SESUG Paper 190-2023

Zen and the Art of Problem Solving

David B. Horvath, MS, CCP

ABSTRACT

Although software development is taught as a STEM out of Science or Engineering schools, it is as much an art or craft – a creative process – as a science. This presentation focuses on innovative problem solving techniques the tools and techniques to use when your normal process just doesn't seem to get you to a solution. Much of the information in this talk is based on Robert Pirsig's *Zen and the Art of Motorcycle Maintenance*, which, although it focuses on Motorcycles, applies to all kinds of problem spaces (and Pirsig was a tech writer for IBM). These techniques have served me well over the years. The difference in art versus science approaches is actually supported by the way the brain works.

INTRODUCTION

We will discuss how Zen applies to Problem Solving – in particular, how problem solving is an Art as much as it is a science. Some of the important topics will include:

- Traditional Problem Solving Methods, where they fall down
- Eastern techniques, advantages, disadvantages
- Tips
- Brain versus Mind

PROBLEM SOLVING

I was first introduced to Robert Pirsig's *Zen and the Art of Motorcycle Maintenance*" at a DECUS (Digital Equipment Computer User Society) Symposium session that I attended on a lark. What a learning experience for me that was – I learned more from that session and later the book about problem solving than I have in many hours of corporate training including "Six Thinking Hats" and "Lateral Thinking".

Although written directly about Motorcycle Maintenance, the book covers many issues in life. Those lessons actually apply to our daily lives and jobs. After all, software is an even better source of problems than motorcycles! If you think about it, pretty much all we do is solve puzzles and fix problems. And the work we do is more of an art than a science – we're really working in a craft.

If you think about it, your job is most satisfying and you feel the most joy when you are building a good product – creating a good solution. "Quality" is the real essence to what you're doing and behind the joy you receive. Unfortunately, it can be hard to solve some of our problems resulting in frustration. Think about how you feel when you are not able to reach a solution (or feel you have reached an unsatisfactory solution).

We really face two aspect: Gumption (or gumptionality) which is the energy to maintain mental acuity under stress and Stuckness which is related to not being able to reach a solution – or more importantly, how to getting unstuck.

PROBLEM SOLVING – STUCKNESS

We all have experienced the condition of being stuck. It happens when we have a problem, do not know the cause, may not know where to look, certainly don't know how to fix (resolve) the problem, and often don't even know what questions to ask. We have a problem, we do not have a solution.

It is important for you and your management to understand that outside pressure will not help you get unstuck – if anything, it will get you even more stuck. Your boss (or even the CEO) standing over your shoulder is not going to help – the stress feeds on itself making you even more stuck.

It really annoys management when you walk away from the problem to solve it. When they do not see you actively working at the keyboard they perceive that you are not working and are not concerned. But walking away or even reading a book sitting off in a corner can help you get unstuck.

When you get stuck, Western thought – the scientific method as taught in schools – has failed you. This method consists of the following steps:

- Ask a question (“Why did this program fail?”)
- Research (inputs, outputs, etc.)
- Hypothesis (“I think it failed because …”)
- Experiment (If you change the “because” does the problem go away?)
- Data/Analysis
- Conclusion (We’re done)

What happens is you have no hypothesis or if they are repeatedly proven incorrect. This occurs because the scientific method is good for verifying truth in retrospect. But when you’re stuck, you’re trying to look forward to solve the problem.

Further complicating becoming stuck is that the fear and anger cycle takes over. You become angry over being stuck, the pressure from not solving the problem increases anger which increases fear which increases pressure in an ever-increasing cycle. As a result, you focus on the problem itself rather than on the facts that could help you diagnose.

In some situations, getting stuck can be deadly. That is one reason so much of pilot training focuses on emergencies. The student is taught about the problem, causes, and solutions. Those solutions are actually practiced and results are debriefed. The instructor will pull the throttle and declare “your engine just quit, what do you do?” The student works out the solution and the end result is discussed afterward. The instructor maintains a safe environment. If the student ever does experience an engine failure, they do not become stuck because they’ve already had a safe resolution to similar problems. The Military, Medical Professions, and Rescue Services follow similar training practices.

Even if you do not have experience in any of those fields, you encounter stuckness. Think about when you're having trouble finding something – you start looking, can't find, thrash around looking in some of the same places. Fortunately, if we get stuck, it is rarely deadly (except in the movies); unfortunately, we can't train around all but the most common problems.

To get around being stuck, you need to find calmness and become one with the problem. You need to be able to look at the facts and sort them by their quality. You likely have a mountain of facts but have been unable to recognize them – to differentiate between relevant and irrelevant – you’ve lost the ability to judge quality.

We’re used to Western Thinking which focuses on objective and arms-length results. In reality, something trivial can become the biggest problem in your life. This happens in IT all the time. You can change some number in the code that had magic properties and suddenly causes all kinds of seemingly unrelated problems. How many times have you heard (or even said) “But I only changed this little thing”?

Pirsig writes about a buggered screw on oil pan cover that, in itself, is really only a minor problem. But because being unable to remove that screw becomes the biggest problem in his life because it is on the critical path! As a result, frustration grows.

Understand that no matter how stuck you are now, you will become unstuck at some point. The problem will go away or get resolved eventually, it is only a matter of time. You could always abandon the problem. Or you can solve the problem. In solving the problem you will learn, become a better person, and become a better problem solver because just knowing you can solve difficult problems can help you become unstuck.

One approach is to look at each problem as being unique. In western thought we solve problems by classifying them. But when we’re stuck, we can’t classify the problem and may just pigeonhole it in the wrong place.

The solution is to analyze the problem on its own merits, its uniqueness. You need to look at all of the facts and even focus more broadly than you had previously (unfocusing). By looking at everything that is happening, including other activities or processes even if seemingly unrelated, you can discover the real factors causing the problem.

The best way to understand this is with a real example. I was supporting a datawarehouse. Without production problems or delayed start reports were available for our Executives by about the time they finished their coffee. But if there was any delay – the source systems didn't complete on time or we experienced a failure, delivery would be delayed dramatically – much more than just the hour or two caused by the failure and subsequent recovery. We would kick users off the system, monitored system resources, and engaged the UNIX administrators. No one could find anything that would cause that delay. We were totally stuck and multiple levels of management were hanging round our desks trying to get resolution.

Purely by chance I was describing the problem to one of our Backup/Storage team members. It turns out the daily backup would kick off at 8 AM because they thought we were always done by then. The system was running slow because of the I/O load on our disks. But because that was independent hardware, it was not apparent in any of the UNIX tools. Ultimately, we changed the process to initiate the backup after our load was complete rather than relying on time. Here was a simple fact that was the cause.

PROBLEM SOLVING – GUMPTION

Gumption is defined as the perception of quality in the thing we are doing. If we are doing good work, our gumption supply grows – we feel better, we feel pride in our work, work/I merge into one (you feel "I am good" when you feel good about the work). Unfortunately, the opposite is true.

Gumption will help you solve problems or make them more difficult. If your supply is empty, problem solving becomes harder, it is harder to focus. If your gumption is leaking, you need to restore it. It is important to monitory your supply, the flows, and direction of flow.

I've also heard this referred to as "emotional bank account" – although that tends to be used more for interpersonal interactions. Another way of thinking of Gumption is as another form of Resiliency.

Gumption Traps

There are many traps or sinks for Gumption:

1. Dependence on a Commercial Mechanic
2. Taking things apart/Putting back together
3. Intermittent Failures
4. Part or tool (or information) setbacks
5. Internal

We will review each of these in turn.

Gumption Traps – Dependence on a Commercial Mechanic

We all end up using commercial mechanics – even if we do not own a motorcycle or other vehicle. After all, consultants are the Commercial Mechanics of IT! The issue becomes when you totally rely on them: they end up knowing more about your problem space than you do, as a result your gumption supply heads towards depletion which makes you weak. They are the ones who gain from solving your problems.

A good consultant will teach you as they solve your problems – weaning you away from reliance on their skills as yours grow. Working in that mode may cost more time (and more money as a result) in the short term, it does make you better at your job – grows your gumption.

When you learn to solve the problems, not only do you gain directly, you restore your own gumption – improving your future problem solving abilities.

I spent most of my career in consulting, often in a staff supplement role where esoteric skills were not otherwise available. I've made a point of sharing knowledge; if anything it has made me more valuable and extend my time at the client as they realize I can help their teams too. That helps my gumption as well as I'm faced with more, varied, and interesting problems.

A few words for consultants: try to avoid your customers being dependent on you. You want them to solve their own problems and grow their own gumption. While the cross training you provide will reduce their reliance on your skills it greatly improves their perception of your abilities. Maybe even more importantly, it helps prevent burnout – depletion of your own gumption. For me, variety is part of the reason for being in consulting and cross training allows me to move to other problem spaces.

Gumption Traps – Taking things apart/Putting back together

You often run into problems when you take something apart and subsequently put it back together. For some reason, the result isn't quite complete or will not work correctly. This is often caused by assembly out of order (did you run the programs in the correct order?), wrong or incompatible versions (this is common with upgrade – operating system or application), missed pieces (or extra screws – are all the correct input files available?), and lost pieces (the spring that bounces away). Not being able to find the missing piece or figure out where the extra one goes will cause gumption desperation.

There are steps you can take to limit problems like this. Keep detailed notes like runbooks: what programs, what ordering, which directories, and names of specific files required. Using this information, you can set up the environment properly before beginning execution rather than constantly fixing failures from missing files. Build tools and shell scripts help standardize the processes to limit the missed items.

Unfortunately, even with the best documentation there can still be issues. When performing updates, read the release notes; while they are not perfect, they should cover the common situations.

Gumption Traps – Intermittent Failures

Intermittent failures can be the most frustrating kind of problem. The bug seems to go away and comes back. Or the strange noise in the car that never happens at the mechanic's shop.

I've spent days tracking down a failure in dynamic memory allocation. It turned out that a bug in the code (written by someone else) was corrupting internal data structures when the input line was too long.

It is important to realize that these will not heal themselves. Sometimes they are here now but gone by the time you can look at them. You are infuriated because they go away on you and embarrassed because they come back. It is even worse when the problem isn't there when you call on a peer or "Commercial mechanic" (consultant) to help.

Sometimes it seems that rebooting the operating system is the solution. It does seem to help and is the preferred solution under Windows. But you are masking the real problem. In all honesty, sometimes it really is the "best" solution because the real solution is too difficult to find.

I worked on one OLAP/BI tool under Windows that would display flakey behavior. Reporting would be running just fine and then queries would begin to fail. Patches and updates did not prevent the problem. Over time, we began to notice that the longer the time since the last reboot the more common the problem. We determined that we never had the problem within 10-14 days of a reboot. Since it was easier to schedule on a weekly basis, we set up a reboot every Sunday evening (since no one was using the system).

I describe the non-deterministic load times we experienced with a data warehouse already. This seemed to be an intermittent problem – if we were done before 9 AM, runtimes were reasonable. If we ran later, runtimes became much worse – well outside every volume variances.

Of course, the more you know about your environment, the deeper you can analyze. In the best case, you can solve the problem but even if you do not, your gumption grows as you learn.

With problems of this nature, you need to look at broader horizons. Completely unrelated facts may have bearing on the problem. You need to broaden your focus and ask “What else is happening?” You may even be looking at new hypothesis.

Gumption Traps – Part or tool (or information) setbacks

It can be frustrating when you have a part or tool or information setback. These are problems that have already been solved but you are unaware of them or cannot acquire the solution. For instance, a bug that the vendor has already fixed but you cannot install the improved version until the next release cycle.

This happens outside of IT too: you can’t get need parts, or the parts are too expensive, or you receive the wrong part, or delivery is delayed impacting your timeline.

All of these situations are frustrating, leading to gumption depletion, and causes you to be demoralized. Most of the time you’ll end up having to find a workaround without the quality of your initial solution (a “hack” or “kludge”). The worst ones are solved before but conditions bring them back.

Ironically, this can be a gumption builder at times. I worked on a project where we had to write our own screen handling package. The vendor would sell us a well-engineered package but we only had expense money (and software purchases are capital). Initially, this was a gumption deplete because we had a preferred solution we were not able to take advantage of. We ended up spending more time and client money than if we could purchase the package. We were forced to learn new technique and grow professionally which ended up building our gumption!

In real life, if you have to buy a part, you should take the part or a picture with you – so you can be sure of getting the correct one. This is better than measuring and describing the part. But this is much harder to do with software. If you bring a peer in to help, do not tell them about the problem, show it to them. And when you’re helping someone else, do not let them tell you about their problem. You should look at the problem yourself and ask your own questions. Otherwise you’ll be conditioned to their selected facts when the viewpoint needs to be broadened. After all, their thinking already got them stuck and you don’t want to go down their path!

Innovation is ultimately a gumption builder. Finding and fixing a bug causes you to grow technically which adds to your supply of gumption (and confidence) and leads to your being a better professional.

Gumption Traps – Internal

There are gumption traps within ourselves too; they are how you think and feel about the problem. These are really the largest group of traps and sometimes the hardest to avoid.

(internal value traps)

Internal Value traps block your ability to solve a problem or apply a solution. This is a mental form of stuckness: you apply the same quality/truth to a problem. When you enter with a set of fixed mental positions, you start applying the same truth values to any problem and force facts to fit your mental positions. This is essentially the backward implementation of the scientific method and you often encounter this in political discourse. When this occurs, you need to slow down and avoid the “ah ha” diagnosis. Simply stare at the machine or, even better, out the window.

With internal ego traps, you believe you (or your code) cannot be the problem. This is when your ego gets in the way and you are in really in trouble. You will recognize this situation is occurring when you find yourself asking questions such as “How could this happen to me?”, “Who, me?”, or “After all, I can’t be the cause, right?”. When this happens you need to slip back into a good methodology: check all the facts and understand that you can be the cause; reading the manual is a good approach. Your ego will isolate you from your problem solving abilities. It seems that machines respond to our real personalities. To fake

out this anthropomorphic mess we call a computer, you must assume an attitude of modesty, even when you know you are not the cause. Especially when you know you are not the cause.

When we become anxious, we fall into the opposite of ego traps: the fear of messing up. When you are afraid of screwing up, you get stopped – unable to move forward. Just remember, this is only software we are dealing with here. You can always revert to the backup (you do have a backup, right? And you've tested recovery from your backup, right?). For most of our applications, you cannot kill anyone, no matter how much you mess up. Older folks seem to suffer this with technology in general – they are so afraid they are going to break the VCR or DVD player of smartphone that they barely use the available features. If necessary, you can spend a bit of time reviewing the problem before you start your work in order to build a written plan. Having a plan slows you down, forces you to think, and builds your confidence (you trick your mind into believing you survived the problem before). Emergency procedure training for pilots and other fields follows this pattern.

When you do not respect the thing you are doing, you are falling into the boredom trap. When you become bored, you must stop before you make a big mistake. Seemingly minor or trivial actions like removing a spring or relinking some code are no longer trivial when they become critical path due to failure. When you find yourself getting bored, get away from the action, do something for a period of time until you realize the value of the original task, and then return – but do this before you make a big mistake!

A similar trap is impatience – when you have more value for the time than the action. When you are impatient you often break something, hurry to fix what you have just broken, and then you'll break something else! Unfortunately, the cycle repeats in an ever tightening spiral. When falling into this trap, you need to throw away time as a factor (scheduling pressure). Even though your manager may say "this must be done by Friday" and the problem is a technical issue totally unrelated to the calendar, try to mentally uncouple the time pressures from the problem solving. But rushing won't get the problem solved any quicker. It can be hard for your manager to understand why you seem to be working so slowly when the result has to be delivered (or demonstrated) this afternoon. They will be more understanding when your credibility as a problem solver has been proven.

The final internal trap is referred to as psychomotor as it relates to the feel a mechanic has. Psychomotor traps inhibit your ability to apply solutions. These generally fall into the categories of inadequate tools, bad surroundings, and muscular insensitivity. With the first two, not only is it difficult for you to get the work done, you end up frustrated. Most developers know exactly how much they can tweak a parameter or expand an array or extend an object before it just won't work anymore. Without that skill, you will need a torque wrench or other good tool to measure the effects of changes. Knowing when to stop is where it all happens. That is the "mechanic's touch".

WHAT DO I GET FROM ALL OF THIS?

No matter how much attention you pay to problem solving and try to avoid gumption traps, you can still become stuck. But understanding what is happening can help you resolve the problem. If you know why you're getting angry about something, you can deescalate. If you know why your gumption is fleeting, you can work around it.

Pirsig's closing remark is telling:

> Some could ask, "Well, if I get around all those gumption traps, then will I have the thing licked?"
>
> The answer, of course, is no, you still haven't got anything licked. You've got to live right too. It's the way you live that predisposes you to avoid the traps and see the right facts. You want to know how to paint a perfect painting? It's easy. Make yourself perfect and then just paint naturally. That's the way all the experts do it. The making of a painting or the fixing of a motorcycle isn't separate from the rest of your existence. If you're a sloppy thinker the six days of the week you aren't working on your machine, what trap avoidances, what gimmicks, can make you all of a sudden sharp on the seventh? It all goes together.

> But if you're a sloppy thinker six days a week and you really try to be sharp on the seventh, then maybe the next six days aren't going to be quite as sloppy as the preceding six. What I'm trying to come up with on these gumption traps I guess, is shortcuts to living right.

MIND VS BRAIN

We often have difficulty with solving problems because of how our brain works and that affects the mind that resides within. The brain is very good at optimizing repeatedly used paths. For instance, when you first started driving, it was difficult. The more you drive, the easier it becomes and reaches the point you don't even think about it anymore.

Unfortunately, if those paths are not "good" or "appropriate", it is hard to stop using them. There is even a term for that: "negative learning". Have you ever lost anything and keep looking in the same place for them? Or look at the same code over and over trying to see the broken part?

The solution to the path lock-in is to break away from the problem and allow other pathways to be exercised.

SOME BASIC TIPS

Sometimes you just have to get away from the problem. Taking a walk around the building or longer exercises other portions of your brain. So does reading an unrelated book. Or even going out to lunch. While your boss may not appreciate these approaches because they give the impression of disengaging from the problem, they really do work.

I've forced team members to physically break for lunch to get their minds onto other topics. When they return to the problem, they're refreshed and come up with quicker solutions!

I'll admit that I've solved some of my grodiest problems in the shower.

FINAL THOUGHTS

In all honesty, Pirsig can be hard to read. After all, he was a technical writer for IBM (and if you've ever read an IBM manual you'll understand this remark) as well has having spent time in a mental hospital. But you can learn a lot from him and the other references I provided below.

CONCLUSION

Our field is as much an art as it is a science or engineering discipline. As a result, following traditional scientific methods may make problem resolution more difficult rather than helping. Learning more about how we actually think and react will help solve problems – will help you become a better professional.

REFERENCES

Brooks Jr., Frederick B. 1982. Mythical Man-Month

Constantine, Larry. 1995, Constantine on Peopleware.

DeBono, Edward. 1973. Lateral Thinking – Creativity Step by Step.

Demarco, Tom and Lister, Timothy. 1987. Peopleware: Successful Projects and Teams

Pirsig, Robert M. 1975, Zen and the Art of Motorcycle Maintenance, Available via http://www.chiro.org/LINKS/FULL/ARCHIVE/Zen/chapter26.htm

Plauger, P.J. Programming on Purpose Volumes I, II, and III

Von Oech, Roger. 1990. A Whack on the Side of the Head

Weinberg, Gerald M. 1971. The Psychology of Computer Programming

Weinberg, Gerald M.1988. Understanding the Professional Programmer

McGlinchy, James. Fall 1991. “Zen and the Art of Programming” DECUS Anaheim Fall 1991 Symposium, session LT062 materials

German, Hallett, 1992, “Problem-Solving Looks East”, NESUG 1992

ACKNOWLEDGMENTS

I would like to thank the organizers of this conference. This is neither my first SESUG nor first time speaking at SESUG – I’ve always enjoyed working with the organizers.

I also want to thank my employer for their support in attending (and speaking at) this conference even though I’m not allowed to mention their name. Last, and certainly not least, is my spouse Mary who doesn’t complain when I spend time working on conference materials.

I also need to acknowledge my previous managers who tolerate my non-traditional approaches to problem solving along with all those who practice similar methods.

RECOMMENDED READING

- *See* REFERENCES

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David B. Horvath, CCP
+1-610-859-8826
dhorvath@cobs.com
http://www.cobs.com
http://www.linkedin.com/in/dbhorvath