SYSTEMS THINKING TESTER

Russell J. Smith
savtech138@gmail.com
https://www.linkedin.com/in/rsmith85

ABSTRACT:

With the implementation of Systems Thinking as a mindset, and pulling from Systems Thinking principles, I believe that testers can be more valuable to their organization in at-least 2 ways. 1- They can apply Systems Thinking to their own existence in the system they live and work in. 2- They can educate and coach the teams they work with and improve the system as a whole. Because of the unique position we are in, and the number of individuals and teams we interface with, we are at an advantage to spreading ideas just as bees spread pollen between flowers and thus having influence on the environment in a positive and meaningful way. Because Systems Thinking can be applied so vastly I have chosen to narrow the focus of this paper to using Systems Thinking laws as a Tester in a non-management role, though some areas may be relevant or touch base on test management or organizational management and situations where you are interfacing with management. This paper will be guided by the Systems Thinking laws as described by Peter Senge and how they are at play during your everyday life as a Tester. It is aimed at helping you recognize the situations in which non-Systems Thinking is at play and how to apply Systems Thinking during those situations. Anyone reading this should be able to extract the concepts and apply them to what they do regardless of whether they are a Tester or Test Manager.

BIOGRAPHY:

Russell Smith is a Software Test Professional and Quality Advocate who started his career in 2006 doing Technical Support and Black-box testing for a high-tech GPS producer in Torrance, CA. Over the course of his career he has held positions anywhere from "Junior QA Engineer" to "Software Test Architect". He has worked on dozens of products and systems deployments and has contributed to projects that span multiple teams with dozens of people involved across multiple countries. Most recently in his career he has been focusing on building high performing software testing teams while at the same time staying hands-on as an individual contributor. He does not believe in "Ivory Tower" management and loves to "get his hands dirty". Through his career he has never lost his focus on the customer and that what he does means nothing unless his customers are satisfied. This combined with his technical skills and leadership abilities makes him a world-class software test professional and customer-centric software quality advocate. Russell now lives in Portland, OR and works as a Test Manager at Janrain, Inc., a leading provider of customer identity management and social login technology that helps businesses manage and understand their customer profiles and profile data.

1 Introduction to Systems Thinking

Systems Thinking stems from General Systems Theory and was popularized by Ludwig von Bertalanffy in the 1940's [1]. There have been many others who have advanced and greatly educated the world on systems thinking such as Deming, Senge, Weinberg and Ackoff. In the most basic sense, Systems Thinking is really just a set of tools, concepts and ideas that can be applied to dealing with messy complex systems. Note that it is not a magic wand or something that solves all of your problems for you. But with a better understanding and discipline of Systems Thinking you will be well equipped to navigate and drive the system in a more aware, intelligent and genuine way in order to achieve a desired outcome and know how you got there. As you dive deeper you may also find yourself crossing into System Dynamics, which shares concepts with Systems Thinking such as feedback loops and time delays [2], but tends to be a bit more math-heavy and uses computer aided simulations. Once you begin to adopt a systems thinking mindset you start to see the connectedness of the parts and can act more intelligently about your decisions and actions when dealing with problems of complex systems. You start to see that focusing on individual parts without understanding how the parts connect [7] and impact the greater system is rather short sighted and naive. The absence of Systems Thinking is to focus on individual parts through pure analysis and with a lack of consideration for the system the parts belong to. Without Systems Thinking people would naively assume that a solution is "the best solution" or "the right solution" without having genuinely framed the problem space they are solving in and considering alternatives. It's not to say that the decision made is right or wrong, but that there are always alternatives that you can model and tools that you can use from Systems Thinking to test your decisions and measure them. As an example we can take the transportation system. Serendipitously as I was writing this paper I stumbled upon a report for just this: http://www.cts.umn.edu/Publications/ResearchReports/reportdetail.html?id=530. In this report the author uses systems thinking to 1. identify the problem space 2. understand the relationship that the system has to other systems 3. suggests alternate recommendations for sustainable growth based on the previous two points. As cited from the report, "This study is an effort to better understand the linkages between land use, community development, and transportation in the Twin Cities metropolitan area. It is designed to investigate how transportation-related alternatives might be used in the Twin Cities region to accommodate growth and the demand for travel while holding down the costs of transportation and maximizing the benefits." This, I think, is a perfect example of applied Systems Thinking. In the context of a Systems Thinking Tester I would expect this paragraph to read something like, "This study is an effort to better understand the linkages between the Database, API and User Interface in Project X. It is designed to investigate how testing alternatives can be applied to maximize test value, effectiveness and defect discovery or prevention while holding down cost". You can keep going up a level and applying the same concepts such as the reason for having a Testing/Quality initiative or a particular team within your organization. The beauty of Systems Thinking is that the concepts and tools can literally be applied at any level you want to apply them to. Anywhere from devising a test strategy, to building a product, building a team, running a company and so on.

2 Systems Thinking Principles Applied to Software Testing

2.1 Senge's Laws of Systems Thinking - Applied Scenarios

I found the easiest way for me to guide this paper was to simply use Peter Senge's 11 Laws of Systems Thinking [3]. Reading below you will see each law's title and a replacement text that changes his words to my own, which are focused on how they relate and are relevant in the context of software testing. It is written such that common scenarios will be described where the systems thinking laws are at play and how to treat them for a chance at a better outcome. If taken and practiced, I believe these laws provide a great entry into a deeper journey and provide the foundation for testers to push from when trying to elevate their effectiveness and existence in working and living in complex systems.

2.1.1 Today's problems come from yesterday's solutions

This law may be self-explanatory, but at the expense of seeming redundant I'll try to define it before giving an example. My understanding of this is that it is the ability to create new problems with every problem solved. Inevitably when you produce a solution you introduce something into the system or change something about the system that will become a problem for something else in the system. Without understanding the trade-offs you may end up with more problems than it's worth having for your one solution.

As a tester you are responsible for coming up with solutions all the time. It's not just developers creating solutions and problems. Automation, Test Approaches, Test Tools, etc. These are all solutions to problems you face, and in-turn your company faces them too. Each individual's problem in a system is the system's problem. If you propose a solution to do automated regression then you should be thinking through the required training and maintenance that comes along with it. You should also consider how accurate the automation will be at the thing that it's doing. Perhaps an unintended consequence is that your automation returns a false positive, in which case you pass the build and a customer ends up with an undesirable experience. So now you have the added complexity that **the solution that checks your solution is correct also needs a solution to check that it's correct**. Turtles all the way down as they say. So maybe this is not the least fallible solution after all. Going up another level we can ask why the automation was even asked for. Did we have an issue with regressions which provided evidence to support the cost of adding an automated way of regression checking? What are we doing about why we have regressions instead of focusing on reacting to a symptom? Asking these types of questions is kind of like doing a premortem. This is an exercise I advocate that all teams and individuals do to question their assumptions and claims. So you see- because your solution was automation yesterday, today you deal with maintenance cost, specific skill-set to build/own it, possible brittleness which takes time away from finding bugs because you are fixing/extending/refactoring/optimizing/training for something else. You will ultimately have worse quality in the end because your team is not finding any new bugs or you delay schedules. I'm not saying automated regression checking is a bad thing but it's not always the right thing, and good testers should realize this and be able to have conversations with their teams about the trade-offs and total cost of ownership for the decisions that are made.

2.1.2 The harder you push, the harder the system pushes back

In Systems Thinking terms this is referred to as compensating feedback. Initially it may be difficult to see this in your life as a Tester. You may not be able to think of scenarios where this occurs in your day to day tasks, such as coming up with test scenarios, setting up those scenarios or executing them. However- I have come up with a few that I think might hit home.

1) You frequently send out long reports about how bad a process is or the quality of an area of the product

Result: Depending on your approach will determine if anyone listens to you or not. If you complain in a negative or impolite manner, then you may be ignored even if you are right. Quality does not improve and the situation does not change because you continue to berate the product or teams in a way that is off putting. Even if you are kind and thoughtful about it but your message is so long that it takes your teams 20 minutes to read them, they won't get read it. Maybe at first they tried to, but the more you sent these long messages the less people read them until they reached the point where they stopped completely. Eventually you have an email rule created just for you that dumps your messages into a folder labeled "Fireside Reading". You have to understand your audience and be aware of yourself when you are trying to send your message. If you send a long message and don't hear back at first, then you should go face-to-face with some of the recipients and try to collect feedback so you can adjust your approach next time.

2) You want a developer to see his/her bad ways, but they refuse to

Result: This is similar to the above point. If you are not careful to understand the developer's personality or feelings, then you can create a situation of hostility where things only get worse. You push them the wrong way for better quality builds, and they in turn feel pressured or even angered which could lead to them delivering even worse builds. Or perhaps they deliver the same build, but when you have questions or need help setting up a test scenario, they will be less inclined to be responsive or help you. All of which will end up creating a poor experience for you and likely a poor performance for the team which could result in poor experience for your customer.

3) You are worried about the quality of a product and the lack of time to test it adequately, so you decide to work late a few night to make up for it.

Result: I can relate to this example quite possibly more than any other I listed. There have been many times when I wanted to play superhero and save the schedule by putting in extra long hours myself. Huge mistake. Usually what this led to was me being late to work the next day and then trying to cram more work in shorter hours. That didn't work out, so I made up for it when I got home. But then I had to cram in my dinner and time with my family. Tried to cram in more make-up time with work before bed which shortened my sleep cycle. Made me late for work again and then the planned testing I wanted to do slipped as well. A perfect example of this principle at work I think. What you should do in this case is make your case for more time to test if you really feel it is important.

These are just a few of the examples I can come up with. Now if you just think about this from your own perspective, you will come up with more like this. Take the same approach to thinking about your own examples of this, and try to think where else you see this happen. For example I can think of some examples where I see teams suffering from this such as:

1) We want more features added so we add more stories to our sprints
2) We want less defects in production so we add more testers to the team
3) We want faster delivery of features so we hire more developers
4) We want less distractions so we create an intake process

2.1.3 Behavior grows better before it grows worse

Here we have the case of a short term solution that causes a bigger problem down the road. Initially you see improvements in the project or product, but later in time, and made possible through compensating feedback [4] (as mentioned in 2.1.2), you find yourself in a bigger mess. In addition to short term solutions you may be treating the same symptoms for a long period of time while missing the underlying systemic cause for those symptoms. Homeopathy may be a good example to showcase how a system can get better before it gets worse, where you are treating an illness with a placebo and, although the patient may trick themselves into feeling better, they are actually allowing the real disease to fester and grow into a worse affliction. Eventually the patient is now at a further stage of their illness which may not be as easily treatable as if they had gone with a more scientifically proven or medically proven treatment option.

An example that may be common for Testers is when you have an insurmountable amount of Testing to do with a schedule that seems too short to accomplish the Testing task. Your test manager or you yourself may decide to "cut" certain tests based on what you perceive as risk factors. You may even consider and ultimately decide to cut entire contexts from your testing strategy. Rather than raise concerns about the schedule and the reason that there is not enough time to do all planned testing, you proceed forward with the "slim down, schedule safe" plan. Great Success! You made your release date and shipped the product. Mission complete and on-time to meet the customer deadline. -Fast forward a few weeks- Bug reports and escalations start to roll in. P1's and long support queues build. The customer and the delivery team that were once so happy are now realizing the mistakes they have made, but each

still has a chance to make it worse or begin to make it better. Deciding to quick fix these issues may actually lead to another dilemma which is addressed in the next law. As you might be able to tell- all of the laws are connected as well. Instead what the team might have done to avoid this situation is explore alternatives. They could have cut the release commitment down to a reasonable level. They could have asked for more time. They could have added more Testers. They could have done a better job at their risk factor analysis. Of course each of these options holds the potential to have its own problems, but without playing out the alternate scenarios ahead of time, you are setting yourself up for unexpected and surprising failure. Rather than feeling a nervous anticipation, you are blind-sided by reality meeting your ill-preparedness. This example is actually a small time delay in comparison to architectural design decisions (or lack thereof) of a system that are found in time to not be sustainable. Again- the laws are at play on multiple levels depending on where you look or who you ask.

2.1.4 The easy way out leads back in

I recently saw this in action while talking with a friend at his company. He was part of a team that was having many distractions due to escalations, bug fixes and customer requests coming into their Engineering department. Because the developers and testers felt distracted by these things, causing them to complete their new feature work slower, the management team decided to institute a "Maintenance Team". This team was tasked with handling all incoming issues that required a code change. Any other issue could be handled by their support team. The maintenance team was there to help alleviate the distractions from the core Engineer team. Can you guess what happened? Of course. The distractions were moved up. Actually they weren't even fully moved up because the maintenance team still needed to ask the core team questions, since the issues came in from many different areas. So distractions did not go down, feature development went down because of the developers moved off core work items and placed onto the maintenance team, and the team tasked with handling distractions now had all the distractions! Wow! A tactical quick fix led to the same issues they were trying to solve, just at a different level. From a Tester perspective we have probably made this same kind of decision though. For example I can think of several scenarios where I have either done it myself or seen someone do it. Examples:

1) Test Case Creation

   Detail: I've interviewed a lot of Testers in my career thus far. One of the questions I used to ask was something I stole from James Bach. James was lecturing at a college where he showed the class a flow diagram with 2 options. If the input was greater than a number, it would be true, and if it was less than a number, it would be false. He then asks the class how many test cases would be required to test this. When asking candidates this same problem, about 90% of them said, "3 test cases". Some even said "3 test cases at a minimum". It was always fun to see how many more they could come up with before they ever asked a single question about the problem. They would throw out answers like 3, 4, 5, or even 100 test cases. Typically though it was 3 test cases which involved the 2 paths for true and false plus one random number like a negative number or a really big number. When asked where they came up with the third number they would just say something like, "I just usually try random numbers when I am testing input". So- Let me get this straight. You just throw numbers at anything? You don't try to think about it or how it gets the input? Or the fact that all I did was draw a diagram on a whiteboard which is probably a very leaky abstraction. But it's a good point about how even testers apply easy-way thinking to problems which really deserve genuine and rigorous thinking that shows intelligence and care for the problems they are solving.

2) Follow the Spec

   Detail: As a junior tester I was given a set of instructions and mechanically followed them to completion. If I needed to print something on 10 different types of envelopes from 10 different printers and measure the printed indicia with a special ruler that measures the line spacing and placement of elements on the envelope, then that is what I did. I checked a box in a spreadsheet for each one and moved to the next thing like a machine. That is how I was initially taught to be a tester. Have a spreadsheet, run through it, check the boxes and log anything that was outside of

the expected result. Eventually I was allowed to write my own spreadsheets for areas of the product that needed to be tested. I read the requirements document, took lines from it and pasted them into a spreadsheet and prepended the word “Verify” in front of them. I might have added a bit of formatting to make it look nice and readable, to show which contexts the tests were to be run in, and perhaps my iteration or time estimates. Run through that, check the boxes, log the results- boom, I’m done. “Wow, Testing is kind of easy”, I thought. So I applied this mechanically over quite a few projects before I started reading about Software Testing from people like Cem Kaner, James Bach and Michael Bolton. Once I realized how much more there was to testing it became less easy, more challenging and more fun. But the point here is that I think I might have discovered a lot of issues or been more effective as a tester if I had been given more than just a hammer to do my job. There is a saying that if all you have is a hammer, everything looks like nails. I wound up using the same mental framework for each job and thus probably was not as effective as if I knew that some nails are different. Some nails require a pin hammer or some come in strips and go into a semi-automatic nail gun. Whoa, wait what is a nail gun? You mean I can fire these nails faster if the job allows for it? But wait- how to decide if the job allows for it? And if it allows for it are there different kinds?- These are the types of questions I was not being trained to ask and did not know to ask. But the easy way out rarely requires you to do much thinking and most definitely not Systems Thinking.

2.1.5 The cure can be worse than the disease

When I first witnessed a large layoff at a company I worked at, I was in a bit of shock. Admittedly I was relieved at the same time because I was not one of the people who got the axe, but still shocked and a bit unnerved. I was relatively new (had not even been there a full year), and so I had not heard much about any turmoil or upcoming layoff. I think people were still trying to let me be a happy new employee, full of cheer and spark. It ate at me for weeks after the event. It always pulled at me, begged me to understand why and to know what businesses looked at to make such a drastic decision. After a while I started to have hallway conversations or after work drinks where it would come up. I was prying. What I eventually found out was that we just had a bad few quarters and needed to adjust some figures to make our bottom line look better for investors. Cold. But that’s the world we live in. It’s illogical and driven by money for most people and especially in the business world. However- what unfolded after that event I don’t think was expected. Slowly but surely attrition kicked in. Folks started leaving for “greener grass”, and I mean really good people too. These were people that I respected and looked up to, wanted to learn from. I’m not sure what did it but even I made the decision to jump ship as they say. Eventually the company lost a lot of good people that I don’t think they anticipated losing and wound up having to spend a lot of money in slowing down projects and re-hiring and re-training people. This is a prime example of how the cure to fix their bottom line numbers disease wound up costing them even more money in the long run, and with no guarantee the sales and bookings would improve to avoid the same situation before they laid off people.

As a Tester we can also see this in our daily lives, and if we can see it we can try to avoid it. Admittedly I had trouble seeing how in my day-to-day tasks as a Tester how I was falling victim to this law in action. I was applying “cures” to situations where I needed Test Plans, Test Cases, Test Setup, Test Execution, Test Design, etc., where it led me to being in a worse place in the long run and thus did not solve the systemic issue I was dealing with. Then I re-read what I just wrote and it hit me like a ton of bricks. Every one of those things I do to try and improve the information I generate and the quality of that information so that in turn the quality of the product we create or the expectation of quality as defined by those that matter in our system match or align with the information we generate. Putting on our Systems Thinking cap, we start to ask questions like “What if one of those things I did led me to generate information that was not of high quality and led developers and managers to re-plan and re-work solutions which were done based on faulty information because of my actions?” I just cost us time and money and possibly my job. So what this tells me is that you really must understand the context of the situation, the relationship of your artifacts to the problem and how in a sustainable way you can produce your artifacts such that it is affordable to provide them and they are of high quality and reliability. Additionally you can look beyond your day-to-day tasks and look at your team to spot this law in play. For example if you are on a team and

that team decides to implement a new process in order to reduce distractions or improve efficiency- I urge you to ask how they measure the success of such a change. What does success look like for this change, and how can it not make things worse in the future? Without knowing this, you are setting yourself and your team up to be treating only symptoms and not the systemic issues that cause them. A Systems Thinker tries to solve the systemic problem, and asks why they are doing something before they ask how they will do it.

2.1.6 Faster is slower

Have you ever been in one of those startup agile environments where everyone is like, “Go fast!”, “Go faster!”, “WE GO LIGHTSPEED HERE!”? How did that work for you? How did it work for them? Personally I feel those are the places that burn people out and create a revolving door of engineers and thus wind up having to move frantic, not necessarily fast. Perhaps cultures like this should try adopting a new way with a slogan such as: “We move methodically!” or even “We move as fast as we must to do things intelligently!” would be sufficient for me. I guess it doesn’t roll off the tongue as easily though. So instead of growing at an optimal rate, they opt for trying to grow at the fastest possible growth rate as selfishly desired by their upper management and investors. This leads to churn and usually leads to creating perceived bottlenecks in other parts of the system who were not properly tuned or equipped to be part of such a fast moving system. An analogy to describe this law would be something like putting a turbo engine into a stock mini-van. What do you think will happen? Well- none of the other parts are meant to handle the excess of air pumped into the engine cylinders and now you blew up your mini-van trying to break land speed records getting your kids to soccer practice. You might have been going faster for a while, but now you are dead still on the side of the road.

Now as applied to the role of a Tester, this should be brutally familiar and obvious to any Tester who has been in the industry for even just a few years. When was the last time you rushed a job, and how did it turn out? Did you get away with it, or did it come back to bite you and the business you work for? My guess would be the latter and not the former. Testing is not something that should be rushed. If you rush it, you will make mistakes. If you are catching yourself rushing any part of your testing process, you should stop and ask why. What led you to mode of rushing? I don’t mean to conflate “rush” with “rapid”. I believe that rapid software testing [6] can be done effectively, carefully and meaningfully as demonstrated by the likes of James Bach and his RST/CDT practitioners. I’m talking about the careless kind of rushing where you are cutting corners in your testing or buying into snake oil tools that promise to save you time and maximize your coverage without you actually vetting out such claims. In another aspect of your Tester life, you may be witness to developers and managers doing the same thing. Quick fixes, code-smells, hacks, etc. You need to gather these people into a room when you see this stuff, and ask why it’s being done and if the quick fix scenario has been played out past the snapshot of time the decision was made in. This is what Systems Thinkers do.

2.1.7 Cause and effect are not always closely related in time and space

An “RCA” document is a good candidate to showcase this concept. For those who don’t know what “RCA” stands for, it means “Root Cause Analysis” and is usually carried out like your favorite murder mystery show. In the episode you have someone (Ops) who discovers a crime (production failure) has been committed, and then a Pathologist (some Dev Manager maybe) sets out to find the cause of death (production defect) and understand the cause in order to determine if it was natural or foul play. Now the difference here is maybe that in our software business the RCA takes either natural causes or foul play and attempts to suggest a mitigation solution, so that such an event does not occur again. Many root cause analysis documents are prime examples of how we fail at understanding that cause and effect are not always closely related in time and space. They are usually shallow analysis of the situation with convenient conclusions meant to shut people up or appease them which then shuts them up. What makes it even more interesting is that the person writing the RCA likely has some reason to not be too hard on themselves for self-preservation and self-interest reasons.

2.1.8 Small changes can produce big results -- but the areas of highest leverage are often the least obvious

You may be familiar with something called the Butterfly Effect. For those who are not- it is basically the idea that the flap of a butterfly's wings in a far off place can be the cause for a thunderstorm hundreds of miles away. You might also be familiar with how crimes are solved from watching your favorite crime tv show where the big break in the case is led to by some minute detail in where most common observers who examine the crime scene or analyze the evidence would have never made the connection. These examples help me explain this law by showing you that a small detail or small change can result in a big finding or big effect. Not only can they produce big results, they are not obvious to just any observer.

As a Tester you may be surprised by how much impact you can have on quality and scope of work or the process of work being done by looking for not just small but also less obvious avenues and ideas. From personal experience there was at least once in my Testing career where I literally stopped an entire team of Engineers from embarking on an expensive journey to build something by asking a simple question that none of them could answer and no one ever thought to ask which was, “Why are we building this?”. Stunned that their Tester asked them this, yet unable to give a substantial answer, the question was raised to management, and it was found that the project had started through improper channels and should not have been started not only for the lack of approval, but for the lack of its value add and reason to exist. So you see, you don’t always have to take obvious or traditional approaches to Testing problems. Sometimes the problem is not really a problem even if you ask the right questions. Other times it may be that you can buy or rent a tool as opposed to building it, recruit developers or the front-desk person to do some testing, try out interns for a summer, etc. Don’t ever limit yourself to obvious/traditional and look for outside-the-box ways to achieve big results.

2.1.9 You can have your cake and eat it too -- but not all at once

At first it might not be clear how to see the relevance of this statement, but there are some examples we can apply this to that are likely familiar to you as a Tester. Remember- This is about “either/or” and “black/white” thinking and realizing that most of the statements or decisions made in these situations are based on thinking about them in a moment of time versus what is possible through moments of time. Think about all of the instances in your career as a tester or in your current role where you have been faced with dilemmas like:

1) You cannot increase test coverage without increasing your schedule.
2) You cannot do “good” testing without documenting your testing.
3) You cannot build automation while upkeeping your manual testing.
4) You cannot increase your quality without increasing your cost.

These are just a few examples where I ask you to exchange “cannot” with “can” and work through the steps of how you can get there eventually. There are many more examples from different points of view of the different people on your team where, even as a Tester, you can help them get past this black and white, static thinking style. As a Tester who is equipped with an understanding of this principle, you can help educate others on your team as well as yourself to not sacrifice one solution for another.

2.1.10 Dividing an elephant in half does not produce two small elephants

Think about testing a web application that has an address book for example and how you approach it. It’s a hosted service with an interface and is accessible through the a browser on a user’s device. Do you create a test plan for the database, service API, and GUI separately? Do you then create a test plan for the device contexts separately? Do you also create a plan for the performance and security of each layer discretely? Perhaps you do and that’s OK. But if you failed to account for and plan for when they all live and harmonize together, then you have failed to think about the problem in a holistic way and will likely miss a large part of delivering a successful product that meets the expectations of the users. The successful software product is not each component separately. You should not have been only concerned with the individual parts of the system in your testing and planning without considering the full

end-to-end when all parts are working together. End users don't really care if you are using Postgres or MySQL. They care that they can enter in contacts to their address book in your application. The culmination of each layer and technology creates an address book web application and not one single part alone is the address book. Looking at this whole system is the process of synthesis. Synthesis is the opposite of analysis and is used in systems thinking to understand and discuss systems problems from the point of view of the bigger picture. The quality of the whole system is an address book.

2.1.11 There is no blame

When it comes to blame, we as testers tend to catch it for things like production defects and missed deadlines. Most organizations I have worked with have had at least one or many peers, managers, and executives, that have made comments about how their QA team doesn't know how to test effectively with such defects arising in production. They say or ask things like, "How could you miss such an obvious issue?" or "We have poor quality because of how incompetent our QA team is". Another classic is that teams are behind schedule because of how slow the QA team is. It's frustrating and concerning because it's toxic and counterproductive. Not to mention they are confusing testing software with assuring quality. What is worse is that they don't stop to think if they too had a hand in the creation of such a perceived quality issue or bottle neck. I'm fairly certain that no one walks into their Tester jobs and says to themselves that they are going to purposely miss important areas of testing. I have also never worked with any Tester who committed to intentionally working slower so that we would miss a deadline. What I have seen though are testers who don't speak up when they see too much work being approved and too much work with a fuzzy definition of done being started on. It's not that they should blame the project lead or engineering manager for not defining this though. We must not look towards individual people or groups for poor quality or unhappy surprises. As stated by Dr. Deming, "~94% of all issues belong to the system" (Out of the Crisis, page 315).

3 Conclusion

The most important thing about Systems Thinking for me has been the reification of my existing thoughts and approach to problems. That and the vocabulary it has provided me. Before I picked up my first book (The Art of Systems Thinking) and moved onto my second book (Intro To Systems Thinking), I had many thoughts about exactly what these books explained. They gave me examples to show my managers and peers. Watching videos of Russ Ackoff led me to Senge, which led me to Deming. My view of the world has become much more mature, and I am more aware of my place and potential impact on the organizations I work for. I am the butterfly, and there are effects when my wings flap. If I can better understand these effects, then I can better influence the system in the ways that I hope to see it become someday. This is the whole point for me in understanding Systems Thinking better. I want to do good and be aware of my actions. I want to work with people who also want to do good but are willing to do some rigorous thinking about what that entails, not just have good intentions. After years of floating from person to person, team to team, I realized that I could bring people together and appreciated my role as a Tester even more. Ever since I realized that I could influence multiple teams, I began to experiment with planting seeds in each of them to see what happened. I began to rally people from different teams that otherwise might have never spoken to each other. Once I saw that there was positive change coming from my ability to spread messages and join people together, I realized that I had some influence of the system. I just never realized it was an actual discipline until late in my career. I hope that testers who are early in their career will find this and begin sooner than I did. I also hope testers late in their career find this and have their thoughts reified as well.

References and Sources

[1] https://en.wikipedia.org/wiki/Systems_thinking

[2] https://en.wikipedia.org/wiki/System_dynamics

[3] https://en.wikipedia.org/wiki/The_Fifth_Discipline

[4] http://www.bretlsimmons.com/2010-03/behavior-grows-better-before-it-grows-worse/

[5] http://www.bretlsimmons.com/2010-04/the-cure-is-worse-than-the-disease/

[6] http://www.satisfice.com/info_rst.shtml - James Bach

Publication: The Art of Systems Thinking - Joseph O'Conner, 1997

Publication: Introduction To Systems Thinking - Gerald Weinberg, 1975

[7] Russ Ackoff - Youtube Video Ref. (https://www.youtube.com/watch?v=OqEeIG8aPPk)

Edward Deming - Youtube Video Ref. (https://www.youtube.com/watch?v=tsF-8u-V4j4)

http://www.4grantwriters.com/Peter_Senge_The_Fifth_Discipline_1_1_.pdf