscilloscope can give you a reasonable estimate of voltage/current in a circuit, only the oscilloscope adds the ability to see the element of *time* in the signals. This means you can estimate frequency or make general observations of signals, noise, and time-varying changes.

In electrical product development, the most common instruments we use are probably voltmeters and oscilloscopes. Temperature sensors of various kinds are used for thermal studies. I don't know the equivalent best answer for every skill area. The people you trust as mentors should know this for their skill, so be sure to ask them. I hope that you know the instruments used in your skill area better than I ever could. If not, that is a good place to start learning.

In general, most skill areas have data recorders or instruments that let you observe and capture (record) a specific measurement over time. They let you "see" the behavior of the system, and they let you record that behavior and play it back as often as you need. In almost every case today, these are computerized instruments, which can easily pass their data collections in common file formats. That makes the results easier to graph, to analyze, and to communicate to other folks. It also increases the accuracy of your information because (hopefully) there is no hand-transcription of the data.

Don't forget that other senses might apply. Perhaps I should have said, "If I could hear it, I could fix it." The sense you need to apply will be determined by the problem. For example, if you needed to know the specific moment in a sequence that a particular event was happening, it might be just as helpful to sound a bell or a beeper as to light an LED. This would be especially important if you needed to watch something else going on in the system and just needed an indicator to know when some event happened in relation to other events. Using an acoustic marker frees up your vision for other observations.

It is interesting to think about why so many instruments convert the value we wish to measure and capture into something visual. I believe this is true because our brains have developed very strong visual-processing abilities. Evolution has given humans incredible ability to recognize patterns, sometimes buried deep in noisy data.

False Measurements

Unfortunately, sometimes that need to find recognizable patterns can lead you far astray. You can be so anxious to find useful information that you “see” patterns where none exist.

There is a common occurrence with medical students in which they put together a few symptoms that they believe they are experiencing and suddenly determine (believe) that they are suffering from whatever exotic and extremely unlikely disease that they have most recently studied. They cannot help it. Their brains are programmed to find these patterns. In a very real way, that skill is working too well.

Equally, people can be fooled by false representations when they make false measurements. For example, if you greatly under-sample a signal, aliases will appear at incorrect frequencies. If you measure a voltage that has some DC and AC with a DC voltmeter, you will get false, misleading, and maybe even dangerously wrong readings.

Does Your Measurement Affect the Thing Being Measured?

All measurements have some impact on the thing being measured. You need to understand whether this is an effect that you need to worry about or if it will be small enough to discount it in your data.

One of the most common experiences in electronic system design is to find a circuit block that works badly—but suddenly works quite well when an oscilloscope is connected to some particular node in the circuit. At the next engineering meeting, there will always be some wag who says, "No problem, we will just need to ship a 'scope with every unit!"

What is happening to the circuit? In many cases, what you will discover is that the circuit design is exquisitely sensitive to the amount of capacitance at that node. The oscilloscope probe adds some small amount of capacitance (typically between 1 and 10 picofarads—yes, that is 10^{-12} farads), and that tiny capacitor is enough to shift the circuit from non-working to working.

Inevitably, some casual observers will quickly say, "Okay, so you just need to add a little capacitor to your circuit and then it will always work like it does with the oscilloscope connected."

Almost always, this is a massively, monstrously, epically bad idea. First, you need to understand the exact mechanism that is at work here. Yes, the primary electrical characteristic of that oscilloscope probe is mostly capacitance. But there is also some small resistance to ground (usually 1 million ohms or more for active probes). Finally, the probe will also act as a tiny antenna and could couple noise into the circuit or act as a path for tiny ground leakage currents.

I have seen circuits where just connecting the oscilloscope ground to the system chassis ground was the change that drove the circuit from working to failure (or from failure to working). Be sure to check for this effect when you find that connecting your 'scope makes any noticeable difference to system behavior.

Even if capacitance is the only significant factor, you need to figure out why that extra capacitance is making a difference in your circuit operation. Also answer the question, “Is there some other effect besides the capacitance of the probe that is making a difference in my circuit operation?”

There are a couple of possible effects, and you will need to understand which is at play in your circuit. Stray capacitance to ground can simply act as a filter, removing some high-frequency noise that was previously upsetting your circuit. In this case, you had better find out the source of that noise. Is it internal to your system? Is it coming from a different circuit block? Is it coming from outside your system?

Stray capacitance to ground can also cause a slight time delay on digital signals. A given signal transition will arrive later than it would have arrived without the added capacitance. Again, noise on the signal can be reduced so that the slower transition will have less high-frequency noise and therefore might make a smoother transition through the switching threshold of the logic. Is your circuit double-clocking on the noise? Or is your circuit working (or not working) because of a change in setup or hold time from the delay?

For analog circuits such as video or audio, the additional capacitance and resistance of a probe will change the shape and performance of any filters in the circuit. You will need to estimate the effect of these on performance. An audio circuit can often survive hundreds of picofarads of added capacitance with only a slight increase in distortion and no noticeable loss of audible high frequency, but even a few picofarads can be enough to destroy the parametric performance of a video circuit.

Laying a plastic ruler on a block of steel is not likely to change the length of that piece of metal by any noticeable amount. But laying a steel ruler on top of a single bacterium might end the life of that sample and ruin an experiment. You have to make some judgment calls in your measurements and have a good idea of the relative scale of what you are trying to measure.

When the measured performance of a system seems to be unstable or difficult to repeat, it is a good idea to look at the characteristics of the measurement system and evaluate how they compare to the thing being measured.

Measurement Errors and Tolerances

Every measurement you make will have some error. Similarly, every component you use will have some variation, which represents a kind of error against an idealized design. We call the allowable variations “tolerances.” Hopefully, tolerances and measurement errors are so small that your data is still meaningful.

If you are doing a problem investigation that involves lots of measurements, at some point you need to do an error and tolerance estimation. Include that information in your report of the experimental data. If your measurement methods don’t change and your instruments don’t change you probably only need to do this kind of error and tolerance study once during any investigation.

⏩ *Once upon a time*, Boris was investigating an apparent television signal quality failure. The customer had specified a specific signal parameter (color saturation accuracy) to be no greater than 2%. This created some concern, since some samples were showing measurements slightly outside this value. Boris did an error/tolerance study and found that the industry standard measurement instrument (the test instrument that everybody in the industry used) only offered a basic absolute accuracy for chroma amplitude measurements of 1%. In order to guarantee a 2% measurement, Boris needed his device to be better than 1% absolute accuracy, since the instrument could contribute that much error by itself. This seemed strange and excessive.

At some point during the investigation, Boris and his coworkers had a sudden insight. The customer was using a video measurement which indicated the absolute voltage level of the color signal within a complex video waveform. They were then assuming this value told them the color saturation of the video. The team realized that saturation in a composite video signal is represented by the *ratio* of the luminance to the chrominance signal.

Suddenly it was clear what was happening. Small variations in amplifier gain showed up as changes in the luminance *and* the chrominance signals. This meant that the color saturation accuracy would not be affected by these small changes in gain, since that parameter was represented by the ratio of these two measurements. The same gain error would cancel out in any ratio calculation. However, the customer was ignoring the ratio by doing an absolute measurement. [In fairness to the customer, no automatic method for measuring color saturation accuracy existed in any of the common instruments.]

Some deep library research showed that multiple studies (40 years earlier) had shown that this particular signal characteristic could not be observed by viewers at less than about 15% error. Boris and his team were trying to meet an unrealistic requirement. The final step was the discovery that the interconnect cables commonly used for this signal often contributed approximately a 5% absolute amplitude error!

The customer eventually was convinced that the product was meeting the spirit of their requirement if not the absolute measurement and the product was accepted without any design change to fix this problem.

You might have heard this saying about tolerances and precision:

“Measure it with a micrometer; mark it with a crayon; cut it with an axe.”

Like the previous video-signal example, it does little good to design part of a system with extreme precision if the next part of the system is drastically less precise. Systems need consistency. More than this, they need common sense applied. It does no good for quality if you measure with high precision but mark it (i.e., create your design) with gross errors. Likewise, the manufacturing methods need to be matched to the design tolerances. Do you cut with an axe or a laser beam?

Precision is the ability to represent a measured value to more digits. For example, 3.3001 volts is a more precise measurement than 3.3 volts. However, if your measurement instrument has an absolute accuracy to only 0.1 volts, all that extra precision does not help. It is meaningless.

Here is a fun exercise that you can try. What do you think is the most accurate or precise device in your home? Two reasonably good answers would be either your personal computer's hard disk drive (HDD) or your DVD player.

After all, DVD players read microscopic marks on a disk. It takes astonishing accuracy for a DVD player to find, focus, and follow the bit track as it spirals out of the center of the disk. It is especially amazing given today's rock-bottom prices for such a device.

Likewise, the HDD in your computer must follow bit patterns recorded into the magnetic surface of the platters. Just a few years ago, the track-following accuracy for placing the heads was approximately equal to the diameter of 12 atoms of copper!

Here is another good question. What is more accurate: the real-time clock in your laptop computer or the 60 Hz power in your house? This turns out to be a trick question. The instantaneous accuracy of the 60 Hz power-line frequency in your house will vary quite a bit, but the average accuracy is actually very, very high. This is because your power company intentionally varies the short-term frequency to maintain a long-term average accuracy.

They do this to be sure to keep motor-driven clocks very accurate. In contrast, your laptop real-time clock runs from a quartz or ceramic crystal oscillator. The instantaneous frequency does not vary much from moment to moment. However, the absolute accuracy of that oscillator will probably be off by several minutes per year.

Sensitive Nodes

What do I mean by sensitive nodes? Do I mean circuits or nodes that easily get their feelings hurt by criticism? Do I mean parts that intentionally respond to touch? Do I mean systems that are so secret, I must not tell anybody how they work? The answer to all of these questions is no.

When people talk about sensitive nodes, they mean designs or nodes within a system that change their behavior greatly when very, very tiny changes are introduced either to their components or by the addition of some small extra element. If you find that something in your design is greatly affected by a reasonable measurement, then you probably have a sensitive circuit.

No doubt, you are now jumping out of your chair and screaming, "How in the heck do I know what a *reasonable* measurement is?" In many ways, you are going to have to work that out for yourself. Experience helps, so you can discuss this with your network of experts. You can also do some rough estimation. I personally think a lot about general rules of thumb. One of mine is a 10% approximation test.

Good engineering says you should have some safety margin in your designs. Can your design allow a 10% variation in any characteristic of one component? Could your design allow 10% variation in all components? Maybe your design won't pass every measurement and every requirement, but would that 10% change make the design completely stop working?

For mechanical designs, if you have a normal tolerance of X millimeters in a manufactured part, could your design still function if the variation of that part is out of specification against your allowable tolerance by 10%? (If you allowed for X mm ±1 mm, will your design break if a given part is ±1.1 mm?) Will the parts bind together and start breaking pieces off when you are at the tolerance limit, instead of slightly less than the limit?

Another way to ask this question would be, did you allow your component supplier any room for error or realistic manufacturing variation?

Of course, the opposite error can creep into your designs. If you try to get too loose, to allow for every lack of quality in your components, your system as a whole can become too sloppy, too imprecise. Pretty soon, you have just made junk that does not work well in any case or that feels bad when you try to use it.

A system needs enough precision to work really, really well and to meet all requirements, while allowing for any reasonable variations in components. Sometimes, you will need to demand better precision from your components; other times, you will have to know when you are asking for too much. Price and availability of components are sometimes good guides to when you have pushed too far.

For electrical designs, sensitive nodes are almost always associated with high impedances. The laws of physics start to eat us up because a high impedance input will be exquisitely sensitive to any signal induced into or coupled onto any attached conductors.

Even having nothing more than a PCB pad on a floating, high-impedance input can spell trouble for a design. Capacitive coupling can introduce large voltages at such a node.

Electrical designers must learn to tie unused nodes to a low-impedance reference plane (ground, Vdd, or some equivalent). Software engineers should always ensure that any unused input/output pins are configured as outputs and driven to an appropriate active state—either high or low.

Tear-downs as a Debugging Tool

There is another method for helping you “see” a problem: You can tear apart a working system to try to find out if the process of putting the system together has fundamentally changed some parameter or aspect of the system that you did not expect. It is possible that the process of putting components together has modified those parts in some way.

A tear-down exercise lets you re-measure the performance of an individual component to be sure it has not suddenly changed after assembly. This process also enables you to verify that the components you specified are indeed the components used to assemble your product. It is amazing how many times you will find that a similar but unequal component has been substituted during assembly.

Tear-down reports are much more commonly used to study existing or competitors’ products or systems. With this tool, you can study how competitors did their system design, the specific components they selected, or whether they have created some clever new arrangement of basic parts. Often, the goal is simply to determine an estimated cost of manufacture for a competing product.

Reading tear-down reports can help you become a better designer. The more you see other designs, the better idea you will have how other people have solved similar problems. The goal is not for you to simply copy other people’s ideas; rather, you want to understand the things they did and why they did them.

Later, I will discuss the idea of “reading” designs, just as you learn to read books and magazines.

Measurement and Instrumentation Concepts

1. Remember, if I could see it I could fix it.
2. Use instrumentation, measurements, and data-capture to see hidden information.
3. Watch out for false measurements.
4. Understand your measurement errors and tolerances.
5. Does your measurement affect the thing being measured? (Of course it does, but is that variation important?)
6. Understand where you have sensitive nodes in your design.
7. Sometimes you need to tear something apart to really see what is going on inside it.

Chapter 14:
Instrumenting Software

The word *instrument* is normally a noun and typically refers to a physical device that helps you do something (play music, measure a parameter, capture some information). Like all engineers, I fall into the trap of converting nouns to verbs: "We need to *instrument* the system to capture this information."

There are not many physical tools dedicated to helping debug software. There are, however, some in-circuit emulator (ICE) devices that substitute for a given processor and enable you to set traces and captures for specific kinds of events. In addition, logic analyzers have been around for many years, but they are disappearing simply because too much of the critical circuitry is buried within a single IC or system on a chip (SoC). You cannot directly probe the internal buses to monitor them. Many of these buses are sensitive circuits and run at such a high frequency that it is increasingly impractical to add any circuitry to observe them.

Most new SoC devices include on-chip debugging abilities. For example, Freescale's background debug mode (BDM) and other similar debug ports have become standard in the industry. System chip vendors are finally addressing the need for embedded debug simply because they have found themselves increasingly in the position of needing to provide the drivers and to directly solve the hardware/software interaction issues that in earlier times their customers were required to fix.

Eventually, every software engineer (designer, coder, tester, or whatever title is appropriate) will find themselves in the same position as the general designer, muttering to themselves, "If I could see it, I could fix it."

Although code seems infinitely dynamic and changeable, software engineers create machines just as much as their predecessors in the mechanical and electrical fields. These machines are subject to all the same problems and worries discussed elsewhere in this book.

Because the machines inside the computer are virtual, there is a need to create virtual instruments within or around those structures and give those virtual instruments the necessary abilities to help you see what is going on in the virtual world.

Sometimes, that might mean adding some code to keep an extra copy of some internal variables and internal states in a capture buffer that can be read after an event or after a crash.

The biggest difficulty often faced in instrumenting software is that the bandwidth of the channels used to report this debug information is too low. One age-old method is to simply add "printf" or equivalent lines of code to report some brief information to a debug port. Often, these are fairly slow serial port interfaces. This is not a horrible method; in fact, it is highly recommended, especially for simpler systems and problems.

The downside is that complex systems sometimes end up reporting thousands or even millions of diagnostic messages. If you do not have extremely rigorous design rules for these diagnostic messages, they can become an impenetrable jungle of meaningless phrases.

Here are some good examples of rules for diagnostic messages:

- Every debug message *must* include a short "locator" word (or phrase or sequence of characters) that lets you identify the exact line of code which produced that message.
- You *must* (no exceptions) have a method that keeps each diagnostic message as an uninterrupted segment of text. There is nothing worse than debug messages that interrupt another debug message, producing nonsense in the output stream.
- You *should* have enough bandwidth in the reporting channel to allow you to get all of the critical information out of the system before (or even after) a system crash. You possibly won't achieve this last item.

One of the most common mistakes in embedded system programming is to leave some exception vectors (interrupts or software “traps”) uninitialized. The programmer says, “We don’t use this function, so that interrupt cannot happen.” This is usually a deadly mistake. The exception happens (for whatever reason), and then the CPU branches off into the weeds. In general, you never get any indication of why the system has crashed—just that it stopped working. Yuck.

To combat this, fill all of the exception vectors and trap pointers, even if you are sure they will never be used. Include unique handler entry points so you can tell which vector was taken. (You know, friend—the vector you swore you could never reach….)

This usually adds some hours of coding to your work. If you are integrating a commercial environment, this code might be down at a level that you normally do not touch. Be absolutely sure that somebody has taken care of this step. It can save you hours, days, or weeks of chasing intermittent unexplained failures.

There are some folks who insist that unused memory should be programmed with instructions that will eventually force a reset or restart of the CPU if random execution starts in those locations. This is not a bad idea, but the implementation will vary greatly with different instruction sets. Where multiple heterogeneous processors share a unified memory space, it might not be possible to use a single value this way.

Try to be sure that you never suffer from uninitialized memory problems. Your code can do an initial memory clearing. You can run your code through validation tools that check for this kind of error.

Tools like *Klocwork Insight* are very valuable in finding common errors in complex code. You should select the tools that you believe are appropriate and make sure they get used.

One additional tool you should use is code peer reviews. Like hardware design reviews, you are more likely to identify weaknesses when the system is presented and explained to other skilled designers.

Finally, I am compelled to include some advice borne from difficult real-world experience in chasing complex system problems. The toughest problems are usually solved when (at least) one hardware engineer and one software engineer work together during a debug. Each person brings different, skills and knowledge to the effort and each will recognize different patterns or behaviors in the system.

The next chapter dives into a topic tied closely to instrumentation: tools and toolmaking.

Instrumenting Software

1. **If I could see it, I could fix it. (Works for software too!)**
2. **Use static, extra memory locations to capture transient values.**
3. **Build “flight data recorders” into your code to let you see what happened just before a crash.**
4. **Follow the rules for diagnostic messages.**
5. **Pay attention to your diagnostic message bandwidth.**
6. **Initialize all exception vectors and have them point to useful diagnostic messages—even if you are sure they cannot happen.**
7. **Initialize all unused memory.**
8. **Use code validation tools.**
9. **Use code peer reviews.**
10. **The most difficult problems are usually solved by having the hardware and software teams working together—not separately.**

Chapter 15:

Tools and Toolmaking

Every good problem solver I have ever met is an expert at using certain tools. Maybe it is obvious and ordinary to you, but I have always been amazed at the interaction of these experts and their tool sets. These engineers master the use of their tools; they go far beyond knowing a few basics. Sometimes they dive into the internal design of the tool itself and find themselves proposing improvements to the tool. Quite simply, they love their equipment.

The opposite of this is a relatively poor problem solver, who has a very limited range of tools. There is an old joke: "To the man who only has a hammer, everything looks like a nail." (A corollary to this has also been stated: "When I only have a hammer, everything looks like my thumb.")

It is painful to watch someone work with a new tool the first time. We all are a bit clumsy and need to figure out how to hold it, how to move it, and how it works best. Equally, it is exhilarating to watch a master craftsman work with simple or complex tools. The tool becomes an extension of the craftsman's mind. Where does this transition from hopeless amateur to consummate craftsman take place? When does this transition happen?

One of my favorite quotes comes from former Indiana University coach Bob Knight. "Everybody has a will to win. What's far more important is having the will to prepare to win."[3]

In fairness, Coach Knight gives full credit to Bud Wilkinson, the great Oklahoma football coach, as the originator of this statement.

[3] From KNIGHT: MY STORY © 2002 by Bob Knight. Reprinted by permission of St. Martin's Press. All rights reserved.

Interestingly, numerous sites on the World Wide Web give (undocumented and untraceable) credit for this quote to a flock of other sports luminaries. I think the quality of the idea is so good that it is inevitable that people hear it and attach it in their brain to their favorite hero. Maybe having finished this book, a few of you will forget the details and attribute such a wise saying to me.

The skills of the master craftsman do not come to you on game day, when the pressure is on to perform miracles of problem solving. Rather, these skills are built up during long days and nights of testing in the early phases of projects. The transition to master craftsman happens sometime during that effort.

Suddenly, you find that you don't need to look closely at the controls on the instruments. You already know where they are. You know how many clicks to the right or left your optimum setting will be found. Your fingers take over for your cognitive brain. It takes a lot of practice and it requires the will to prepare to win—or in your case, the will to prepare to solve problems.

IMG2749-5438.JPG

The Right Tool for the Right Job

You don't want to be the bad problem solver—the guy who only has a hammer and sees everything as a nail to be pounded into oblivion. You want to understand the task and pull out the appropriate tool to complete it.

A digital multi-meter can give you extreme precision in making a DC measurement, but it is nearly worthless if you are trying to capture a runt pulse that shows up 2 nanoseconds after some transition on a particular memory signal. Likewise, a digital storage oscilloscope can capture that runt pulse, but cannot easily tell you the precise resistance of a milliohm PCB trace. Neither tool is useful when you need to capture a 64 bits data bus, 32 bits of address bus, and 7 control lines for 1 million cycles prior to a system crash. A logic analyzer is better for that task.

Every skill has specific tools that make some tasks easier, faster, or more accurate than other tools. It is your job to understand the different tools that are available and why each might be better for some tasks.

Let's face it: Sometimes, economic limits prevent you having all the toys—ahem, I mean *tools*—that you want. Even the best companies have budgets and limits. You need to learn to ask yourself, "Would that be the best expenditure of corporate cash?" If you owned the company, would you spend that money on that tool? What if it meant that the company had to fire you to buy that tool for the engineer who sat next to you?

Buying tools can be a tricky political game when you put it in those terms. You really have to assess whether the organization's goals are better advanced when they have more people (a different kind of tool) or more equipment. If you are the boss, do you want more hardware, more software, or more wetware? It won't do you much good to have the hardware and software if you don't have enough bodies to use them. Equally, it doesn't help to have lots of people sitting around if you can't support them with the hardware and software they need to do their jobs.

Balance, grasshopper. It is all about balance.

My friend Hugh once observed that in the period from 1970 to the late 1980s, a small business might be composed of five guys and one computer. By 1990, however, a small business might composed of one guy and five computers. Today, I think networks allow us to have more guys, but they might all work independently and in different locations. And each of them will use dozens of processors, but we don't call them all computers anymore.

Be prepared to help your managers understand the value (not just the cost) of the tools you need. You will need to clearly communicate the business advantages of providing you with necessary (not just interesting) equipment.

Make versus Buy

In every problem solver's life, there comes an "aha" moment, when he realizes that he could see or fix or really understand a problem if he only had the right tool. Suppose, however, that in a similar "aha" moment, he realizes he cannot buy this piece of equipment. Maybe it does not exist because the requirements of this task are extremely unusual. Maybe nobody has ever thought of combining the pieces together in such a way. Or maybe building and certifying such a tool for use by the public would cost too much.

In that case, sometimes the smart thing to do is to build the tool you need yourself. I have seen many really good problem solvers do this. They make the tools they need, but only those tools that it would not make sense to buy or that cannot be purchased from anyone else.

Sometimes, these tools might be some small simple gizmo or widget. Maybe it helps to hold something in place or to get access to something that normally is buried inside a system. The widget might let you connect to a signal that is physically or electrically sensitive. It might be a piece of custom software, developed just to test certain functions during development. It might be a complex mashup of many standard instruments and pieces, but put together creates something far more useful than those parts.

One time-honored tradition is that some of these test tools will suddenly blossom into an entire business or industry by themselves. The tool that solved a difficult problem for one engineer turns out to be useful to 10,000 others. If the current employer is not interested, sometimes the individual engineer starts a new garage shop to help other engineers solve their problems.

Again, we come to the balance thing. When faced with the decision to build or buy a tool, ask yourself a bunch of hard questions:

- Is this really something you cannot or should not buy?
- Can you build it quicker than you can get signatures on a purchase order?
- Will it be useful for other projects?
- Will you use it multiple times during the development of this project?
- Does it make sense to an outside observer?
- Does it make sense to your boss?
- Can you build it out of pieces that nobody will regret losing?

If the answer to all or most of these questions is yes, then you have a good case for making your own tool. Go for it.

Balance this passion to make your own tools against the constraints of your organization and the time, money, and results within your reach. You really need to understand your limitations.

If you envision building a million-dollar instrument for a project whose whole budget is $100,000, then you are not going to build that instrument. If your project must be done in two months and your widget will take a year to develop, you won’t be able to make it or use it (this time). If you can make that widget in the background, however, maybe you can have it ready for the next project.

As always, it is a good idea to involve your boss. It lets him know that you are a good problem solver—or at least a thoughtful problem solver. He might not support this effort, this time. You might have to find another way. In that case, he might have already been there and done that, and have some good suggestions. Or it may be that another group in your company has already developed a very similar widget.

Why do We Love Tools?

There is something deep and powerful between problem solvers and tools. It does not quite meet the standard of love between two people, but it sometimes comes frighteningly close. Maybe it is because tools can relieve pain and open new worlds to us.

Imagine driving nails with your hands. Imagine cutting thick wires with your teeth. Both would be painful. Or imagine being the first person to put a sample of blood or pond water under a microscope and experiencing the pure joy of discovering a whole new world.

I have carried a quote in my head since college. I had always thought it was attributed to Arthur C. Clarke, the science-fiction writer, but I have been unable to confirm that, even after extensive research. If you will please forgive the gender-specific form of the expression, it was this: “Man made tools; then tools made man.”

Do you know the true source of this quote? Is my wording so mangled as to hide its origin? I would really like to know the correct quote and attribution. Send me a postcard, drop me a line, an email would do just fine.

Measurement, Calibration, and Common Sense

Because I am discussing tools, there is one additional topic that I must address. When you make a measurement, you want to know that the value you get is correct to within some small error that you expect to be insignificant. To do that, you calibrate your measurements. This sounds very sophisticated and formal, and there are certainly many publications and standards for measurement. For example, a portion of the ISO-9000 series of quality standards are committed to calibration.

> As an aside, ISO-9001 can nearly be summarized as:
>
> 1. **Plan to do something.**
> 2. **Say what you do (in writing).**
> 3. **Do what you say.**
> 4. **Verify that you are doing items 1-3.**
> 5. **If it breathes, train it.**
> 6. **If it doesn't breath, calibrate it.**
> 7. **Hold management and all employees accountable for 1-6.**

How did this passion for calibration turn into obsession? It certainly started simply enough. Let's say you are making widgets that have a hole in them. You drill this hole with a very nice drill press. Maybe your customers want to hole to be 0.25 inches in diameter because they stick a bolt with that size shaft into the hole.

If you periodically measure the hole in your widgets, you find that as time goes by, it is getting smaller. Eventually, you might come to realize that some tiny portion of the drill is wearing away every time the tool makes a hole. Aha! So you build a little test fixture. When the drill gets smaller than a certain size, you throw away that drill bit and put in a new one.

That's calibration. Of course, you also have to decide whether there is any wear and tear on your little diameter test fixture.

Do the holes in the test fixture get bigger over time?

You want some traceability on the measurement that tells you the test fixture is the right size. Maybe once a year you check the test fixture. Every time you do that, you write it down. And once in a while, you audit those test records to make sure you don't forget.

This all seems pretty straightforward. You understand the general concept, the methods, and the record keeping. The problem is when this suddenly swirls into an ISO-9000 priesthood. You begin paying outside companies big bucks to calibrate every single instrument. Mysticism surrounds the actions of the calibration company.

Oddly, it rarely occurs to folks that they could calibrate a few key devices using that outside company and then cross-calibrate their remaining devices (instruments) in-house. Yes, it takes more effort. Yes, you have to understand all of the fundamental parameters that your instruments can measure (for instance voltage, time, mass, and so on). You would have to calibrate some primary instruments to traceable standards or obtain new instruments periodically that are calibrated by the vendor to traceable standards.

Ultimately, what you care about is simply how accurate your measurement device is and how accurate your measurement needs to be. (Here I am talking about accuracy, not precision.)

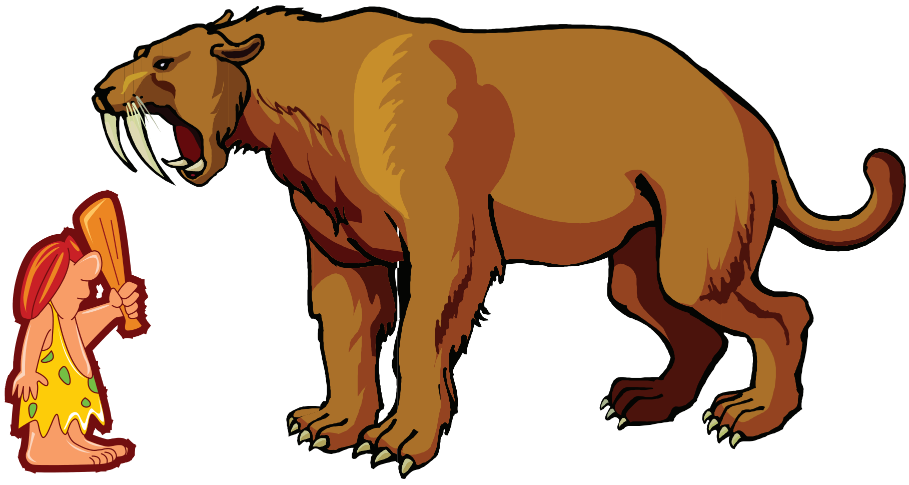
"How accurate does your measurement need to be?" can be reworded as "How much error do you allow in the measurement?" Clearly, you need to know how much error comes from the measuring device and all of its accessories. That's it, folks. Not much magic, but it does require a lot of care.

My experience here is that the best problem solvers have a really good understanding of the typical and worst-case errors from their tools. They have common-sense understanding of how to verify that their tools are not broken, not worn out, or telling lies. They understand the difference between accuracy and precision.

Then again, "common sense" turns out to be not so common….

Tools and Toolmaking

1. Practice, practice, practice until the tool becomes an extension of your mind.
2. Choose the best tools that your organization can afford and then make it work. Be creative and don't blame your tools.
3. Think about making special tools when that is necessary, reasonable, and appropriate.
4. Calibrate your instruments and keep good records of your calibrations.
5. Understand accuracy and know the common sources of error in your measurements.



Zog suddenly realized that he needed a much better tool.

Chapter 16: Troubleshooting Complex Systems

It is all well and good for me to sit here writing lots of little checklists (What do you know? What don't you know? Etc.) and giving you guides to finding out the things you don't know. But in the end, you are going to be the person confronted with debugging some incredibly complex system. It won't be a system with only three blocks; it won't have only hardware or only software; and it won't arrive with lots of good instrumentation that lets you quickly see what is going on behind the curtains.

When that happens, there are some rules to help you:

- Rule 1: Don't panic.
- Rule 2: Get organized.
- Rule 3: Be methodical.
- Rule 4: Record your work.
- Rule 5: Be persistent.
- Rule 6: Ask what changed
- Rule 7: Localize problems.

Rule 1: Don't Panic.

Your calm attitude will help others stay calm as well. Conversely, nothing inspires panic in your coworkers like seeing it on your face. Even if you have no confidence in your ability to fix this problem, pretend that you have loads of confidence. There will be plenty of time and energy later to waste on mindless panic.

Rule 2: Get Organized.

Gather your information and make sure it is readily available to you in whatever form is most convenient. If you work from a tablet or notebook computer, make sure you have collected all the files you need and have the necessary reader and viewer programs installed and working.

Do you need to keep your files secured? Different jobs have different levels of required security. Certainly, in some jobs, it would not matter if your entire computer and all of your files were suddenly exposed to the public. You (should) already know how sensitive your project information might be, so take appropriate care.

Use appropriate tools to protect your critical information. Computers get borrowed, lost, and stolen. There are ways to protect private information without making a computer unusable for other employees.

Do you need a lot of different passwords to access these files? Take the time to set up a secure password manager.

Is it more appropriate to use printed copies and physical binders in your work? If yes, feel free to create one or more loose-leaf binders with the critical information you are going to need.

Even with all the computer technology we have today, it can sometimes be helpful and reassuring to be able to reach for a printed reference. You can scribble notes onto these pages and keep track of your short-term progress there. Just be sure to convert your notes and discoveries back into electronic form when you need to distribute that information as a project or management update.

The best time to get organized is *before* you need to. Once a crisis hits, it can be difficult to keep everybody waiting while you sort out the documentation.

Rule 3: Be Methodical.

I am not consumed in religion about which debug method(s) you should use. Whichever path you choose, be sure to follow the method as completely and carefully as possible. Jumping from method to method only creates confusion and frustration. Erratic choices make it more difficult to communicate your thinking and definitely more difficult for other folks to follow and reproduce your work.

Rule 4: Record your work as you do it.

After all of the times that I have emphasized the need to write it down, do you think I won't mention it again here? So much of work is about communication. Get it documented and make sure that what you have written (or photographed or drawn) is clear and has meaning.

Rule 5: Be Persistent.

Don't give up at the first sign of trouble. There will be plenty of setbacks and many ideas won't work. Constraints will hinder every project. You will never have enough time or money. That is okay. What matters is that you develop the aptitude and attitude to solve problems big and small.

There are going to be times where you will try everything you can imagine and still will not find a solution to a problem. You will then try everything other people suggest and still won't find a solution. You might loop back several times: following suggestions, making changes, running tests and experiments and still no joy.

At this point, you must not become discouraged. You can solve this problem. But you might need to develop an ability to keep trying, even when your task seems impossible or improbable. You are going to need persistence.

The Enthusiastic Rooster

⏩ *Once Upon a Time,* a farmer needed a new rooster. He purchased a strong, brightly colored rooster at the local hatchery, and then placed it into the barn yard. He was pleased to see the rooster immediately began chasing and mating with various hens; doing a rooster's job with great gusto.

Now in stories like this, people and animals can talk to each other.

After about a week, the farmer stopped to have a chat with the new rooster.

"Rooster," he said, "You need to pace yourself. I see you working all day, every day. I am afraid you will wear yourself out too quickly. I don't have the money to buy another rooster this year. So please, just slow down a little and take it easy."

The rooster quickly replied, "Love it, Love it, Love it! Can't get enough, can't get enough!"

The next week, the farmer saw the rooster chasing some geese across the barnyard. "Hey, cut that out! I warned you that you would wear out if you keep doing that! And your job is the hens, not the geese!"

The rooster simply says, "Love it, Love it, Love it! Can't get enough, can't get enough!"

The next day, the farmer sees the rooster chasing a pig. Although impressed with the rooster's ambition, he repeats his previous warning.

The rooster says again, "Love it, Love it, Love it! Can't get enough, can't get enough!"

And on subsequent days, the farmer sees the rooster chasing ducks, dogs, cats and even horses across the farm fields. Again, he cautions the rooster to slow down.

The rooster only repeats, "Love it, Love it, Love it! Can't get enough, can't get enough!"

Finally, the farmer comes out one day and discovers the rooster laying in the dust, eyes closed, face up to the sky, one wing spread out at an awkward angle. The farmer shouts angrily, "Darn-it rooster! I told you to slow down! Now you done went and got yourself all busted up and I can't afford a new rooster!"

The rooster rolled open one eye to the farmer, waved a wingtip in front of his beak, "Shhh!" He whispered, then pointed up into the sky and winked at the farmer, "Buzzards!"

Sometimes you are going to need a lot of persistence to solve a problem.

You will try so many things that don't work that your head will be spinning. When you are chasing multiple theories, some experiments will produce no change. Some experiments will show confusing results, because you will see some random variations that have no significance to your problem.

It really helps if you "Love it, Love it, Love it!"

Rule 6: What Changed?

Be sure you know which kind of problem you are solving. Are you working on a design and development problem? Or are you trying to repair something in the field that previously worked? The difference between these is huge. In the first case, you don't actually know that the system has *ever* worked correctly. In the second case, you (mostly) assume that the design was capable or working correctly and that something specific to this system has broken.

Rule 7: Localize Problems.

The most difficult challenge in complex systems is to figure out where the problem really exists. Is it a hardware problem or a software problem? Is it an electrical problem or a mechanical problem?

Be very careful about jumping to cause at this stage. Let's look at a very simple problem, which can quickly become complex.

Suppose you have an electrical push button that is used to tell your system to take some action. You find that every time that button is pressed, the system does the same action two, three, or four times—even though you pressed the switch only once.

Many engineers will quickly recognize this as a common problem called keybounce. Although the human time-scale observation is that you have pushed the button only once, a fast electrical system will see the contacts connect and disconnect several times before settling to the next state.

So what kind of problem is this? It turns out that there are at least five different possible causes:

- **Mechanical:** The keybounce could be a mechanical issue. If the hardware and software are designed for a maximum of 100 milliseconds of bounce (settling) time, and the selected component is specified to meet this requirement, it could be that something in the actuation lever is causing multiple actuations. For example, switches are sometimes used as part of a navigation cluster, with up, down, right, and left buttons. The cluster design might create accidental actuations from pressing adjacent buttons.

- **Electrical:** It could be that the electrical design was intended to de-bounce a 100-millisecond settling time, but the designer mistakenly used too little capacitance in the circuit. Occasionally, multiple pulses could be seen by the software.

- **Software:** In some cases, the hardware and mechanical teams might have specified their components and interfaces such that they expected the software team to de-bounce the incoming signal with no help from the physical implementation. It is possible that the software implementation is not doing a good enough job of removing the key bounces.

- **Component quality:** It is possible that a 100 millisecond switch settling time was specified by the vendor, but simply is not supported by the existing product. The component might not be as good as the vendor claims.

- **System design (interface specification):** This fifth answer is often the most painful. Maybe the problem can be localized, but not to one specific skill area. Rather, it can be localized to the points at which two or more blocks come together. In this case, you can say the fault is a system design or interface specification error.

As an example of this fifth failure, the pushbutton switch might have been originally specified as needing to be better (less) than 100 milliseconds of settling time from an actuation. A switch with a really fast settling time (less than 10 milliseconds) is purchased for prototypes. The software engineer realizes that his code is far too slow and tolerant for these excellent switches and changes the code to give a quicker user response. The hardware engineer does not see any need for extra circuitry for hardware de-bounce. The prototypes are smashingly successful and everybody is happy.

Then the production samples arrive and all of them show multiple keystrokes for every push. After a long and painful investigation, the team finds that a different switch has been purchased for production.

This new switch meets the written design specification. However, the entire design team had created a new (unwritten) expectation of using the better switch component. In a very real sense, the failure occurred because a part that was *too good* was used during development.

Fixing the design might be done several ways:

- **Changing (back) to the better switch:** This would cause increased cost and production delay while new parts are purchased.

- **Adding hardware circuits to remove switch bounce:** This would cause huge schedule delays and require re-certification of much of the hardware design.

- **Changing the software architecture to support a longer de-bounce period:** This will cause a schedule delay and additional cost because a new software approval cycle must be started with an outside third-party.

There might be no good, happy, satisfying answer to this example problem. You may be forced to select a solution that minimizes one kind of pain over the others.

Compliance Engineering

I want to take note of a particular area that creates grief for many projects. In addition to the functional requirements for a system, there are often requirements for certain kinds of stresses or performance of a product that do not directly relate to the (normal) behavior of the product. These might be tests like UL safety, or environmental checks such as drop - tests, acceleration, shock-and-vibration testing, impact testing, radiated or conducted EMI, or radiated or conducted susceptibility. These might also be for compliance with third-party agencies like Dolby Laboratories, HDMI, or USB certification.

Too often, these tests are pushed to the very end of a project. Many times, this delay in testing is because you need to have more of the system development completed before you can execute a given test. The worst aspect is that too many managers regard these as "victory lap" tests that simply demonstrate you have done a great job instead of seeing these tests as "design tests" that help you track the overall quality of your system design.

If you view these as design tests (integral to the process of creating your final product), you will be much more likely to ship a high-quality product, on-time and within budget. If you view these tests as unnecessary hassles where somebody is just trying to make your life difficult, you will find your life quickly becoming difficult and full of hassles.

An Example of a Real-World Complex System Problem

▶▶ *Once upon a time,* there was a complex electronic system that had finally reached the point where it was believed to be ready to build in a factory. The mechanical packaging was solid and had passed all stress testing. The electrical design had passed functional testing on every circuit block and all compliance testing had passed.

The system on a chip (SoC) in this product incorporated at least five different processing cores, all directed by a main processor. The coprocessors communicated with the main processor through interrupts and shared memory mailbox structures.

The software was late (as software always seems to be), but not having 100% perfect software was not viewed by management as being an impediment to running the factory. After all, this was a communications product, where new firmware other than the security boot loader could quickly be downloaded to every sample in the field.

The command was given to start, and the factory began producing hundreds of units per day. The first 100 samples were tested, and a surprising number of them crashed, showing a mostly sky-blue output screen with corrupted characters. Between 3 and 5% of each production group showed this problem. This was far too many failures and would ruin the economics of the already low-margin product.

A team was assembled to investigate. Brian, a hardware manager, was tasked with identifying and fixing the problem.

Samples of failing units were shipped from the factory to the design centers. Tests showed that the defective samples indeed would fail to boot. However, after many attempts, a given box would sometimes boot successfully. That same box would then boot successfully every time, even if booted up hundreds of times over the next few hours. If that same sample sat unpowered overnight, however, it typically would fail to boot up correctly the next morning.

The problem appeared to be extremely intermittent. Bad samples were very likely to continue to behave badly after a long power-off time.

While investigations continued in the design centers, the factory put a new batch of samples through repeated testing. Previously isolated samples were also retested. An alarming pattern was noted. Samples that were previously declared bad would typically continue to show up as bad. However, samples that were previously declared good would sometimes move to the bad column in later testing. With repeated testing, 7% of the units were now listed in the bad column. It appeared that further tests were likely to move even more samples from good to bad. The disaster was growing.

A major complicating factor was that the product's SoC had to be "locked" with special security measures for the production samples. This meant that normal test code could not easily be loaded into the main IC to allow diagnostic code to be run. Only "signed" code could run on the production IC; getting generic debug code signed was a big security risk that nobody was willing to support.

A further complication was that there were many teams working on the software. The SoC vendor provided most of the low-level hardware drivers because they understood the internal functions of all of the hardware cells and coprocessors. The boot code and much of the application layer code came from a third-party vendor. Brian's company had to provide drivers for special hardware on the PCB and to do all of the integration of the various pieces.

Brian was called into his manager's office for a meeting with his boss and his boss's boss. The upper-level boss had a background in software development and managing such projects.

The time required to ship samples around the world and the investigations at multiple design centers had ballooned the delay in production startup. The factory was threatening to dump the entire project. The main customer for this product was threatening to cancel their entire order. The pressure was on.

Brian's boss's boss began the conversation by stating that this was clearly a hardware design problem. His logic was that if the code were bad, given that the exact same code was loaded into every box, they should all either be bad or be good.

Brian said no. This was almost certainly a software problem. Brian's boss rolled his eyes with a look on his face that said, "You and I had better be updating our resumes if you continue down this path."

"But my logic is perfect!" said the boss's boss. "There is no way the same software can be both good and bad just because it is in different boxes."

“Well, there is one specific way that the same software can be the cause of behavior like this,” Brian replied. “I have seen something like this before in a previous job.”

At this point Brian’s boss looked like he wanted to crawl under the table. This argument was headed for nothing but trouble.

“Here is how this can happen,” Brian continued. He pulled out a dry-erase marker and began scribbling little drawings on the whiteboard. “The problem might be uninitialized variables in the code.”

He continued, “The main processor comes along and is finding some kind of value or pointer that was not initialized to a known value. That processor then takes a jump or selects an action based on the value it finds in that location. The result is completely unpredictable. If the DDR DRAM cells for that specific memory location happen to come up in a certain state, then the box will boot successfully. But if they happen to power up with the wrong value, then that box can crash somewhere during the boot process.”

Brian went on, “I have verified with the software team that the basic boot loader does not initialize all of the memory to a known state such as all zeroes during startup. They were worried that such code would take too much time during power on. But they say any code should be setting up the variables before using them, so they don’t like my explanation either.”

The boss’s boss pounced. “I know that DRAM stores values onto capacitors in each cell. All of those millions of little capacitors should discharge over time when the box is off. So the DRAM should be completely set to zero after a long power-off time, and long power-off time is the condition giving us trouble!”

Brian responded, “Yes, I can see why you might think that. However, we are talking about very tiny capacitors and very little charge stored on them. Everything in that IC design is intended to keep the leakage from that capacitor very low. So some cells will have the ability to store some charge for hours or even days.

That charge ends up looking like a one after the DRAM starts up. There is also some random noise that can couple a little extra energy into any given cell during startup. Once the refresh cycle starts running, even a tiny amount of charge might refresh up to ones instead of zeroes."

Brian continued, "There is a quick test that shows this behavior. If you print out the contents of memory before the code runs, you will see a random pattern of ones and zeroes. Every sample will be different. Yes, the general trend is to mostly zeroes, but it is never consistent or perfect."

The boss's boss was still doubtful. They ran through the complete argument cycle a few more times. Eventually, he decided (as good bosses do) to let Brian hang himself with his own argument. There would be time enough later to replace Brian if he could not come up with a good solution. Success would eventually reflect well on all of the bosses, and failure would mean that Brian had made himself an easy target for sacrifice.

In the meantime, Rajesh, a senior programming expert for the company, had been brought in to see if he could capture anything useful from the locked software. His efforts paid off. He was able to report that a failing box consistently reported a specific coprocessor initialization failure just before the system crashed.

Armed with this new information, the team was able to bring in a field applications engineer (FAE) from the SoC vendor. He arrived with a laptop loaded with all of the driver code and the complete code release history. The team briefed the FAE on the symptoms and gave him a detailed problem report, including the latest error uncovered by Rajesh. Before the FAE would even look at their failing sample, he asked for a few minutes to study the release notes.

It was not long before he was smiling. "Here is the deal," the FAE said. "Your software team has to pick a release point for the drivers or they will perpetually be chasing a moving target. They cannot keep re-integrating day after day because they would never have a stable system to test. It just happens that you picked version 9.1, which was released about two months ago. In looking at the release notes, I see that a bug was reported and fixed in version 9.3, about one month ago. The bug was unexpected behavior of the coprocessor system due to *uninitialized locations* in the memory mailbox used to communicate between the processors."

A wave of relief swept over Brian. It took a few hours to build a new version of code that included a specific patch to ensure that all of the memory mailboxes were initialized before starting up the coprocessors. Quick testing and then long-term testing showed that the startup crash had been eliminated.

New code images had to be signed, and all of the development partners had to be updated as to a new schedule. Hundreds of boxes had to be reworked with new code images for the boot loaders. Tens of thousands of boxes were then manufactured with the new firmware. The problem had been solved.

And Brian got to keep his job for a while longer (although nobody lives happily ever after, even in debug stories like these).

Troubleshooting Complex Systems

- **Rule 1: Don’t panic.**
- **Rule 2: Get organized.**
- **Rule 3: Be methodical.**
- **Rule 4: Record your work as you do it.**
- **Rule 5: Be persistent.**
- **Rule 6: Ask what changed.**
- **Rule 7: Localize problems.**

Chapter 17: Intermittent Problems

Intermittent problems are probably the most frustrating and difficult problems we ever tackle. I use the term "intermittent" to mean problems that do not occur continuously or in all instances. Maybe your computer boots up successfully 98 times out of 100 attempts. That means it fails only twice in 100 attempts. You are going to need to do a lot of testing to find that problem. In many ways, the 98 successful attempts are useless to you unless you are doing some very deep captures on the internal operations. Oddly, you must get past the "good" results to find and examine the bad cases.

You usually need to see the failure and capture information about the circumstances of that failure to get any kind of clues about the failure. But when the failure is not happening very often, you don't have much opportunity to gather information or clues.

Reproduce the problem

As discussed in previous chapters, the first thing you need to do is to reproduce the problem. It is important to note here that you must reproduce the problem that has been reported. If you find a bunch of other problems, you might need to fix some of them or you might need to note them and continue testing until you can reproduce the problem you initially were trying to solve.

If you can never reproduce the reported problem, you might need to go back and confirm the initial problem statement. It is possible that the first reports were just badly described, and the problems you are observing really *do* match the intended complaint. But be sure to update the problem statement, or else everything you do that follows won't make sense to anybody else.

Try to Change the Rate of Failure

Let's assume that the problem you observe does match the original description, but the "when" is not often and there is no obvious correlation of the problem to a system variable or an environmental condition.

One fairly obvious plan of attack is to arbitrarily change something and see what happens. Of course, you can be smart about this and choose a factor that already has aroused your suspicions during your brainstorming about what you don't know.

For example, if you feel like the failures seem to happen only on a hot day, but you don't have good data to support this vague belief, then you can run some experiments in which you vary the system temperature. If the failure rate does not change, that guess was not very good—but you now can record this in the things you know: "This system is not failing because of its temperature."

Increasing Failure Rates can be just as Informative as Decreasing

Suppose your previous testing showed a sudden increase in failures as the temperature rises. "Oh no!" You smack your head and exclaim, "We can't get any more heat out of the system!" Don't panic and don't get ahead of yourself here.

Associate the Failure

That increasing failure rate could be great news. You might eventually find that you have just one sensitive component that can be upgraded. You might find that a material specification forgot to include a temperature coefficient. The important thing is that you now have something that is known to be associated with the failure.

Be very careful with intermittent failures that show only a weak correlation to some variable or condition. You might be seeing patterns that you *want* to find instead of finding patterns that are reproducible and verifiable. Write down what you think you are observing and then get a neutral observer to share his opinion.

Try Parallel Testing

You can do parallel testing to increase the number of failures. This can be expensive because you might need a lot of test equipment to support multiple systems. Nonetheless, parallel testing is sometimes the only way to increase the number of failures for an intermittent problem.

Change Only One Variable at a Time

You know this. It is in every basic science text that teaches experimental methods. You still will forget it, however, in moments of weakness, in moments of high management pressure, or maybe from boredom or inattention. Write it on the back of your hand or put up a big sign on the wall. Change only one variable at a time.

Collect More Information for Each Test

One key to quickly solving intermittent problems is to have lots of good recording instruments hooked to the system when the failure occurs. This increases the knowledge you have about the system just after and maybe just before the failure happened. Collect all of the data into a single folder and label it clearly with an identifier for that test cycle. Somebody else can study the failure while you are capturing the next one. (Or, if you prefer, let them do the next test run while you study the data. But be sure that they do the test the same way you did it, or else you will have an added variable.)

The Intermittent Transmitter

⏩ *Once upon a time*, Matthew worked as the chief engineer at a college radio station, maintaining and improving the physical systems while getting his degree as an electrical engineer. He possessed a First Class RadioTelephone Operator's License, a government-issued authorization to control, repair, and modify those transmitting devices that filled the public frequencies with broadcast signals.

After graduation, Matthew took a job with a large corporation that designed and manufactured broadcast radio transmitters. The company had received complaints about a particular installation. This station transmitted just fine all day long and deep into the night, but the transmitter shut itself off at 4:00 p.m. every day.

This was a huge problem, because radio transmitters are often located far from populated areas. This is because such transmitters typically need a lot of land to allow them to have very tall transmitting antennas. Stations use sophisticated remote-control systems to monitor and switch various functions from the control room and studio facility, which is typically located in the heart of the city being served by that station.

Thus, every time the transmitter shut down, somebody had to drive out into the countryside and restart the transmitter. As a result, the station was off the air for nearly an hour whenever this happened. And it was happening far too often—pretty much every day.

The station operators found that the transmitter was tripping a protective circuit breaker, which, for safety reasons, was not accessible by the remote-control system. It was considered important that the system not simply be restarted without a human to make sure that nothing else had overheated or overloaded. The designers of the system did not expect this safety device to activate under normal circumstances. In fact, it had never been observed to activate in any other installations, anywhere or any time. Never, ever. It was one of those protective devices that designers put into a system "just in case."

During every visit to the transmitter, the station engineers found nothing wrong. They simply reset the circuit breaker, and the transmitter would spring back to life, with all measurements and indications showing normal readings. The station could not afford to be off the air every day and could not afford to have a person sitting at the transmitter all day, just to reset the circuit breaker.

The company that made the transmitter received a problem report, and that meant that Matthew got the call.

He asked the station representative if they had ever had a person sitting at the transmitter when the problem occurred. The representative said that indeed they had, but that person had not observed anything before, during, or after the failure. The transmitter was working fine, then suddenly kicked off. It came back as soon as that person reset the breaker, which had taken only a few seconds from the time the circuit breaker tripped.

So Matthew headed for the town where the station was located. When he arrived on a Monday around noon, he picked up an engineer from the station's main office. Together, they drove out to the transmitter site. It was a crisp fall day, with the weather just starting to turn cold in the evenings. But the days were clear, with deep blue skies, and the colors in the trees were magnificent.

During the drive, Matthew was surprised to learn that the transmitter had not shut down the previous afternoon. This could be a real headache, because some intermittent problems can seem to "heal" themselves, making them even more difficult to find.

When they arrived at the transmitter, Matthew did a quick survey of the location. The transmitter was located in a small brick building, not far from the country rock-and-gravel road that they had driven down to reach the site. Occasionally, a pickup truck would scoot down the road, but the total traffic this far from town was very small.

The transmitter installation was neat and well-maintained. The incoming power connections were solid, and the power-source voltage at the transmitter input was steady. There was no indication of excess heating, and the fronts of the transmitter panels were cold or slightly warm to touch. All of the signal measurements along the chain were within specification. All they could do was wait.

They sat in chairs facing the transmitter. The air conditioner and cooling fans in the equipment racks hummed along with the reassuring sound of a rushing river. Eventually, 4:00 p.m. arrived, and the transmitter kept running. Then, at 4:02 p.m., the excitement started.

Out of the corner of his eye, Matthew noticed the needle on the power-output meter seemed to swing upward. Then, with a loud click, the circuit breaker on the exciter tripped. Immediately, a loud alarm bell began to ring. Within seconds, the phone also began to ring as the control room back in town was calling to demand that the transmitter be put back onto the air.

Matthew reached up and hit the Alarm Mute button. The station employee picked up the phone, but said nothing as Matthew suddenly motioned to him with a finger across his lips, hissing “Shhh!” The station employee was mystified as Matthew quickly stepped away from the transmitter and walked over to the window. Craning his neck back and forth as he looked out the window, he nodded and then scratched behind his ear a bit. He quickly stepped back to the transmitter and lifted the circuit-breaker lever, which restarted the transmitter.

The station engineer confirmed with the control room that everything was okay, and then hung up the phone. “What do we do now?” the station engineer asked. “I’m going to take a nap,” Matthew replied.

The station engineer looked like he had been slapped. “What? We have to fix this thing! Now!”

Matthew calmly replied, “Anything we do at this point in time would further disrupt your station’s broadcast schedule. But now that we have a clue to work with, we can wait until signoff at midnight and then we will fix the problem.”

“Clue? What clue? It just did the same thing it did all last week, and you don’t have any additional information! And you ran away from the problem. You weren't even looking at the transmitter!"

Matthew smiled. "I cannot be 100% certain, but I think I know the cause of the event and I think I can guess where to look for the problem. We will need to wait until the station is off the air if we want to debug this down to the specific component. Until then, we can just relax and maybe read over the schematics again."

The station engineer was not happy, but he could understand the logic behind Matthew's choice not to disrupt the station operation any further. Broadcast stations make money by staying on the air and transmitting commercials, so it made a lot of sense to keep the transmitter running during those lucrative programs.

When midnight finally came around, the station engineer was ready to see what magic Matthew might have up his sleeve.

After the final signoff, they manually brought the transmitter back on line and played a brief audio recording, identifying the station frequency and assigned call letters.

Matthew then walked back and forth in front of the transmitter racks. Pausing in front of the exciter, he reached back and delivered a solid strike with his open hand to the face of the system. The circuit breaker two racks over on the transmitter clicked over, and the transmitter was off. Matthew flipped the breaker and repeated the exercise a couple more times. Once or twice, depending on where he struck, the breaker did not trip, but mostly it did.

The station engineer was dutifully impressed. "Okay, now that is cool. You have to tell me what is wrong, and how you figured it out."

Matthew started slowly, but talked faster as he went along. "When we had the event at 4 p.m., I noticed the transmitter power meter ticked upward just as the circuit breaker popped. But more importantly, I realized that I was hearing and feeling something at the same time. It was a very low frequency vibration—a rumble, just something vague that I had not heard before or long after the event."

"But why did you walk away from the transmitter then? Didn't you want to hear what part in the transmitter was making the noise?"

Matthew replied, "No, we never heard any noises like that coming from the transmitter. It had to be coming from outside the building! I looked out the window, and sure enough, there was a huge—no, I should say gigantic—grain truck passing by out on the road. I will bet that truck has been collecting from the fields north of here, and then it comes by fully loaded every afternoon around 4:00 p.m. If I am correct, then we must have something a little bit loose in the system that is changing based on the vibration. Do you remember that the system did not fail yesterday? Well, yesterday was Sunday, and I'll bet they take Sundays off to rest."

The station engineer was catching on now, "So you hit the section of the circuit you thought might be most likely to be sensitive! You were trying to reproduce a vibration failure."

"Yes. The transmitter final amplifier stage is built with very large, heavy, solid components. It seemed less likely to suffer from a vibration-related surge, although the problem might have still been there. But the exciter has lots of tiny, sensitive components, so my first guess was to tap the equipment frame in that area."

Because Matthew and his new friend had a good idea of where to look, they then connected an oscilloscope to the output of the exciter, keeping the output amplifier turned off. They could see the exciter output jump higher when they tapped the frame. They slowly worked their way across the various circuit boards, and then across the assorted components until they could identify the most sensitive area of one circuit board. They found that a particular variable resistor (called a "trimmer") seemed to be the most sensitive spot.

This made a lot of sense to Matthew because trimmer resistors are constructed with a slider mechanism that rubs against another piece of material. The further down the track the slider goes, the higher the resistance. However, if the slider is not built correctly, or if some excess material gets between the slider and the track, then the resistance of the device can jump around in unpredictable ways.

They replaced the trimmer resistor and then reassembled the exciter system. The mysterious sensitivity to external mechanical disruption (hitting it) was now gone.

The station engineer was happy to have been part of the debug process and recited what he had learned back to Matthew. "It seems to me that you fixed the system by doing less, but by observing more carefully. You heard or felt a vibration and then simply looked more widely to see if the problem was internal to the system or could be external. Once you had a clue about a stimulus that could cause the problem, you then waited until an appropriate time to simulate the same kind of stimulus. That stimulation—hitting it, could be repeated until you homed in on the specific area, and finally identified the specific device causing the problem. That is really good debug work!"

"Yes," said Matthew. "And now we can feed back this information to the factory. First, they can test these assemblies with a calibrated tap hammer to be sure that the variable resistors in these circuits are not defective. And then, if our guys are clever enough, maybe they can design the circuit to not require this kind of mechanically sensitive adjustment at all."

The ride back to town seemed a lot happier and much quicker than the ride out to the transmitter had been!

Debugging Intermittent Problems

1. Reproduce the problem.
2. Try to change the rate of failure. Remember: Increasing failure rates can be just as informative as decreasing them.
3. Try to associate the failure with a variable or condition of the system, but watch out for superstition.
4. You can do parallel testing to increase the number of failures. This can be expensive, however.
5. Change only one variable at a time. (Sound familiar?)
6. Collect more information for each test.

Chapter 18: How to Think, Part I

In some ways, this is the most important—yet most difficult—part of this book for me to write. Who in the world am I to tell you how to think? After all, I am pretty sure that you are smarter than I am. Okay, you might not have as much time in the field as I do. And you probably have not performed as many different roles within your skill area. But those last two are not such a big deal.

The bottom line is that I am offering up some wisdom earned the hard way: from painful experience; from finding lots of new and different ways to fail at something. You are welcome to take any of these ideas that fit and use them. You are also welcome to ignore or reject any of these suggestions. But go ahead and read this chapter. If even one single idea helps you, it will have been worth your time.

In every problem-solving effort, there comes a time when you are sure that you have a good solution. You are ready to declare the problem solved and move on to the next challenge. This is the time to be most cautious. It is natural to experience a sudden high after finding the answer. Exuberance fills your brain and can make you ignore warning signs that you normally might see.

Warning Number 1: Correlation is not Causation

If you have ever read the online technical forum Slashdot (http://www.slashdot.org), you probably have encountered instances of the statement "Correlation is not causation." The first few times I encountered this saying, it did not have so much impact. As time passed, however, I began to see dozens, then hundreds of examples of the wisdom of this phrase.

This goes to the very core of science and the scientific method. We believe in a general idea of causation: When *this* happens then *that* outcome will follow. If you hold a ball in the air and open your grasp, the ball will drop to the ground. We explain this with a story about something we call gravity. We say that gravity is the cause of the ball falling.

You can repeat experiments as many times as you want and the outcome will always be the same (assuming you have stated and followed your test conditions accurately). You develop a little story (a theory) that explains the action in terms of cause and effect. As long as your experiments never disprove your theory, you come to accept that theory as being an accurate description of reality.

The theory itself is not a fact; it is just a good story that helps you wrap your mind around the observed behavior. The observed behavior is a fact (or a set of many facts), however—unless somebody is lying about it. The theory is how you explain the observed facts. If (after many attempts) nobody can poke holes in a theory, it's decided that the theory is proven and is therefore true.

Often, you can consistently observe two events as happening near to each other in space or time. We say these things are "correlated." But this does not prove that one event caused the other. Maybe event A caused event B, but maybe event B caused event A. Or there might be some third factor, C, that is not so easy to observe. Of course, there are far more possibilities as the system becomes more complex.

When you only pay attention to correlation, you might start to believe in voodoo. Waving a chicken bone near the server might seem to have a strong correlation with the server not crashing. If you did not run an experiment in which you waved something else, you might not realize that the server's failure to crash is really associated with any additional air movement in the room. In the meantime, employees would be frantically hanging chicken bones near every server.

One step you should take is to come up with other meanings of your solution. Ask and answer questions like, "If *this* is causing *that* to happen, then we should be able to see (measure, capture) *this other effect* at the same time."

Keep this warning in your mind, and look for good examples of it to teach other folks around you: Correlation is not causation.

An appropriate cartoon can be found at: http://xkcd.com/552.

Warning 2: Your Proposed Causation Had Better Show Really Good Correlation.

Having just warned you to remember that correlation is not causation, I have to state that your theory of causation had darn well better show really good correlation with your experiments—like maybe 100%. I mean really, what would you think if Einstein had said, "$e=mc^2$ except on Tuesdays or at some locations in Australia"?

Warning 3: There Might Be More Than One Correct Answer.

Indeed, there might be an infinite number of correct answers, but some might be a little better than others.

Researchers who study the art of problem solving in an academic way talk about problems that are ill-structured as compared to problems that are well-structured.

For example, in a classroom, you might study a problem where a rule is that [F=ma] (force equals mass times acceleration). You are given a force and a resulting acceleration, and you know what mass has been acted upon. This is clearly a well-structured problem. There must be no sliding friction, no air resistance, no deformation of the object, no temperature change, or any other nasty reality intruding into the problem.

But outside of the classroom, you might find all of these side effects plus some constraints in an ill-structured problem. Maybe for safety, the company insists that the acceleration must stay below a certain level. As the designer, you can make the object more frictional, or let it deform more easily.

Suddenly, you find that you have almost unlimited "correct" solutions—although some will be far more expensive, some will take more time to develop, and some will not be cosmetically attractive to the buyer.

You need to find the center. You must achieve a balance that does not improve one thing too much at the expense of another thing.

Warning 4: You Must See If the Problem Comes Back

When you think you have a good story to explain your observations, and you believe you have a fix, then you should undo your fix and see if the problem comes back.

What if you undo your fix and the problem does not come back? Ah, you might be suffering from an intermittent problem, or it could be that your fix actually had nothing to do with the observed problem. In that case, it is time to go back to the beginning of the debug process and begin anew. In product development environments, an alternative test is to apply the same fix to several samples. Does the fix work for all samples?

Warning 5: There Are Many Ways to Find a Good Solution

Avoid thinking that there is only one right path to finding a reasonable solution.

⏩ *Once upon a time*, Harry and Bruce worked at a computer design company. Harry was British with appropriate English attitudes. He had worked and lived in many locations around the world. In contrast, Bruce had never left the United States. Indeed, he had not ventured far from the Midwest. Harry was brilliant and could run through deductions at such velocity as to put Sherlock Holmes to shame. Bruce was methodical and patient—neither brilliant nor stupid, but rather somewhere near normal (whatever that statement might actually signify).

Several prototypes of a new single-board computer had recently been delivered to their company. One sample refused to start up, but made it far enough along during the initial memory configuration to report failing results of a DRAM read-write test.

Harry printed a copy of the onscreen error messages and went back to his desk to study them. Bruce took the sample to a workbench, where he could hook up an oscilloscope and power-cycle the sample. Bruce probed around the processor, one pin at a time, watching the scope waveform produced during each boot cycle. Then he probed the memory controller and repeated the process. Pin after pin was examined, with a reset between each viewing. Finally, he was sure he knew what was wrong and walked back to find Harry.

In the meantime, Harry had been methodically studying the error codes, looking closely at the expected values and reported values for each memory write/read cycle and then comparing them to the memory address being tested.

Their paths intersected in the hallway.

"It's A3 shorted to D1," they exclaimed in unison. "Wait, what did you say?" asked Bruce.

"It's A3 shorted to D1" replied Harry.

Looking down at his sheet, Bruce said "Yep, that's what I found too! I see the same bizarre waveform on both of those pins and every other pin looks normal."

After some inspection under a microscope, they found a suspicious area on the PCB where those two signals ran in parallel for several centimeters. Bruce ran the blade of a very sharp razor knife down the valley between the A3 and D1 PCB tracks, while Harry monitored the resistance between those signals. Suddenly, the resistance jumped from zero to infinity: The track-to-track short circuit had been removed. A tiny finger of stray copper had not been completely etched out between those close tracks, which had created the short circuit.

So who did the debug correctly? Was one method superior to the other? No. Each engineer found a solution using the method with which he was most comfortable. Neither method was more correct; neither method was wrong or bad. Ultimately, both methods uncovered the same defect. In this case, they even took the same length of time to find the problem.

Some engineers will charge ahead with a particular debug path and tell you, "My way will find the problem faster." Sometimes, this is even true. But the warning here is to keep your mind open. Don't fall into the trap of thinking your method is the only valid approach.

Warning 6: During Development, Never Allow a Defect or Problem to Go By Without Documenting and Fixing That Problem

If you see a problem once during development, you are likely to see the same problem in some or all of your production samples. In other words, if you see it once, you will see it again—but at the worst possible time.

There is an old adage about problems. If you find a problem while you are sketching the design, it will cost you about 5¢ to fix it. If you find the problem during the formal documentation, it will cost you $5 to fix it. If you find the problem during prototyping, it will cost you $5,000 to fix it. And if you find the problem after production, it will cost you $5 million to fix it. (You can replace the dollar with any unit you like—RMB, yen, or euros. The general curve and the truth of this advice are universal.)

Very rarely, some problems will occur during prototyping that you can truly attribute to the process or materials used to create those samples. *Maybe* your prototype assemblers are just a little more careless than the normal production workers. (Come on, do you really believe that?) It is far more likely, however, that each instance of a defect represents a weakness in the design. Your design might simply be too difficult to assemble in a production environment. Factory workers cannot be craftsmen, hand-tuning each sample to perfection. The process must be simple, automatic, and error-proof.

Quality experts around the world have taken a phrase from the Japanese and made it universal: pokayoke (ポカヨケ) which means error-proofing. It is a method of thinking about the design of products and production processes to make them such that the product can only be built correctly. If Murphy's Law is "Anything that can go wrong, will go wrong," then pokayoke is the effort to anticipate everything that could go wrong during production and make the design and assembly process incapable of allowing those errors.

In summary, keep a complete and accurate list of every defect found during development. If you want a satisfactory product, you must hunt down and kill each and every defect.

When You Think You Have a Solution, Heed These Warnings:

1. Correlation is not Causation.
2. Your theory of Causation had better show good Correlation in your experiments.
3. There might be more than one correct answer.
4. Undo your fix and check to see if the problem returns.
5. There can be multiple paths to good solutions. Your way is not the only way.
6. Never allow a defect or problem report to go by without documenting and fixing that problem.

Chapter 19: How to Think, Part II

Before you get to put a gold star on your assignment board and declare that you have solved a problem, you need to ask yourself a series of Challenge Questions. The answers to these question might not change anything, but the questions are designed to help you gain confidence in your solution—or else to go back and keep working.

Challenge Question 1: How Did Other People Solve Problems Similar to This One in the Past?

When you are solving problems, this is an excellent question to ask yourself both at the beginning of the process and again later, when you think you have a solution.

I am not saying that you can only select from existing solutions. Indeed, you must not reject a unique or creative solution from other folks just because you have never done it that way before. The concept I am getting at here is that good writers are always good readers. It does not matter if you are talking about mystery novels or engineering designs. Good writers of designs are good readers of other people's designs.

The best engineers are almost obsessive about looking at other designs. They bring an incurable curiosity to every encounter. They constantly want to know how other engineers have solved similar problems. They compare and rate solutions. Of course, there is a natural desire to discover that their own solution has the most advantages, but there is also a complete willingness to learn and to load up their toolboxes with more good ideas.

They spend a lot of time reading tear-down reports. They read current publications (magazines and trade papers) to know what the competition is doing and what new components might help their designs. Furthermore, these best-in-class engineers spend a significant amount of time studying cross-over design. By this I mean that an electrical engineer might be found reading a mechanical-design publication or vice-versa.

By exposing themselves to the limits and challenges of other skills, they deliver better designs within their own skill that help rather than exacerbate the other skill's problems. Because they understand more of the system, they find the center more cleverly. They find better balance in the design.

Good writers of design are always good readers of design. Seriously. I am not joking here.

Challenge Question 2: Could It Ever Have Worked That Way?

You might be surprised how many designs that never worked make it into production or even into the field. Well, in fairness, they probably worked a little on somebody's desk one time—but they never should have made it into a factory, let alone into a consumer's hands.

In addition to this question, there are some follow-up questions you can ask here:

- Does this design really make sense to you?
- Does it seem to defy some basic law of physics?
- Is it possible that the design "works" but is actually operating in some way that you or another designer did not anticipate?

There are lots of examples in every field of this last kind of failure. Sometimes, a key component is removed and the design continues to appear to work, even though it should not. Perhaps it turns out that a signal is taking a path through the design that nobody recognized or understood.

A classic example in high-frequency signals is where a key series component is removed with no effect on the signal-output quality! It often turns out that stray capacitance was coupling some signal from one place to another, going completely around the expected signal path.

Challenge Question 3: Does It Work That Way Now?

It is usually pretty easy to construct an experiment or test to see if your design is working according to your theory of operation. Just as I described removing a key component in the previous section, you can usually perform some small modification that should disrupt the system. If the disruption is not observed or is very different from what you predicted, then the system probably isn't working "that way" now.

Challenge Question 4: Are You Suffering from Optimism Bias?

Sometimes, no matter how hard we try, we all fall into the trap of simply wanting a design to work instead of making that design work. We start interpreting the data in whatever way makes our design look best instead of recognizing upsetting anomalies.

Most companies use some kind of phased development in which products or projects go through a formal process. Gates are placed between each phase to force a design to reach a certain level of maturity before the project moves to the next stage. Often, the level of investment increases exponentially at each phase, so it is very logical for organizations to put these checkpoints in place along the way.

Many larger companies require peer reviews or design reviews with an expanded audience before a product can be released through a development gate. The intention might be noble, but the implementation frequently fails. Too many of the reviewers will have a stake in seeing the product stay on schedule. And independent reviewers are rarely encouraged to be as blunt and honest as they should; remember, it will be their turn next week or next month.

Too many plans and too many projects turn out like the story of the emperor's new clothes: Everybody really *wants* to see how nice the clothes look, but my goodness! That dude is naked!

Find and develop a network of people who are willing to give you honest evaluations. Sometimes, you will need to hold a private review outside official channels. Does your network agree that the problem is solved? Do they think the solution is reasonable and economical? Or are they telling you to go back and keep digging? It is better to hear the truth early, before you are the one standing naked on a stage.

Challenge Question 5: How Complete Are Your Checklists?

You *are* using checklists to verify your designs, yes? If there is the slightest question in your mind regarding the value of checklists, I encourage—no wait, I insist—that you read Atul Gawande's outstanding book, *Checklist Manifesto*. This slim volume tells you everything you need to understand about why checklists are the critical tool for almost any complex activity. This is one of those books that should be in every engineer's personal library.

Big Debug and Little Debug

There is some kind of arbitrary dividing line between a "big debug" and a "little debug" during projects. I like to think the division is based on the number of people or the scale of resources that have to be applied.

Giant disasters like the nuclear-reactor meltdowns at Three Mile Island or Fukushima or the loss of a space shuttle are obviously big-debug events. But there are plenty of smaller problems that can easily qualify as big-debug events.

We might be able to work the definition from the reverse side. A little-debug problem is one that involves only you—or you and maybe one or two other people. There is no involvement by management; the product or project (or company) does not come to a complete halt waiting for results.

In many cases, the personal stakes can be just as high for a little-debug problem as for a big-debug problem. Plus, if you don't solve a little-debug problem, it can quickly become a big-debug problem. The more hidden a problem is and the longer it remains hidden, the more likely you are to find that problem in a spectacular and very unfortunate way.

A long time ago, I worked at Texas Instruments. A senior manager there was quoted in a magazine article as saying that the company did not really mind bad news. Bad news was a daily occurrence, he stated. When management received bad news, they had resources and decisions they could take to deal with the problem. However, he stated, surprises were something different—and absolutely unacceptable. A surprise was defined as bad news, delayed.

Isn't that interesting? The thing that made all the difference was the timing. If you are communicating the problems and constraints you are battling on a timely basis, then management has no room to complain that you have surprised them. But if you hide a problem, you have no excuse when that problem later comes back to bite you.

Let's look at a big-debug story. Later, I'll switch gears and tell a little-debug story.

Big Debug: Global Problem Solving Team

▶▶ *Once upon a time,* Alice faced a serious problem. After a complex hardware/software product had finally reached its first build in the real factory, an explosion of emails documented the surprising failure of about 25% of the first factory samples.

This was a classic global design from a company I'll call "Misguided Corp." The electrical and mechanical designs had been started in an American design center and then passed to a different location, halfway around the world. Certain key blocks of the electrical design had come from a third design center in Europe, which had a long relationship with the system chip vendor.

Misguided Corp had pushed the original designers to transfer the project before anyone involved thought the design was ready and long before most participants thought would be wise. The true motivation was quickly revealed when the company laid off most of the original design team and all of the team at the third design center. Obviously, Misguided Corp was exercising a "save money at all costs" approach to business.

Suddenly, the newest samples of the systems were making it through the first few stages of startup and then crashing. No previous samples had shown this kind of failure.

A global team was hastily organized to investigate the problem. During daytime in the U.S.A., debug took place there, switching to the Asian design center during the night. Conference calls were scheduled daily, but represented a significant burden on both teams because they were 12 hours apart. There was no overlap between normal eight hour workdays. Both teams ended up putting in 16 and 18 hour days to keep communication flowing.

The crashes were found to be happening at different memory locations, during completely different sections of code. Software debug was minimally productive because the code seemed to be failing at random locations. The one common factor was that the failures happened after the code began executing from main memory.

A brief explanation of memory controllers is required at this point in the story. It was not cost effective to embed the main memory itself into the SoC because that would increase the cost of that SoC beyond the cost of the memory devices. The SoC vendor did, however, embed the various timing and interface structures necessary to drive proper signals at the proper times to these external memory ICs. The printed circuit-board layout was developed by the system integrator (Misguided Corp), but had been carefully reviewed by the SoC vendor.

SoC designs are amazing collections of high-speed digital and analog electronics. Process variations mean that some devices will be faster or slower than others. These variations make it extremely difficult to control the signal integrity of the interface between the SoC and the main memory. Timing tolerances for these signals are measured from tens to hundreds of picoseconds—values on the scale of one-tenth of one-billionth of a second.

Different SoC vendors have come up with clever ways to monitor and control the temperatures and process variations of their devices. In this case, a special signal source drove an external precision reference resistor. An on-chip analog-to-digital (A/D) converter turned the resulting voltage across the resistor into an error value that was applied to the timing and drive-strength control circuits for the main memory.

After many long days and nights, additional conference calls were scheduled with the SoC vendor. After receiving samples (via international rush shipments), they were able to confirm that the memory was failing to properly store and deliver data, but they were not able to identify a clear cause.

Experiments were run in multiple design centers to see if there was some kind of temperature or sample-to-sample sensitivity. Some samples showed a slight temperature and voltage dependency, but in general, "good samples" were mostly good and "bad samples" stayed bad independent of temperature, voltage, or other environmental conditions. The outliers in the data were troubling and inconsistent, almost like random noise in the experimental results.

Long multinational conference calls continued. Email summaries and new action items followed each call, but progress slowed to a crawl.

As each new result was acquired, a running progress report was generated. These reports were shared to a large audience, including Alice in the Far East design center.

One day, the American design center reported a breakthrough. They had a limited number of "bad" samples but found they could get those samples to move to "good" if a specific filter capacitor was removed.

Digging through the ECN history and the recollection of the engineers, it was determined that this particular capacitor had been added at the insistence of engineers at the third design center, who had been recognized as the experts on this portion of the design. Unfortunately, those engineers were no longer available to support or explain their design change.

During the many debug experiments, some small weaknesses had been uncovered in the product design. None were sufficient to cause the failures, but could be seen to contribute marginally to the failure point on a few of the failures. Fixes were known for these weaknesses and could be applied to any sample, but they were not observed to completely fix any units. Thus, the discovery that *removing* this "essential" capacitor had fixed the problem on a few samples was a stunning and confusing result. Everybody was struggling to explain this behavior.

Alice asked the SoC vendor to create a software routine that could read the automatic correction value one million times in succession and report a histogram with the number of times each possible value was read. It was a tool nobody had ever needed before, but now seemed quite valuable. This tiny software routine turned out to be the key to understanding the problem.

On samples without the capacitor, the A/D read one or two adjacent values during startup. But samples with the capacitor showed a wide spread of values from the A/D converter. The automatic compensation circuits were therefore picking a correction value that sometimes was far outside the normal range predicted by the temperature and process variations. Clearly, noise was being coupled into the A/D by the addition of the capacitor.

This behavior made no sense, because the intention of the capacitor was to remove noise from the compensation signal.

Finally, Alice joined the discussion with an explanation. She explained that the engineers from the European design center had been rightly concerned with possible noise at that input.

However, their solution turned out to be the problem. The added capacitor had one end tied to ground, but that ground point on the PCB was about a half-inch (12mm) from the ground connections of the SoC.

When the system became more active during boot, the level of electrical noise in the PCB ground plane increased dramatically. The key, Alice explained, was that although the schematics showed various parts connected to ground, there was actually significant voltage variation between one ground point and another.

The ground connection could not be considered ideal in this case. There was enough resistance and inductance between the two nodes such that fast-switching currents in the ground could produce significant voltages across the two points shown in the schematic as sitting at the same voltage.

Under normal circumstances, there was some small capacitance to a local SoC ground inside the A/D circuit. This was enough in combination with the precision external resistor to form a tiny RC filter that could remove noise from the input signal. But when an external capacitor to ground was added, it was acting as a direct path for high-frequency noise from ground at one point on the PCB into the A/D input. That added capacitor was much larger than the input capacitance of the A/D, meaning the input RC filter was swamped by the direct noise injection through the much larger added capacitor. In effect, the added capacitor was bypassing the input resistor and allowing the full ground voltage variation to hit the A/D input. The capacitor was doing the exact opposite of its intended function.

Remember the various design weaknesses mentioned before? Each of these contributed a small amount to an offset from the proper controller correction value. The biggest offset came from the noise. Removing these problems and removing the extra capacitor turned the design from unreliable into a stable platform, which was then manufactured and shipped with quantities approaching a million units.

There is no hero to this story, but there was a villain. Misguided Corp certainly shot themselves in the foot by accelerating their restructuring without regard for the consequences. Each design center contributed greatly to the solution of the problem. (And of course, in some ways, each had contributed to the problem in the design.) Alice finally provided an explanation (a theory) of the failure based on the debug work from all locations. No subsequent testing could invalidate her theory, so it became accepted as the explanation of the root cause.

A custom software tool was critical to the final understanding. Many standard hardware and software tools were also used to collect statistical failure data. Finally, clear communication between the design teams was also essential as debug progressed around the clock at locations around the world.

Now I will turn to a little-debug story where there is no global consequence, no large team, and no big communication challenge. One individual keeps fixing the same problem multiple times—until one day he stops to ask why the problem keeps coming back.

Little Debug: The Never-Ending Hole

⏩ *Once upon a time,* Bill and Sarah were very happy with the new house they had purchased in the suburban community of Marshmallow, Indiana. They had arranged with a builder to construct it to their specifications. Every day after work, Bill came to inspect the site, noting in a little journal the progress made that day. Their excitement was nearly unbounded when they moved into their new home.

After a few months, they had a nice lawn growing in the front yard and backyard. The next spring was when the trouble began. Bill noticed a small depression in the ground near the house when he was cutting the lawn in the backyard. In another part of the backyard, there was a small pile of pea gravel left by the construction crew, so Bill took a shovel-full and filled in the small hole.

Because Bill had seen how the house was made, he knew that the construction team had first dug away a big hole for the basement and the foundation walls. Then, after the basement walls were poured, and the concrete forms had been removed, the piles of dirt were backfilled. So it made sense in Bill's mind that maybe the backfill had been a little too loose. Maybe the ground was simply compacting over time. That just had to be the problem.

Then, a few weeks later, there was the hole again. It was odd, because Bill could not see much of the fill material in the hole. He had expected it to simply be lower down as the ground below it collapsed. He looked carefully around the hole, thinking that maybe the problem was that the fill material was washing down the yard during rain storms. But he could not find a trail of material leading away from the hole, so that could not be the problem.

Then he thought, maybe the fill material was washing down into an underground trench. It could be following a trench line that was dug out for the electrical power leading to the house. Indeed, Bill could see a place where the power line trench had collapsed a little, further back in the yard.

The cycle of fill and disappear continued over the next months, and even into the next year. The power-line trench did not show further collapse, yet the hole continued to appear and grow, no matter how much dirt, sand, or pea gravel Bill put into that space. He was beginning to think it was haunted.

Then Bill decided that something else must be at work here. On a hot Saturday morning during late summer, he decided he had had enough. He was going to uncover and fix this problem, once and for all.

The first thing Bill did was to sit down in the backyard and stare at the freshly opened hole. "Where in the heck is the dirt going?" he asked himself. He studied the hole in the ground, but gained no new understanding.

Bill stared at everything nearby and inventoried what he saw. Virtually every utility other than cold water entered the house near this spot at the back of the building. The cold-water entrance was at the front of the house, so Bill knew he could pretty much ignore that one. Here is what he saw:

- A power meter attached to the house, with a fat gray plastic pipe going down into the ground.
- A white PVC pipe, about five centimeters in diameter, coming out from the house, running down about 30 centimeters before going into the ground.
- A water spigot to connect a hose to the house water system; used to water the grass or trees. (This was a few feet from the hole.)
- An air conditioner outdoor unit with an electrical connection and coolant connections in and out of the wall.
- A telephone company service box with wires going into the wall and a 2.5-centimeter PVC pipe going down into the ground.
- A natural gas meter with a pipe going into the house and one pipe going down into the ground.

The second item started to nag at Bill. He knew the function and general operation for all of the other items, but he was not sure about this one. He went down into the basement of the house and quickly realized the purpose of that pipe. It was the exit pipe of the basement sump-pump system.

A sump pump works something like this: Large (about 10 centimeters in diameter) PVC water pipes (some perforated and some solid) are buried under and around the foundation of the basement. One pipe comes down from the ground at the bottom of a window well. One buried pipe delivers condensation water from the in-house air conditioning/furnace unit. All the pipes end at a large-diameter sump, which is really just a cylindrical opening in the basement floor and the ground beneath that floor. That large opening is about 50 centimeters across and maybe one meter deep. Water accumulates in the sump, especially during rainy periods. The water is eventually pumped up the five-centimeter PVC pipe to get it away from the house. A few times in the past, Bill had heard the sump pump operating after a storm, but not very loudly and only for a short time.

Bill was pretty sure that the sump-pump exit pipe crossed the yard somewhere and eventually reached all the way to the city storm-sewer system at the back edge of the yard. Could it be the cause of the mysterious hole? He decided it was time to do an experiment. If the sump pump was causing the mysterious hole, perhaps Bill could get the pump to run while he was watching. If water from the sump was carrying the dirt and pebbles away, Bill should be able to see some evidence of that, even if it was very weak.

The first step for the experiment was to dig down into the hole, even deeper. Using a small hand tool, Bill slowly removed dirt and rocks until he uncovered a thick plastic rope about one-half meter down from the top of the ground. “Uh-oh, that is the buried main power cable into the house!” thought Bill. It was a good thing he did not dig down there with a sharp shovel, or he might have poked through the insulation and killed himself with electric current! It was a hot summer day, and his hands were sweating. A wet shovel handle was probably conductive enough to do a lot of damage.

Bill continued to dig, but very carefully, to avoid the power line. He switched to a small one-hand spoon-like tool to allow him to go around the power line with less risk of cutting into it.

The next step of the experiment was to put some water into the sump and see if any effect—even if only a tiny one—could be observed in or near the hole in the yard. Carrying buckets of water to the sump was very time-consuming, however. And if the pump started running right away, Bill might miss seeing the effect. So he hit on the idea of running water from a hose down into the pipe from the window well. That water would eventually reach the sump and should fill the sump until the sump pump started running.

He set up Sarah in the basement to watch the sump and listen for the sump pump to start running. "IS IT FILLING?" Bill shouted, so that Sarah could hear him from inside the house.

"YES!" Sarah replied. It had taken some minutes, but water from the hose was reaching the sump through the window-well drain pipe.

"IS IT RUNNING YET?" Bill asked after a while.

"Ngfthmt," was the barely audible reply.

"WHAT?" he yelled back.

"NO, NOT YET," she shouted.

This went on for a good 10 minutes, back and forth. Surely, the water level must be high enough to trigger the sump pump! But nothing happened, and each response was negative.

Nonetheless, Bill was excited. His theory was starting to put together some clues. The hole had always seemed to show up after a rainstorm. But then, rainstorms were frequent here, and correlation is not causation. Bill had checked to see if he could find some kind of trail or fantail of dirt leading away from the hole after a rain, but he had never found so much as a few pebbles or a smear of dirt.

Bill got down on his hands and knees and put his face down at the hole. He was wondering if the pump could be running very quietly. Maybe they just could not hear it over the continuous trickle of water into the sump.

Then Bill heard a muted click and a mechanical whine in the distance. A rumble was almost instantly followed by a roaring whooshing sound. But Bill was a little too preoccupied to congratulate himself on having finally triggered the sump pump. There was indeed an observable effect in the hole; this effect could best be described as a three-to-four meter high spray of water, dirt, pebbles, water, and more water. Bill managed to react quickly enough that the volcano erupted only partially into his face, but a significant portion that made it to maximum height then came showering down all over the rest of Bill and a good portion of the entire backyard.

"IT'S RUNNING," came Sarah's voice from the basement.

After about 30 seconds, the geyser subsided. Our debug hero stopped cursing and laughing a minute or two later. He finally had a clue as to where the dirt was going. It was clear to him that the sump pump could deliver enough force to spray a significant amount of material several meters high and spread across the yard. It was no wonder that he could never find evidence of the missing material. It was distributed across an absurdly large area of the yard.

After drying off, Bill dug down a little further. He finally uncovered some white plastic PVC pipe that was clearly connected to the vertical sump-pump section he could see above ground. As he cleared more dirt, he could finally see a clearer picture of the root cause of the problem. The PVC pipe came out of the wall and descended to a point deep in the ground—far deeper than the power-line burial. At that point, it made a right-angle turn and then ran horizontally across the yard. But the right-angle joint was made by a PVC fitting, and that fitting was neatly fractured at a 45-degree angle.

At some point following construction, the fitting had simply broken. It might have been caused by ground settlement. Or it might have been caused by water in the pipe freezing during an especially cold winter. This last thought seemed unlikely. In this part of the country, the ground would not get below the freezing point of water at such a depth below the surface.

In any case, the news was about to get much worse. Bill was struggling to understand why so much water came out of the break at the joint. Why didn't more of the water simply flow through the pipe? He got out a plumber's snake (a tight coil of steel, used to clear clogs in sewers) and began probing the pipe leading away from the broken joint. The snake could not be pushed in more than about one meter before it hit an obstruction.

Pulling the snake back yielded a small pile of mud and pea fill. Oh no. Bill suddenly knew where a lot of the material he had been shoveling into that hole for a year had ended up: inside the PVC pipe. When the water flow diminished, some material would always fall into the gap of the broken joint. Each subsequent burst of water from the sump pump packed more and more dirt and rock into the drain pipe.

Bill started digging a second hole at the distance from his hole to where the snake stopped. When he reached the PVC pipe, he could tell that it was packed with material simply by tapping on the pipe and listening to the sound. So he dug some more along the length of the pipe, turning the second hole into a long trench. After about two meters, tapping the pipe again produced a hollow echoing sound.

Bill cut the PVC pipe in two places: first where he thought the clog started and then where it sounded like the clog ended. His guesswork was reasonable. The remaining pipe was mostly clear of debris, except for the long clogged section of pipe he removed. That long section was completely packed with mud and pea gravel. A quick trip to the local hardware store procured a replacement section of pipe and some joining fittings, plus PVC glue and pipe-preparation solvent.

He replaced the broken right-angle joint and installed a flexible fitting in the vertical run. The intention was to allow some further settling without fracturing the pipes or right-angle fittings again.

Over the following years, the hole did not come back. But Bill often smiled a little any time it rained.

Challenge Questions:

1. How did other people solve problems similar to this problem in the past?
2. Good writers of design are good readers of other people's designs. Are you a good reader?
3. Could it ever have worked that way?
4. Does it work that way now?
5. Are you suffering from optimism bias?
6. How complete are your checklists?
7. Are you doing big debug or little debug?

Chapter 20:
A Great Idea does not have to work, To be a Great Idea

Long ago and far away, I worked at a broadcast center that used its TV character generator system to store various messages for over-the-air display. This was long before computers were available on every desktop. This character generator allowed the storage of hundreds of unique messages, which made it relatively advanced for that point in time. Like I said, it was a long time ago.

The character generator had become something of a bulletin board for the operators. They used it to store various sayings and messages that had struck them as funny, unusual, or meaningful. In some ways, it was a 1970s version of Twitter. What could a person say that was clever or funny, but still fit completely onto one standard-definition TV screen?

One particular quote was attributed to a senior engineer in the facility: "A great idea does not have to work to be a great idea." Over the years, that statement has always struck me as funny—but I also began to see some wisdom beyond the irony of such a contradictory expression.

At times, people become locked into a single idea. They cannot see any other explanation for a problem or any other solution for it. In short, they become obsessed with their "great idea." Even when all of the evidence is screaming at them, telling them that the great idea is wrong, that it does not work, they cannot let it go. It just seemed to fit so well when they started!

Great ideas can become like religions, where belief outweighs all other considerations. Heretics and their comments are not welcome.

There exists a relatively well-known problem-solving method that uses a so-called fishbone diagram. The method assumes you already know all of the possible causes of a problem when you start. You are supposed to categorize these causes and line them up into nice, neat diagrams. You then design (the fewest possible) experiments; these lead you directly to the correct answer. It’s neat, sweet, and so complete.

My experience tells me that the real root causes and solutions to a complex technical problem are often unknown to the team when they start troubleshooting (but this should not be true when they finish the investigation). This is why I am so emphatic that brainstorming efforts early in the debug process should not talk so much about “why.” Sure, if you want to list some of the potential causes, that is fine—but don’t focus on proving those before you have thoroughly documented all of the other *who, what, when, where,* and *how* facts.

In some ways, I think the possible cause list should be the least organized, most messy, and most casual part of your communication. It is okay to make it difficult for people to lock in on this part. You want to make them focus on what is known instead of what *might* be true.

So here is the thing about getting a great idea: When you find out that your great idea is wrong, you must unwind the investigation and go back to the beginning of the debug process.

This often happens when you are at the very end of the process. You have gathered lots of evidence and you have come to a final experiment that was going to prove that you truly understand the problem. You tried *this* and really expected to see *that* result—but *something totally different* happened.

If you are really clever, you might be able to walk backward through your logic and your previous deductions and actually see which fact has misled you. I have talked about this before. Sometimes, it is not what you don’t know; it’s what you “know” that just is not so! In other words, somewhere in your facts and your deductions from those facts there is some false item. This thing, this falsehood, is leading you down the wrong path.

Like in the scenario with that 24-year-old man in the bar, you started drawing conclusions that made no sense once you knew the real facts. In fact, they led you to exactly the opposite conclusion about the situation.

Although intuition can lead you to a brilliant diagnosis, it can also lie to you. You were so sure that you saw some event, it colored every observation after that. But when you find out that this one observation was wrong, every subsequent conclusion becomes suspicious.

No matter how painful, the best thing to do in this case is to go back to the top of the debug process and ask again: What do you know? Walk through your documentation (you have been keeping good records, haven't you?) and challenge every statement of fact you find there. If you say that a resistor is 100 ohms, have you measured your sample? If you say that a sheet of plastic is 10 millimeters thick, do you have measurements to prove that?

Insight

Once you have really found the source of a problem, you will have some insight into the whole problem. I like this word, "insight." It goes so well with the phrase "If I could see it, I could fix it." Insight is how you "see" things in your brain. You visualize causes and effects. You suddenly see how effects become causes in their own right, looping and connecting in multiple complex ways. Insight lets you connect facts that might otherwise seem unrelated. It is the glue that holds your story together.

Proving your solution is good enough.

When you really understand the problem, when you finally have achieved that insight, you can usually design one experiment that proves the correctness of your understanding and often proves some validity of your solution.

Remember that any given problem might have an infinite number of solutions. Two 100-ohm resistors in series might work just as well as one 200-ohm resistor. Those two resistors might cost a little more than the one-resistor solution. But the single resistor might fail an ESD test because a high voltage can jump across the lead space of that single resistor—but not across the lead spacing of two resistors.

Maybe you will need to devise more than one test to prove your solution. In some cases, you have to run a product through an entire suite of tests. It can be very frustrating to discover that your solution fixes every aspect of the original problem, but that your new design has exposed a previously hidden problem.

That is the life of the engineer. Like peeling the layers of an onion, you will shed a few tears and then keep going. Many engineers are familiar with this analogy of problems being like the layers of an onion. This is a good time to discuss the possibility that any debug effort might be chasing multiple problems at the same time.

Sometimes, you will encounter a situation where multiple problems are creating the symptoms you have observed. It will not be sufficient to fix only one of the problems. You must keep debugging until you have fixed all of the problems.

Occasionally, you will also encounter a situation where two (or more) problems have no relationship to each other. You just happened to find one problem while looking for a different problem. It is important to keep track of the unexpected problem. You will need to fix it eventually, so maybe you should go ahead and fix it immediately. No matter what, make sure you record the existence of that second bug.

Failing Gracefully

Sometimes there are good reasons that you will not solve a particular problem. If your boss comes in on Friday and demands a "small change" that requires a million lines of new code, you won't have it ready on Monday. You cannot deliver $50,000 worth of hard-steel tooling for $50, no matter how smart you are.

There will be times that the correct solution is simply beyond the capability of today's technology. What we fit into a handheld package today previously required a room full of six-feet-tall racks. Understanding the nature of the problem does not mean that you can always fix that problem today.

As discussed, there are constraints of people, time, money, and results. You might find that your problem is not worth solving given those constraints. For example, it would not be worth putting $5,000 of repair parts into an old printer that can be replaced today for $500. Even though your problem-solving skills might be beautifully demonstrated by doing this repair, it just is not worth the time and expense to do so.

You should not be angry with management for telling you to stop. These emotions are natural, but you will gain more by keeping good records of your debug and understanding why the constraints kept you from winning this battle. Remember, the problem-solving war always continues.

Once in a while, you might find that you have a different kind of constraint keeping you from solving a problem. It might be that you do not have the physical or hand skills to do something. For example, I could practice all day, but I will never have the hand skills to draw like Leonardo Da Vinci. My soldering skills are not terrible, but clearly not as good as some other folks', who have trained and practiced that art. These are limitations that I must understand and accept.

If the economic or time constraints do not prohibit it, I might be able to hire somebody to do an artistic drawing for me. I might be able to purchase a drawing or make a suitable photograph instead. But if the only solution is to have the hand skills, I am out of luck.

When an investigation is terminated due to constraints or by management decision, don't throw away all of your documentation. You might encounter a similar problem later. I have been in situations where management issued a stop order and later changed their minds. The problem solving had to start up again—quickly. You might even need that documentation to show your manager (or his replacement) that you made an honest and appropriate effort—that stopping was not your choice.

Okay, some of that sounds like political advice and CYA* stuff, but there are clearly plenty of good reasons to generate good records during a debug—and to keep them long after the problem solving is complete, whether by success or by failure.

*CYA is an acronym for Cover Your Ass. It means taking an action that is intended to protect you in the future from some possible negative outcome.

A Great Idea Doesn't Have to Work, to Be a Great Idea

1. **Be careful of Grand Plans and Grand Ideas—it is too easy to fall in love with a solution that you later find does not or cannot work.**
2. **Watch out for multi-layer, or multiple-cause problems.**
3. **Understand your limitations.**
4. **Good documentation helps you succeed and protects you in failure.**

Chapter 21: Study Your Mistakes

So here is a somewhat painful piece of advice: Study your mistakes—even more than your successes. It turns out this is really easy to say, and really, really difficult to do.

There is something in the human condition that makes us want to forget our mistakes—to erase the bad stuff and remember only the good stuff. So this advice comes with some modification: Study your mistakes, but only until you have learned the lesson well enough to not repeat that mistake. Once you have figured out the mistake, put it into your checklists, make sure you understand it, and then stop worrying about it.

In addition to my regular work, I also do some part-time photojournalism. I often find myself failing to get a shot or getting a technically awful shot at a critical moment—or sometimes shooting better photographs, but at unimportant moments. I was determined to do better, but the question remained: How could I improve? The camera would still be the same, but somehow I needed to adjust the nut behind the viewfinder.

Digital photography gives you the opportunity to sneak a quick look at your images (on the back-of-camera display) just after taking them. Sports photographers call this chimping. It got that name because someone observed that people looked and sounded a bit like a chimpanzee when a great photo showed up on the camera-back display as they were reviewing them: "Ooh, ooh, ooh, ooh!"

There is a great temptation to immediately delete the photos that are obviously bad: poorly exposed, out of focus, blurred due to camera shake, wrong color-balance settings, etc. Any photographer can instantly tell that those shots will never get printed in a newspaper or magazine or even used in a Web collection. Nonetheless, I realized early on that keeping these shots was more valuable than the cost of computer storage they required.

My method is to sort the bad shots quickly into a folder called

"Bad" and mark or segregate the better shots so that I can later annotate and process them (e.g., crop them, adjust the contrast, add sharpening) for final use. After the time pressure (deadline) to deliver a few good shots with captions has passed, there is plenty of time to go back and study the bad shots.

Modern digital cameras include EXIF (exposure information) metadata to help people understand what the camera *thought* it was doing when it took the photo. Many cameras include extensive information about the focus point and focus mode used, in addition to the normal ISO sensitivity, shutter speed, and aperture values. In a very real sense, the EXIF data acts like a flight data recorder for photographers. It tells them exactly what the settings were at the moment the exposure happened.

By studying the EXIF file and the resulting image, I have found that most of the time, I can identify a basic mistake I made in the setup or handling of the camera. In some cases, it becomes obvious that the automatic functions of the camera were deceived by the conditions. Maybe an unimportant but very bright object unexpectedly entered the scene, which caused the camera to underexpose the total image. Most of the time though, the problem is that I simply did not think about all of the aspects that would affect the image capture. In other words, I made a mistake. Or several mistakes.

As discussed, checklists can be tremendously helpful for situations like this. I typically keep a small, laminated 10-point checklist for camera setup when shooting.

Many companies include formal process steps to try to study their mistakes. They might call these reviews something like "lessons learned." In theory these should guide a development team to avoid repeating the same mistakes. Unfortunately, far too many companies immediately file such reports away and never look at them again. But in truth, the "lessons learned" review of previous projects should be the first thing you look at when starting a new project. Make sure the mistakes you made previously, don't bite you again on your new project.

If only it were so simple, eh? Unfortunately, the worst projects are often followed by management reorganizations. Punish the innocent, promote the guilty, and all of that. There is a secondary effect here, too. Your checklists can become overloaded with items. Sometimes you will look at a checklist that came from somewhere else and you will say, “I would never make such a stupid mistake; why do they ask me to verify *that* thing?”

Likewise, somebody seeking to improve a checklist (hoping to eliminate all problems) might start asking insanely vague questions like, “Have you checked every conceivable factor?” If you answer yes, you are probably arrogant and overconfident. But if you answer no, you look lazy or careless. The question becomes a trap with no right answer.

The correct solution is that you must actually care about generating a good checklist. Make sure you cover the problems you have seen or created in the past. Make sure you have specific actions with understandable guidelines for your checklist items. Take the time to document what your checklist means and how you run specific tests. Write down when specific tests can be skipped. More than anything, have some peer reviews and be sure to treat your checklists as living documents that can be either expanded or trimmed when appropriate.

If you are a manager, make sure that you do open and honest “lessons learned” reviews about management culpability for project mistakes. For example:

- Did you take six weeks to choose a key component vendor and then tell the engineering team that they were late—before they even started?
- Did your management team cripple the development with absurd constraints?
- Did management set a schedule that the team said could never be met—but management had already sold the customer on such a ridiculous delivery?
- Did you price the product without even asking engineering

> for an estimate? And then did you insist that engineering give you a solid-gold product for the price of dirt and rocks?

There is a bigger lesson here than just abusing people for their mistakes. The goal is to find methods (I mean good ways that work, not stupid things that nobody could possibly do) to prevent such errors in the future.

Study Other People's Mistakes

"*You are idiots to believe that you can learn something from your experience, I prefer to learn from the mistakes of others to avoid my own mistakes.*" Otto von Bismarck (1815-1898,) Prussian Chancellor. [Various Internet sources, none I have found with absolute traceability.]

Good old Otto, who could not have been tagged as a comedian for sayings such as this one, must be acknowledged as having made a decent point: it is certainly less painful to learn from the mistakes of *other* people.

Therefore, in addition to recommending that you study your own mistakes it is certainly appropriate here to recommend that you study other people's mistakes. That is not always so easy to do. Autobiographies and biographies often wish to extoll the achievements and excellence of their subjects. Avoiding or minimizing painful failure is understandable and forgivable.

Nonetheless, there exist collections of stories and reports of famous engineering or design failures.

There are some excellent examples of open discussion of errors for you to study. One is the medical field, in which M&M (morbidity and mortality) conferences are used to openly discuss mistakes. Punishment is not part of this process, but prevention of future mistakes is a primary goal. Likewise, the Food and Drug Administration encourages reporting of adverse medical outcomes that might (or might-not) lead to action against a given provider. The primary intention is to get the information out into the open.

A similar example comes from the field of aviation, specifically air-crash investigations. I have always been fascinated by reading about the process by which investigators uncovered the chain of events that led to an air disaster. These parables inform us, humble us, and ultimately can make us smarter designers.

Some of the investigations read like mystery tales. Subtle clues are collected and initial theories are painstakingly disproven until a final set of conclusions can be reached. Indeed, a few crash investigations have been made into great novels.

In any case, it is certainly good practice to read about other folks designs and debug methods. If the failures of others lead you to avoid even one mistake, the effort to read about them was worthwhile.

You don't want to be an engineer who buries his mistakes.

Problems Created During Debug

There is another kind of problem to remember and study. Once in a while, when you are trying to diagnose a problem, you will accidentally create a new problem. Maybe you strip a bolt while trying to remove it. Maybe you accidentally leave a little extra solder on a printed circuit board and create a short-circuit between two points.

This goes straight to the heart of my previous discussion about sometimes having two problems that are contributing symptoms. In this case, you have now added a whole new problem on top of the problem you were originally trying to solve. Remember to watch for changes in the problem symptoms. Of course, many times you will be completely aware that you just created a new problem.

Study Your Mistakes

1. **Study your mistakes, since they will teach you more than your successes.**
2. **Once you have learned the lessons, put them into your checklists and then stop obsessing about them.**
3. **Management has an obligation to be honest about their contributions to problems.**
4. **Openly discuss mistakes without retribution.**
5. **Study the mistakes of others too.**
6. **Sometimes during debug, you will create new problems.**

Chapter 22:
Business (and the Business of Solving Problems)

Many years ago, my friend Hugh Macdonald-Smith told me that the reward for solving a customer's problem was the opportunity to solve that customer's next problem. At the time, this statement did not seem so profound. As I thought about it over the next few years, however, it started to make more and more sense to me.

Another friend, Jon Fasig, once said to me, "If you find the source of a customer's pain and then remove his pain, that customer will reward you handsomely." I wondered a bit about that phrase, "reward you handsomely." It is an idiom that generally means that someone will give you lots of money. But the meaning of the phrase can be taken in a more general context, too. Even if that customer is not throwing cash at you, he is probably continuing to do business with you. More importantly, if you remove one customer's pain, there are probably many more like him. The reward is multiplied as you sell the same (or similar) solution to many customers.

Both of these comments go directly to the heart of the vendor-customer relationship. Customers are looking for a product or service that improves their current situation. They want you to make their life better in some way.

Often, you solve problems with your own designs because you need to solve that problem to make your product better (cheaper, faster, smarter…better). On the other hand, you might need to solve this problem just to get your product to be minimally acceptable. And in some cases, you are asked to modify your product with the intention of solving a specific problem that the customer presents.

Although the motivation for each of these is slightly different, the ultimate need is still just to solve a problem. Therefore, the problem deserves your full attention and best solving methods, no matter the reason you started down the debug path.

Many years after first hearing them, I condensed these wise words from my friends: The reward for solving problems is the chance to solve more problems. These new problems will probably be bigger and more complex.

There is an implied ceiling here: the Peter Principle. This comes from a book by the same name published in 1969, which stated that people rise to the level of their incompetence. Looking through the lens of solving problems, it predicts that we rise in an organization to the level at which we are no longer able to efficiently solve problems.

Problem solvers need care and feeding. Just as sales people need recharging due to rejection, problem solvers need recharging due to frustration. Engineers are trained to solve well-structured problems, but are not so comfortable with ill-structured problems. When confronted with a seemingly endless series of problems, it is natural for a human being to become fatigued. Yet engineers often persist and find great joy in that "Yes!" moment when a solution becomes evident and that "Yes, Yes!" process when the solution is validated.

The storm cloud on this landscape comes from that layer of additional requirements discussed before, called constraints. Perhaps the technical solutions are clear, but out of reach of the economic structure of the project.

"Rats!" I shout at the emptiness.

Perhaps the time frame allowed is simply not sufficient to create a new or altered design.

"Scoundrels!" I cry.

The worst problems an organization faces are those that are self-inflicted. The list is endless, but you can probably think of a few from your own experience.

Perhaps management demanded the organization switch tools in the middle of development—new computer aided design (CAD), timekeeping, email, or status-reporting mechanisms, to name just a few that I have encountered.

Some organizations will introduce several new systems simultaneously in an effort to "optimize" an organization that was previously functioning quite well. Managers are re-assigned and groups are realigned. Predictably, chaos ensues; management then demands a new round of new systems to recover to previous levels of operational performance.

"Clowns!" I lament.

Problem solvers wear down. They need to be carefully cultivated. You must feed and water them and treat them with kindness, or they will run (not walk) to your competitors. Some will retreat like turtles into their protective shell. A few might actively scheme to hurt your organization.

Organizations and managers must recognize when the biggest problems facing the problem solver have been inflicted from above. Some managers behave like passengers in the back of a vehicle: napping until an obstacle is encountered and then bellowing forth instructions and guidance from their vast intellect and experience to tell the idiot employees how to clear the road. Better managers scout the path ahead for obstacles and do as much of the heavy lifting as they are able, clearing the path and marking the craters to help keep the vehicle moving.

A recurring theme in many quality-improvement programs is to recognize that every problem is an opportunity for improvement. What ways could you put that opportunity to work?

You could make that product better before you even begin the design—at the product-definition stage. You could make that product better at the design stage. Or you might make your process for building that product better. The important part is to see every problem-solving process as an opportunity—not as a setback.

Aptitude and Attitude

Ultimately, your ability to solve a problem, despite the support or interference of your organization, will come down to two of your strengths or weaknesses:

- **Your aptitude:** Aptitude refers to your skills and craftsmanship. Your preparation to win will help (or hurt) you now that game day has arrived, with your practice, experience, and talent either being ready or lacking. Your ability to organize the problem-solving effort is part of aptitude, as is your ability to clearly communicate the problem solving.
- **Your attitude:** This is all about your mental status. Will you show persistence or give up quickly? Are you tireless in attacking the problem or are you exhausted after the first hours?

Often, the only factor that you are fully in control of will be your attitude. You will need to find some tricks to keep a positive attitude. These might include the following:

- Having a mentor or buddy who can build you up
- Setting personal goals and a reward structure
- Setting realistic schedules and achieving them

Working in Organizations Large and Small

There are some differences between working in larger organizations and smaller ones. All organizations, even tiny one-person companies, require some amount of communication and cooperation. If nothing else, there is always the need for clear communication and cooperation with your customer and vendors.

Larger organizations add layers of management and often demand more formal reporting of periodic status. In these larger structures, more emphasis is put on command and control in addition to communication and cooperation.

In larger organizations, the people at the top may become more isolated from the day-to-day activity and often feel too distant from critical operations. They might become more dictatorial, demanding that their whims and orders be followed precisely, regardless of whether they make any sense at the front lines. Any further problems exacerbate this feeling of helplessness at the top and can lead to micro-management of all activities.

Small businesses sometimes add a layer of family conflict as a constraint against problem solving. If Fred cannot tolerate talking to his wife Ethyl, who shares all of her complaints with her friend Lucy, who demands that her husband, Ricky, fire Fred, who Ricky needs as his technical expert—well, you have some big problems.

No management system can overcome personal problems in the interaction of family members. They can leave work at the end of each day, but they don't leave the family at the same time. This can make communication more difficult in a smaller company, even though logic would tell you that the smaller organization should allow easier, faster, and simpler communication.

Bigger organizations often take on bigger tasks. They will need more effective communication systems to get these tasks completed. Think of it this way: If I hand you a single brick and ask you to put it on top of a nearby hill, you can probably complete that task in a few minutes. In fact, you will probably do an excellent job. But if I ask you to move an entire brick house to the top of a distant mountain, you face a much bigger, more difficult task. You could take one brick at a time across the distance, tearing down and rebuilding the house with thousands of trips between the two locations. The transit time is multiplied, and there might be pieces you cannot carry by yourself.

Imagine for a moment that the bigger organization knows nothing about special tools, but they have lots of people available to help. They position thousands of employees around this building and instruct everybody to carefully lift their small section on the count of three. Let's assume that by some miracle, this works;

the building is lifted up in air.

Now, the boss tells them, "When I count to three, everybody must take one step to the left." He shouts "One! Two! Three!" Inevitably, the house will spin to the ground, because the people on each side went to their left, which was to the right of the people on the opposite side of the house. The building is ripped from everybody's grip. Clearly, precision in communications, coordination, and cooperation would be really, really important in this operation.

Big companies are like that. Everything has to be clear, precise, and well-planned if the project is going to get anywhere.

Don't confuse being the best-managed company with being the most-managed company. Best-managed companies know how to trust their people and run operations with a confident and light touch. Best-managed companies put a lot of time into hiring the best employees. Once they have those employees, the best-managed companies put some effort into keeping those workers happy. Most of all, they empower those employees to take positive actions that work for the benefit of the company—and for the benefit of the employee.

Most-managed companies try to control everything from the top. They disempower employees and try to substitute an unyielding rulebook in place of trusting employees to think. They see employees as easily-replaced interchangeable components.

I want to close this chapter with one of my favorite George Rostky editorials from *Electronic Design*, first published December 6, 1977 and reproduced on the next page.[4]

The Shell Race

Charlie knew his company had severe morale problems. They affected everything. The reject rate on the factory floor seemed to get worse every week. His people in the field couldn't sell. And his engineers had lost their flair. Even the newer engineers had lost the spark and enthusiasm they had brought with them when they joined the company.

Everybody wants to work for a winning company but, it seems, everybody felt that Charlie's company was a loser. So there was a general malaise.

Charlie recognized the importance of morale, so he worried about it a lot. What to do? Finally, he got the brainstorm: competitive sports. He would organize some sports competition against representatives of all the other companies in the area.

Everybody loved the idea. The sport they chose, since they lived in a riverside community, was shell racing—a fine competitive sport requiring skill and teamwork. Teamwork! That's what builds morale.

The big day came and the river bank was mobbed. Thousands upon thousands of sons and daughters, uncles and aunts, sisters and brothers, wives and mistresses crowded the shores, screaming encouragement to their favorites.

When the first boat came to the finish line, alas, it wasn't Charlie's. Nor was the second or third boat, which came close behind. In fact, long after all the other boats had crossed the finish line, Charlie's still was not to be seen. When it finally appeared it was zigging and zagging toward the finish line. As the shell came closer, Charlie could see activity on the shell more clearly. Eight executives were shouting orders to one man with an oar.

Charlie fired the oarsman.

[4]*Electronic Design*, page 53, December 6, 1977. Reprinted by permission of Electronic Design magazine, all rights reserved to Penton Publishing.

Business (and the Business of Solving Problems)

1. The reward for solving problems is the opportunity to solve more problems. These new problems will be increasingly difficult. Don't panic. Your skills should also be increasing.
2. The best organizations understand how to enhance your ability to solve problems. They actively remove roadblocks and help you when and where they are able.
3. Some organizations decide you need more managers.

Chapter 23: Customers and Customer Service

Many years ago, an electronics magazine ran a very short piece about the question "What do you need to start a business?" They cited many common answers from engineers, business types, and financial experts. It is a great exercise, and I encourage you to jot down a couple of quick answers yourself before moving down this page.

Technical-type folks will often answer that the key to starting a new business is to have the right idea. Technical-type folks with a little more experience will modify this answer to be, "You must have the right *product*"—meaning that you have to have taken that great idea further and turned it into something more complete.

Business-type folks will typically tell you that having the right management or sales team is the real key to starting a new business.

Financial-type folks might emphasize having adequate financing, accurate cash-flow projections, and good financial controls, because nothing kills a business like running out of money at some critical time.

Oddly, this article said, entrepreneurs all gave the same answer when asked "What do you need to start a business?"

What was their answer? It was simply: "Customers."

I have repeated this story to as many young engineers and hopeful business owners as I could over the years. What I've realized, however, is that the article was incomplete. There is an additional piece of wisdom that we can add here: What is the definition of a customer?

A *customer* is a person or organization that is ready, willing, and able to purchase a good or service from your company for (enough) money to sustain your business.

Although that looks simple on the surface, the "ready," "willing," and "able" components make it far more challenging than you might think:

- **Ready:** One of the first things a good salesperson finds out is where the person or business sits in the process of moving from "thinking about it" to "ready to make a purchase." The effort to close a deal is wasted too early in the process but is essential to getting a sale at the end of the process. The trick is in knowing where you are along that timeline.
- **Willing:** The person must want to part with his own retained earnings (cash) in exchange for something you are offering. I hated to pay so much money for shingles to cover the roof of my house, but it was better than letting the rain come in and ruin everything inside. You can be willing, even if it is an unhappy desire.
- **Able:** The person must have the resources to pay you for your good or service. If he cannot pay you until next Tuesday, he is not able (and not ready). If purchasing your product means he will not eat for the next month, he is not really able. Social responsibility says that you must not move forward with such a transaction.

Here is another way to look at ready, willing, and able: They are closely related to the three features of all projects—time, money, and results.

- **Ready = timing or time**: You probably will not get a sale when it is too early and you will never get a sale when it is too late.
- **Willing = results**: The customer is looking for some specific result. Can your good or service deliver that result?
- **Able = money**: No matter how much the customer wants what you have, if he is not able to pay you (enough) money, there is no viable business here.

That is the second time I have mentioned the word "enough" with respect to money. Let's be very clear here: You can sell personal computers or cell phones for $1, but you will quickly go broke if those items cost you $500 to build or buy. You are giving away $499 with every sale. The customer-to-vendor relationship is false in this case.

The return on every sale must add up to enough money to keep you in business. You have to pay your rent, your utilities, your taxes, and for the cost of your materials. You also have to pay your employees.

All of this is paid from the difference between what your good or service cost you and the price at which you sell it. Your gross profit margin is not a dirty word. It is how you keep going. If you are consistently quoting work at a price below your cost, you do not have a business. Delaying the hard decisions will not make them any easier; in fact, it will make the final result more painful.

Investors versus Customers

Over the years, I have witnessed many smart business people fall into a trap when they became desperate to keep a struggling business alive: They turn all of their focus to finding investors.

Here is one of the most important things I can teach you: There is a giant difference between customers and investors. If you do most things right, a customer will not want his money back. In fact, the better you do things, the more likely that customer will be to come back to try to give you more of his money for more of your product or service.

In contrast, even if you do *everything* right, an investor wants all of his money back. And he also wants some of your money on top of that.

Is that clear enough?

Customers, Prospects, and Suspects

I don't know what it is, but some companies are utterly unable to distinguish between customers, prospects, and suspects. They expend insane resources chasing prospects and suspects, while paying customers are ignored or treated like dirt.

I have already clearly defined what makes a customer. A customer is a person or organization that is ready, willing, and able to buy your good or service for (enough) money to sustain you.

A prospect is a person or organization that could become a customer at some time in the future. A prospect might not have enough money today, or might not be ready or willing (yet), but is somewhere on that continuum between "just thinking about it" and "ready to buy." It is the job of marketing and sales to help the prospect move down that path.

A suspect is somebody who recently was or probably will be involved in a crime, very soon. It might be an offense against your company or, if you are lucky, something involving your biggest competitor. At best, suspects consume ridiculous amounts of time and effort. They never, ever return a dime. They dither forever while demanding your full attention during the decision-making process. A good salesperson can smell these guys a mile away. A desperate salesperson makes suspects into his full-time passion.

A good customer (or prospect) often knows your business better than you do, but respects your abilities and efforts. Good customers realize they cannot do everything themselves and they like to purchase from somebody who is willing to make the commitment necessary to deliver the highest quality at an acceptable price.

You should treat customers best, prospects fairly, and suspects with appropriate caution. For some reason, many companies cannot understand this relationship.

⏩ *Once upon a time,* a man died suddenly and found himself at the entrance to heaven. He was greeted warmly and was given a tour of the place. There were choirs singing praises and people floating on clouds, all with a calm air of contentment.

The new arrival was surprised when he was asked if he would like to visit hell. "I always thought we were sent to one place or the other!" he exclaimed.

"No," said his guide. "We show you both places and then you are allowed to choose your final location."

When they got to hell, the man was surprised. It seemed that a damn fine party was perpetually in progress. People were dancing, feasting, and drinking. Many beautiful women were scantily dressed and pressing close up to the men, who all appeared healthy and vigorous.

At the end of the visit, the man returned to heaven and was asked to choose.

"I don't wish to appear ungrateful, but the other place seems so enticing and heaven seems, well, downright boring by comparison. I will take the other place."

"As you wish," replied his guide.

The man was escorted back down and placed in a room to sleep. When he woke up, he was thrown into a pit of unspeakable slime. Brimstone rained down upon him and fire belched forth from the walls. Demons tormented him with sharp spears and salt was poured upon the wounds. Thirst and hunger wracked him. The devil laughed endlessly.

"What is this?" the man cried in agony. "Yesterday this was completely different! Why have you done this to me?"

The devil replied, "Yesterday, you were a prospect; today, you are a customer."

Is this how your company treats customers? If the answer is yes, you are not alone. You might be thinking that you have no impact on this process. You might feel that your little corner of the business could not change this. You would be wrong. Indeed, you might be surprised at how much you can help.

Do you remember the first steps discussed in solving problems? Clear communication is the key. The better you do at communicating technical issues clearly and without anger, the faster the customer will realize that you are trying to help. The same clear summaries will help other folks within your organization help you to find out the things you don’t know.

When you become aware of problems, use the skills and methods you have learned to put together clear descriptions. You will need the customer to confirm or modify your understanding; you will need to reproduce the customer’s problem report; and you will need to come up with good solutions.

In some cases, you will not be able to discuss all of these solutions directly with the customer because the customer might fixate on a solution that is completely against your company’s best interests. Nonetheless, you must be prepared to discuss all of the options within your organization so that an appropriate solution can be proposed to the customer.

Sometimes you are the vendor and sometimes you are the customer. Try to keep that balance in mind when you are working through customer problems. Are you treating the customer the way you would want to be treated if you were in his shoes? Are you demanding something from your vendors that your company would not be willing to do?

Learn from your customers. Learn from your vendors. Listen to them and develop some empathy for their situation. In turn, give them better problem reports or teach them to give you better problem reports. Use all of the tools available—such as video, photos, sketches, graphs, raw data, and clear descriptions—to avoid bad communication. Make sure you share a common vocabulary. Publish what you know and ask good questions. Find the center of the problem and seek balance in your solutions.

Remember, customers are paying your salary. Are you giving them a fair deal? When you solve a problem for one customer, that person's peers will often hear about it. Your success with one customer will cause other customers and some prospects to ask you to do the same for them.

Abusive Customers (or Vendors or Management)

Occasionally, you will run into an enterprise or individual in an organization who is simply abusive. Realizing that he is in a position of power, the person may be verbally intimidating. He may show no respect for the individuals trying to solve his problems.

Some expect absurd levels of service in exchange for too-low pricing. Sometimes, these enterprises or individuals are in monopolistic or near-monopoly positions within their own market. Companies too weak and too desperate to say "no" will line up to get into the cash-flow stream of these companies.

You do not have to accept abusive treatment. You can tell the individual or organization that their behavior is not acceptable. You can find a new job or business if it bothers you too much.

Every day is a series of choices. Sometimes, one choice (like walking away from a paying, yet abusive situation) might seem impossible. The choice is still there; it's the consequences of exercising that choice that make you claim you have no options. You are stuck between the proverbial rock and hard place.

But you do have choices, even if you don't like them. Rest assured that there is indeed a special circle of hell reserved for those who abuse other human beings, whether those victims are customers, vendors, or employees.

Know the Difference Between Customers, Prospects, and Suspects

A *customer* is a person or organization that is ready, willing, and able to purchase a good or service from your company for enough money to sustain your business.

A prospect is a person or organization that could become a customer at some time in the future.

A suspect is somebody who recently was or probably will be involved in a crime, very soon.

Know who pays your bills and keeps the lights on.

Hint: Your customers pay your salary.

- You will get 80% of your income from 20% of your customers.
- Never treat anybody with disrespect.
- Don't let people treat you with disrespect.

Chapter 24: Final Thoughts

A summary of the main points in this book follows at the end of this chapter. But more importantly, this is the chapter where my innermost thoughts can be shared openly and honestly.

This is not to say that I have been untruthful in the previous pages. Indeed, if you look there, you will find the real me exposed in many ways. Sometimes, though, I felt like I had to keep under control. A writer should not undermine his own position in his work.

If you have made it this far through the book, you deserve some straight talk. What follows is what I would say to you, person to person, were we sitting together in an excellent restaurant following a great feast.

I am troubled greatly by people who dive deeply into the religion of any problem-solving method. Please don't do that. This book simply teaches an approach to solving problems that has worked well in my life and for many other folks with whom I have worked. I try to learn from the best.

For obvious reasons, quality systems and problem solving are closely coupled subjects. Therefore, much of the literature you will find about solving problems comes directly from quality or manufacturing quality groups. These folks are subjected to daily abuse in most corporations. They are often tasked with cleaning up messes that the design and purchasing (sourcing) folks have created. Because quality teams are typically better at documenting and communicating their work, it stands to reason that the literature is filled with their problem-solving methods.

Everybody—design, marketing, sales, sourcing, production, field or customer support, and even management—needs to be able to solve problems effectively. They need to understand how to contribute to the problem-solving efforts of other people and how to lead an investigation to success.

Don't get hung up on the methods. Don't get so obsessed with the steps and paperwork of one system that you ignore the deeper meaning of what that system is trying to lead you to find (even if you are failing to find it). Quality systems like 8D (Ford’s eight-disciplines approach), the five whys (Toyota), A3 problem solving (summarize the problem on one A3 size sheet of paper), or Ishakawa fishbone diagrams have many devotees.

You can brainstorm life and death on a whiteboard like the fictitious Dr. House, or you can write up a long-winded thesis on a trivial problem. But the fact is, your success will not be measured by slavish adherence to a particular method, format, or sequence of steps in solving problems. Your success will be measured in terms of time, money, and the results you delivered. Your success will be measured in the ratio of products shipped to products returned. Your success will be measured in whether the balance and center of your efforts helped other people or hurt them.

Not all problems are equal in size. Some are tall and some are wide and a few are both. Your success will be measured as the integral of the problems you solved divided by the integral of the problems you created.

Remember that your solutions should be bigger than the problems they solve, when measured in all dimensions. A medication that cures a non-fatal disease in half of the patients, but kills the other half of the patients is not really a good solution. An elegant user interface that looks great on a screen, but connects to a broken database is not going to help your customers.

I believe it is critically important that you become a leader who solves problems for other people—not just your own problems. If those people are customers, your success might also lead to wealth or recognition. All problem solvers achieve a special satisfaction that comes from within.

The people you can help to solve problems include the following:

- Family
- Friends
- Neighbors and fellow employees
- Citizens of your community
- Citizens of your state (county or province)
- Citizens of your country
- Citizens of your world

As you can see, these represent ever-widening circles. People go from being insiders to outsiders. You are less directly related as you go down this list, but I hope that you can still imagine yourself in the shoes of those people who don't look exactly like you.

Will the five questions discussed in these pages feed the poor? Will these methods stop hate or war or racism? Sorry, probably not. But *you* might solve some of these big human problems in your own way.

You might relieve enough pressure from somebody's daily life that they stop blaming the strangeness of other folks for their own misfortunes. You might brighten one child's life just enough. You might spread knowledge instead of ignorance. You might share what you know instead of hoarding that knowledge and wisdom. You might make a friend instead of an enemy.

So as you seek success, fortune, fame, or satisfaction, be sure to seek balance. Be sure to find the center of the problem.

Now get back out there and go fix something.

Sum of the Summaries:

The Five Questions

1. What do you know? (Describe the problem.)
2. What are the rules? (Know the basic science behind the system.)
3. What don't you know? (Outline the missing information.)
4. How can you find out the stuff you don't know? (Do research and experiments.)
5. How do you know when you are ready to solve; or have already solved the problem? (Evaluate and verify your solution.)

The 5 C-words of successful debug:

1. Communication
2. Contemplation
3. Concentration
4. Confirmation
5. Communication

Write a problem statement that collects together this critical information:

1. What?
2. Which?
3. How Many?
4. When?
5. Where?
6. Who?
7. How?

Clearly note items that are suspect. Watch out for false assumptions and avoid jumping to any conclusions this early in your investigation.

Different settings create different communication problems.

1. Verbal Discussions
2. Conference Calls
3. Emails

Use these tools to improve clarity and understanding:

1. Photographs
2. Videos
3. Block Diagrams

Key Question: Has the system ever worked before? (What evidence do you have?)

Hints:

1. Listen to your customers.
2. Practice active listening.
3. Be sure you can write a clear description of how the system is supposed to work.

Overcome common barriers to clear communication

1. Start with an assumption that the person reporting the problem is intelligent and sincere.
2. Listen to what they are telling you without jumping to any conclusions.
3. Be very careful about jargon. Make sure you are both using the same words to mean the same things.
4. Just because you have never seen the problem does not mean that problem is not real.

Methods for Better Communication

1. Use whiteboards (but in a limited way).
2. Convert your brainstorming to an electronic format.
3. Share photographs, video, or documents; any files you can easily store and transmit electronically.
4. Use simplified block diagrams.
5. Use Bug Tracking Software to organize your efforts

What are the rules?

1. Rules represent the basic science behind a system.
2. Many problems come from violating the most basic rules.
3. Constraints are special rules imposed by the goals of each project.
4. All projects include the four basic constraints of People, Time, Money, and Results.

What Don't You Know?

- Brainstorm to create lists of things you don't know (yet).
- Prioritize these items to make efficient use of your time.
- Do not fixate on any one item. This is jumping-to-a-conclusion. Such behavior is immensely damaging to a good debug effort.
- Include this list in your reports. As you discover the answers to things you don't know, you can move them into your list of things you *do* know.
- Don't be afraid to add new items to the *Things You Know* and *Things You Don't Know* lists as you gain knowledge about the problem.

How Do you Find out the Stuff you don't Know?

1. Derive knowledge from First Principles
2. Ask an Expert
3. Look it up on the web (DANGER!)
4. Look it up in a book or magazine
5. Observe the problem with your own senses
    a. See it
    b. Hear it
    c. Smell it
    d. Touch it
    e. Taste it
6. Do some experiments or tests to prove an idea
7. Do some experiments to collect information

Experiments and Tests

1. Experiments can help disprove a specific idea.
2. Experiments rarely prove a theory, but can give you good evidence to support a theory.
3. Experiments can be done to collect lots of raw information.
4. Only change one variable at a time during any experiment.
5. Try to predict the outcome of changes. When you have excellent agreement between prediction and outcome, you probably are gaining some insight into the problem.

Divide and Conquer (Debug by Division)

1. Verify the problem (output is bad).
2. Verify the input is good.
3. Start testing near the middle of the system. Keep a record of every test.
4. If the result is good, set that point as your new start and then pick a new middle.
5. If the result is bad, divide the system from the start point to your current point.
6. When no more division is possible: diagnose that specific node (stage or block) of the system.
7. Check for common subsystems that affect all nodes or stages.
8. Learn the *first-check* mantras for your skills.

Stimulus-Response Testing

1. Do normal environment testing *first*. Build some confidence that your system works.
2. Next stress-test your system over the expected operating ranges.
3. You can reduce your test burden if you prioritize your testing, automate your testing, and use specific point-testing instead of continuous-sweep testing.
4. Don't forget to do appropriate Overstress Testing.
5. Always do all regulatory compliance testing.
6. Test early and test often.
7. View testing as part of design.

If I Could See It, I Could Fix It

1. Your brain helps you convert images to understanding.
2. Sometimes, you just need to see the problem, and then it becomes obvious.
3. Use every tool available to help you see.
4. Sometimes we need to “see” with our mind’s eye, not our physical eyes.
5. Digital cameras with macro capability open up new possibilities for viewing small devices.
6. You can see things remotely using inexpensive cameras and software.

Measurement and Instrumentation Concepts

1. Remember, if I could see it I could fix it.
2. Use instrumentation, measurements, and data-capture to see hidden information.
3. Watch out for false measurements.
4. Understand your measurement errors and tolerances.
5. Does your measurement affect the thing being measured? (Of course it does, but is that variation important?)
6. Understand where you have sensitive nodes in your design
7. Sometimes you need to tear something apart to really see what is going on inside it.

Instrumenting Software

1. If I could see it, I could fix it. (Works for software too!)
2. Use static, extra memory locations to capture transient values.
3. Build "flight data recorders" into your code to let you see what happened just before a crash.
4. Follow the rules for diagnostic messages.
5. Pay attention to your diagnostic message bandwidth.
6. Initialize all exception vectors and have them point to useful diagnostic messages—even if you are sure they cannot happen.
7. Initialize all unused memory.
8. Use code validation tools.
9. Use code peer reviews.
10. The most difficult problems are usually solved by having the hardware and software teams working together—not separately.

Tools and Toolmaking

1. Practice, practice, practice until the tool becomes an extension of your mind.
2. Choose the best tools that your organization can afford and then make it work. Be creative and don't blame your tools.
3. Think about making special tools when that is necessary, reasonable, and appropriate.
4. Calibrate your instruments and keep good records of your calibrations.
5. Understand accuracy and know the common sources of error in your measurements.

Troubleshooting Complex Systems

- Rule 1: Don’t panic.
- Rule 2: Get organized.
- Rule 3: Be methodical.
- Rule 4: Record your work as you do it.
- Rule 5: Be persistent.
- Rule 6: Ask what changed.
- Rule 7: Localize problems.

Debugging Intermittent Problems

1. Reproduce the problem.
2. Try to change the rate of failure. Remember: Increasing failure rates can be just as informative as decreasing them.
3. Try to associate the failure with a variable or condition of the system, but watch out for superstition.
4. You can do parallel testing to increase the number of failures. This can be expensive, however.
5. Change only one variable at a time. (Sound familiar?)
6. Collect more information for each test.

When You Think You Have a Solution, Heed These Warnings:

1. Correlation is not Causation.
2. Your theory of causation had better show good correlation in your experiments.
3. There might be more than one correct answer.
4. Undo your fix and check to see if the problem returns.
5. There can be multiple paths to good solutions. Your way is not the only way.
6. Never allow a defect or problem report to go by without documenting and fixing that problem.

Challenge Questions:

1. How did other people solve problems similar to this problem in the past?
2. Good writers of design are good readers of other people's designs. Are you a good reader?
3. Could it ever have worked that way?
4. Does it work that way now?
5. Are you suffering from optimism bias?
6. How complete are your checklists?
7. Are you doing big debug or little debug?

A Great Idea Doesn't Have to Work, to Be a Great Idea

1. Be careful of Grand Plans and Grand Ideas—it is too easy to fall in love with a solution that you later find does not or cannot work.
2. Watch out for multi-layer, or multiple-cause problems.
3. Understand your limitations.
4. Good documentation helps you succeed and protects you in failure.

Study Your Mistakes

1. Study your mistakes, since they will teach you more than your successes.
2. Once you have learned the lessons, put them into your checklists and then stop obsessing about them.
3. Management has an obligation to be honest about their contributions to problems.
4. Openly discuss mistakes without retribution.
5. Study the mistakes of others too.
6. Sometimes during debug, you will create new problems.

Business (and the Business of Solving Problems)

1. The reward for solving problems is the opportunity to solve more problems. These new problems will be increasingly difficult. Don't panic. Your skills should also be increasing.
2. The best organizations understand how to enhance your ability to solve problems. They actively remove roadblocks and help you when and where they are able.
3. Some organizations decide you need more managers.

Know the Difference Between Customers, Prospects, and Suspects

A *customer* is a person or organization that is ready, willing, and able to purchase a good or service from your company for enough money to sustain your business.

A prospect is a person or organization that could become a customer at some time in the future.

A suspect is somebody who recently was or probably will be involved in a crime, very soon.

Know who pays your bills and keeps the lights on.

Hint: Your customers pay your salary.

- You will get 80% of your income from 20% of your customers.
- Never treat anybody with disrespect.
- Don’t let people treat you with disrespect.

Bibliography and Notes

This section can be used as a suggested reading list for engineers and their managers. Not all books or references will be appropriate for every skill. I have added some comments related to the various sections, but all of these should be considered as only my opinion. *Your mileage may vary* and all other usual disclaimers apply.

Troubleshooting or Engineering Failure stories

There are some great sites on the Web that offer collections of debugging (troubleshooting) stories or stories of engineering failures. I firmly believe that these stories improve engineering skills and that they should be required reading for all engineers and their managers.

http://www.eetimes.com/electronics-blogs/4199254/Engineering-Investigations-Blog

http://www.eetimes.com/electronics-blogs/other/4390231/Engineering-disasters

http://www.edn.com/blog/Tales-from-the-Cube

http://www.designnews.com

(Look for "Sherlock Ohms" blog section. Also look for the "Designed by Monkeys" blog about bad designs that readers have encountered.)

The last entry on this page is a Web site maintained by me to support and promote engineering problem solving in general (and of course to promote this book).

http://www.prettygoodproblemsolver.com

Great Books for any technical person or their manager

- *The Checklist Manifesto: How to Get Things Right* by Atul Gawande (Metropolitan, 2010)

If you only can afford to buy one book, get *The Checklist Manifesto*, listed first above. Read it and embrace it. You will never regret it.

- *The Mythical Man-Month: Essays on Software* by Frederick P. Brooks (Addison-Wesley Professional, 1995)
- *The Psychology of Computer Programming* by Gerald M. Weinberg (Dorset House, 1998)

Great Books for Electrical or Electro-Mechanical Designers

- *Troubleshooting Analog Circuits* by Robert A. Pease (Butterworth-Heinemann, 1991)
- *Hot Air Rises and Heat Sinks: Everything You Know about Cooling Electronics Is Wrong* by Tony Kordyban (ASME Press, 1998)
- *More Hot Air* by Tony Kordyban (ASME Press, 2005)
- *High-Speed Digital Design: A Handbook of Black Magic* by Howard Johnson and Martin Graham (Prentice Hall, 1993)

Here is one additional reference that is worth a look. Some folks might find the topic too dry or too peripheral to design, but this book earned a spot on my bookshelf:

- *Engineering Documentation Control Handbook, Fourth Edition: Configuration Management and Product Lifecycle Management* by Frank B. Watts (William Andrew, 2011)

Engineering Failure

If you really want to understand good design, you have study failures, not successes. I have encountered some really good books on this subject. Another source of similar information is to read the NASA reports on their shuttle accidents.

- *To Engineer Is Human: The Role of Failure in Successful Design* by Henry Petroski (Vintage, 1992)
- *Set Phasers on Stun: And Other True Tales of Design, Technology, and Human Error* by Steven Casey (Aegean Publishing Co., 1998)
- *Air Disasters* by Stanley Stewart (Ian Allan, 1986)

Professional Troubleshooting

There are some other folks out there who teach formal problem-solving for engineers and managers. The gold-standard has to be Kepner-Tregoe. I have never attended training by Kepner-Tregoe, but I have read various works published by their organization.

- *The Rational Manager: A Systematic Approach to Problem Solving and Decision Making* by Charles Higgins Kepner and Benjamin B. Tregoe (McGraw-Hill, 1965)

http://www.kepner-tregoe.com

During my research, I encountered the book listed below by David Jonassen, who dives deeply into the question of how one should teach problem-solving to engineers. He has published extensively on this topic.

- *Learning to Solve Problems: An Instructional Design Guide* by David H. Jonassen (Pfeiffer, 2004)

Here is an excellent reference for Project Management:

- *It Sounded Good When We Started: A Project Manager's Guide to Working with People on Projects* by Dwayne Phillips and Roy O'Bryan (John Wiley and Sons, Inc., 2004)

After I finished writing the first draft of this book, my friend and former manager, Dr. Lauren Christopher told me that she was teaching a class from a book that seemed similar to a brief description she had read of my book. I purchased a copy of the book she referenced only to find that David Agans' book *Debugging* runs deeply parallel to the intent, style, and structure of what I have written.

I was both thrilled and stunned. I was thrilled because it meant that I was probably describing something reasonable. Stunned, because I had never seen anyone else commit such familiar ideas into writing. I must therefore acknowledge that I am not the first to offer such notions. Where Agans tells "war stories," I mostly tell "fairy tales." Where I say, "If I could see it, I could fix it," Agans tells readers to stop thinking and start looking.

I recommend that you treat Agans' book and this one as bookends on your debugging shelf. These books share some similar concepts, presented differently by different authors. I confess to really liking Agans' book. A lot.

- *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems* by David J. Agans (AMACOM, 2006)

There exist some additional books and teachers of Problem Solving methods (Debugging, Troubleshooting or whichever term you prefer). After much debate, I decided that you can use a search engine as well as anybody else, and since I had not read the full text of these references, I could neither recommend nor caution against them.

Complexity Studies

In the course of researching the general topic of troubleshooting complex systems, I came across several fascinating books that in some way or another address complexity or complex systems. The most interesting aspect to me was the rising role of evolution as an observed mechanism in complex systems. Here are a few books that discuss this topic:

- *The Watchman's Rattle: Thinking Our Way Out of Extinction* by Rebecca D. Costa and Edward O. Wilson (Vanguard, 2010)
- *Simplexity: Why Simple Things Become Complex (and How Complex Things Can Be Made Simple)* by Jeffrey Kluger (Hyperion, 2008)
- *The Origin of Wealth: The Radical Remaking of Economics and What It Means for Business and Society* by Eric D. Beinhocker (Harvard Business School, 2007)

Philosophy

I had the odd experience of reading a book about Peter Drucker (management expert) the same week I read *The Peter Principle*. I came to the conclusion that both were teaching the same idea: the latter by making fun of how badly we do management and the former by explaining how we *should* be doing management.

- *Inside Drucker's Brain* by Jeffrey A. Krames, (Portfolio, 2008)
- *The Peter Principle: Why Things Always Go Wrong* by Laurence J. Peter and Raymond Hull (William Morrow & Company, Inc., 1969)
- *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values* by Robert M. Pirsig (HarperPerennial, 2005)
- *The Last Lecture* by Randy Pausch and Jeffrey Zaslow (Hyperion, 2008)
- *The Dilbert Principle: A Cubicle's-eye View of Bosses, Meetings, Management Fads & Other Workplace Afflictions* by Scott Adams (HarperBusiness, 1997)

After you have worked as an engineer for a while, you begin to see the world in a different light. I recently found and loved reading these next two books, but I suggest you wait until you have a few years of industry experience to increase your enjoyment of them:

- *A Shortage of Engineers: A Novel* by Robert Grossbach (St. Martin's Press, 2001)
- *Easy and Hard Ways Out: A Novel* by Robert Grossbach (Harper's Magazine Press, 1974)

Do you ever think about why you became an engineer? You might gain some insight from:

- *Quiet: The Power of Introverts in a World That Can't Stop Talking* by Susan Cain (Crown, 2012)

Other Articles of Interest

Electrical engineers should read anything and everything they can find by by the late Bob Pease. Online, try:

http://electronicdesign.com
and then search for "Pease".

Notes

Where direct quotations are cited by permission of the original publisher, they are referenced by footnotes within the appropriate chapter.

Any real company names, trademarks, tradenames, or servicemarks referenced in this text are property of their appropriate registered holder.

Several clip-art graphics are used in this book. The following images are licensed and copyright © GraphicsFactory.com.

- Page 15: Barking dog graphic.
- Page 78: Mouse, detective, and elephant graphics.
- Page 142: Caveman and saber-tooth tiger graphics.

Acknowledgements

My favorite storyteller has always been my friend, Dr. George Lemaster, who knows more of the history and design heritage of anything electrical or electronic than anyone on planet Earth. Exaggeration? I doubt it. Beyond encyclopedic knowledge derived from reading and study, George has a peculiar talent for striking up conversations with total strangers. Before long, he will have extracted lengthy threads of knowledge and history that people did not even realize they possessed.

George possesses a particular enthusiasm for tracing design heritage. He learns how *this* fellow made the first one of these things; and how *that* person taught the next person who created the next products. George follows how these items came in turn to influence the designs of yet another company through the entire history of a particular type of gizmo.

Here's an example of George's thoroughness: After I shared an early draft of this book with him, he responded with contacts for two additional people he had recently encountered with connections to the parable of "The Dog Barks When the Phone Rings." One said he had encountered nearly the same tale during a Xerox training class in the 1970s. A second contact had personally experienced a nearly identical set of circumstances as a phone repairman. In his case, a dog's chain had wrapped around a pole with a phone wire running from a high crossbar down to an underground feed point. The chain had sawed its way through the phone-wire insulation, with similar results for the poor dog.

I have encountered many guiding lights like George, who also deserve acknowledgment.

One is Andy Bowyer, who is an engineer's engineer. He not only has mastered the most difficult areas of electrical engineering, but he also has deep appreciation for and great skills on the mechanical side. We have enjoyed many discussions about the power and virtues of tools.

Harold Roberts taught me how a systems engineer deals with systems of systems, and that there can be beauty, logic, and inspiration in process and organization.

I learned very much from Hugh Macdonald-Smith, who is probably the most brilliant deductive reasoning engineer I have ever encountered. Hugh introduced me to the book Zen and The Art of Motorcycle Maintenance.

Jon Fasig is an ingenious design engineer who has always freely shared his knowledge and wisdom. He brings a wonderful sense of humor to every problem and has often given me key insights at just the right time to rescue me when I was lost.

Mark Fehlig allowed me to hang around, listen, and learn at a time when I knew nothing, even though I demonstrated my ignorance on a continuous basis. Mark also shared some wise and important stories.

I must mention my good friend Mark Schultz, who can open locks and safes by simply listening to them and visualizing in his head how they work and how microscopic defects in their manufacturing lead to tiny indications in the sounds they make. Mark also has designed integrated circuits, from the physical transistor level up to massively integrated system-on-a-chip projects. He has more than 50 patents, but at that level of genius we stop counting.

The manuscript for this book was edited by the amazing Kate Shoup. She deserves full credit for anything you find here that is coherent or readable. Any mistakes belong to Bob.

My thanks to Susan Hall and Jason Schmidt for their excellent quality reviews prior to publication.

A special thank you goes out to Joe Desposito at Electronic Design Magazine for his kind permission to include George Rostky's *The Shell Race*.

Finally, this book is dedicated to my beautiful and brilliant wife Debbie. None of this would have been possible without her.

Index

BECOME A BETTER PROBLEM SOLVER

The transition from engineering school to real world problem solver can be rough. Suddenly there is not just one correct response to a problem. There might be an infinite number of correct solutions where some are simply better than others. Some problems are so layered and twisted that their solutions seem absurdly complex.

Arm yourself for success with the methods in this book:

- **The Five Questions every problem solver must answer.**
- **The best and worst ways to communicate your ideas.**
- **New ways to see what other observers miss.**
- **Mastering the right tools.**
- **Six warnings to heed when you think you have a solution.**
- **Critical challenge questions before you declare victory.**

Employers and customers cherish engineers who consistently meet their toughest challenges. ***An Engineer's Guide to Solving Problems*** delivers simple methods, practical advice, and entertaining stories to help you sharpen your skills.

Bob Schmidt is an electrical engineer with four decades experience in hardware, software, and systems design. With over a dozen patents, he has solved problems as an engineer, as a manager of global teams, and as an entrepreneur. Bob also shares more parables for engineers at: **www.PrettyGoodProblemSolver.com**

ISBN-13: 978-0-9887476-2-3

51995

9 780988 747623

AN ENGINEER'S GUIDE TO SOLVING PROBLEMS

BOB SCHMIDT