Hard Lessons About Soft Skills -- Understanding the Psyche of the Software Tester

Marlena Compton
mcompton@mozilla.com

Gordon Shippey, MA, LAPC
gshippey@gmail.com

Abstract

Testers are often characterized as the conscience of a project. This session will dive deeply and uncomfortably into the psyche of software testing and those of us who claim it as our profession. Attitudes that are deeply embedded in our testing culture and their effectiveness will be examined:

1. Our bug finding mandate is a wrecking ball, which we aim and swing at will
2. We haven't done our jobs until the developers are crying and the ship date has been extended by months
3. Developers live to write bad code and should be eyed cautiously at all times
4. Crunch time on a project is our signal to be as harsh and unforgiving as possible to everyone else on the software team.

The positive role that emotions can play in testing is also discussed. Crucial conversations are also introduced to help testers understand how to argue their point without bullying. This presentation will show why testers must have and act with conscience and have an emotional fluency in order to test software effectively. We will show that despite being erroneously called soft skills, the effective care and handling of emotions is the hardest skill in testing.

Biographies

Marlena Compton has been testing software since 2007 and has an innate talent for making developers feel angry, inadequate and ashamed. After noticing that happy developers make better software, she embarked on a journey to learn how to be an effective tester and help the developers feel supported, at the same time. This led her to research psychology and communication skills as they apply to testing with friend and licensed counselor Gordon Shippey.

Gordon Shippey has completed degrees at Emory University, the Georgia Institute of Technology, and Argosy University. A Licensed Associate Professional Counselor in the state of Georgia, Gordon provides psychotherapy to individuals, couples, and families. Gordon is particularly interested in work/life balance and organizational psychology.

1. Introduction

One of the main ideas in the Agile Manifesto is that individuals and interactions are valued over processes and tools. This has brought a new dimension to software development especially now that ideas from agile software development have become mainstream. People are encouraged to talk and work more closely with each other. Whereas before, everyone had their own area of working with the software, people on software teams engaging in agile software development have to figure out a way to work closely with each other. Office plans suggested for agile teams include the removal of cubicle walls which means that if people are "faking it 'til they make it" in trying to get along with each other since there is no place to hide.

For testers, it means that if a tester uses software and hates it, developers can now literally hear the screams. If a developer talks about putting off bug fixes in a standup, and it makes a tester cringe, everyone on the team sees it. Such knee-jerk, emotional reactions are important in testing. It is important for a tester to develop a sense of how an application will fail. Unfortunately, the reactions spurred by tester instincts can be also be damaging for a team's morale.

While it is extremely easy to make these mistakes and perpetuate bad attitudes (the authors have first-hand experience), the skills needed to avoid them are not easy to come by naturally or even easy to learn. Examining this failure in tester communication and attitude is difficult and uncomfortable, but necessary. Some of these attitudes are ugly and therefore may be denied or resisted, but ignoring them allows them to linger on. The authors hope that testers will see this paper as an opportunity for self-reflection and growth instead of a simplistic attempt at painting testers as villains.

There are few places in the testing literature to turn for answers if a tester is having problems communicating with developers on her team or if she is feeling ostracized because of communication mistakes she has made. On the contrary, the literature on skills required for software testing is well established. Much has been written about techniques that can be used to analyze software. Testers have any number of resources that they can choose from, such as *How to Break Software* or *A Practitioner's Guide to Software Test Case Design* if they wish to improve their test case design or to learn how to break software in any number of ways. We are, however, missing out on a foundational skill in software testing. Testers must learn how to find the balance on their team between providing criticism of the software but remaining constructive in the process.

This paper will borrow from recent writing on workplace interaction and place it within the context of software and software testing. Books such as Dr. Robert Sutton's *The No-Asshole Rule: Building a Civilized Workplace and Surviving One that Isn't*, and *Crucial Conversations: Tools for Talking When the Stakes are High*, by Kerry Patterson et al. is used to describe the social and emotional skills necessary for software testers to do their work while remaining constructive members of the software team.

We will begin in Section 2 by discussing why a work environment that is psychologically safe results in better software. With the need for a safe workplace established, we will move into describing negative behavior and what it looks like when placed in the context of software testing in Section 3. This behavior forms the basis of what is described in Section 4 as "The Tester's Paradox." "The Tester's Paradox" explains how testers are set up to fail at communicating effectively on any software team. With the need for better communication skills from testers established, the next sections describe in a practical manner, fundamental skills testers can use to improve their communication foundation. Crucial conversations are

introduced in Section 5 as they apply to software testing. The physical barrier to effective conversations is examined in Section 6. This is followed in Section 7 by methods for creating physical and conversational space in software testing. Section 8 links the concept of mindfulness with software testing as a method for working more effectively with our emotions when we are testing. Finally, Section 9 explains how we can reach out when things go wrong.

2. Why psychological safety means better software

Building software requires creative thinking even if the budget is tight and the software must be turned around quickly. In their book, *Artful Making*, Rob Austin and Lee Devin write about how the creative work in software takes place on an edge. It requires a team to stretch itself uncomfortably and should allow for frequent failure. In such an atmosphere, having a high overhead for failure will block creativity necessary for making software. Working on the edge cannot include fear or the quantification of work, but must include practice and self-knowledge.

It is important for the members of a software team to be able to argue constructively in order to work out complex problems creatively. Unfortunately, it is easy for arguments over software to descend into personal insults and ill will. This damages a project's safe atmosphere and means that team members will be less likely to discuss their ideas.

People are damaged in unsafe workplaces. Sutton describes this damage in *The No Asshole Rule*. The immediate effect on a person can include heightened anxiety, heightened depression and trouble concentrating. Negative interactions are felt five times more strongly than positive interactions. They affect not only the target of the insult, but also have a ripple effect on the rest of the team. Teams experiencing high levels of negative behavior become fear-driven, with everyone looking over their shoulder.

A good indicator of safety is what happens when people make mistakes. In his book, *Good Boss, Bad Boss,* Sutton suggests that failure can be handled in any of three ways. If a tester keeps a wall of shame on their cubicle wall showing bugs they found particularly offensive and the name of the developer responsible, the tester is engaging in a common and very destructive reaction to failure described by Sutton as “the Silicon Valley standard.” The tester is simultaneously remembering, blaming and shaming developers for bugs assigned to them. Another less destructive but undesirable method of handling failure is forgiving and forgetting. This is where the line is drawn between being effective as testers and being “the push over” on a team. It might make the developers feel better momentarily if a tester allows them to ignore bugs that are minor. This can especially be the case if a deadline is close and the testing schedule is extremely tight. It is essential to remember that a tester’s job is to report when things go wrong.

Rather than being overly harsh or too forgiving, a tester can, as part of their team, practice what Sutton calls, “forgive and remember.” If it appears that a developer has not written code that matches a specification or user story, it is worth asking the question of whether the team communicated that user story effectively. Since the team acknowledges the mistake, the team can now learn from the problem. No fingers are pointed. This can be a lesson for developers as well. If a tester has failed to find major problems in a release, the tester’s team should acknowledge the failure and do what they can, together,

to understand what went wrong. This turns the failure into a learning opportunity for everyone rather than isolating the blame on one person.

One cost of a work environment based on fear is the increased time spent on finger pointing instead of fixing problems or even reporting them. In a study conducted at Harvard Medical School, nurses working in units where they felt psychologically safe reported much larger numbers of errors. In these units, mistakes were seen as natural and reporting them as just part of a normal day's work. In units where nurses did not feel safe, the number of errors reported was much lower because the nurses were afraid of the repercussions that would result, not because they were less error prone.

One of the authors of this paper has seen the effect of a fear driven environment when she worked with developers who had bonus pay docked every time a bug was found in their code. This led to discussions in which developers would “bargain” with testers to try and do anything they could to get out of having a bug filed against their code. This was in stark contrast to the atmosphere at a different employer where developers would seek out the tester and tell her their suspicions about how their code might break. This opened the door to constructive dialog between the developers and the tester.

3. Negative behavior in the context of testing

Before we delve into the ways testers can strengthen their communication skills, we will examine negative behavior and what it looks like in the context of testing. This is not intended as an insult to testers but is merely intended to give testers an opportunity to reflect on how they may be sabotaging their own best efforts to contribute to discussion on their software team.

Hurtful or negative behavior is familiar to everyone. No one escapes life without experiencing insults, bullies and toxic relationships. Sutton examines this behavior as it applies to the workplace in *The No Asshole Rule*. He points out that the damage suffered is not large and dramatic but rather through an accumulation of many small, demeaning acts. His list of negative behaviors include:

- Personal insults
- Threats and intimidation
- Sarcastic jokes and teasing as an insult delivery system
- Overly dramatic emails containing harsh language such as words written in capital letters
- Public shaming
- Interruptions
- Nasty looks

Bugs are easily used as insult delivery systems or for delivering our own personal judgments about functionality. Testers should be sensitive to how they write bug reports since they can last for a long time and are often read in many different contexts.

If testers feel they are not being listened to, they may result to public shaming such as pointing fingers in a standup. Threats delivered in emails are an unfortunately easy way to amplify a message we may feel is not being heard. One of the authors is particularly guilty of using nasty looks to let developers know when she was displeased by a feature or decision.

Finding most of this behavior in almost any software team is easy, and it is not just testers who participate. Indeed, the culture of software has thrived on sarcasm and jokes based on insults such as the humor found in popular web comic, *The Oatmeal*. Testers are no exception. In fact, tester hazing of developers is a typical source of "tester humor." It can be difficult, however, to know when the humor ends for someone, and the danger to psychological safety begins.

4. The Tester's Paradox

In *The No Asshole Rule*, Sutton points out that anyone can be a jerk given the right working conditions. To place negative tester behavior in context, it is necessary to consider working conditions for the average tester. If the tester is on a team using a waterfall approach or a team lacking the benefit of continuous builds the tester may spend significant amounts of time being blocked by problems further upstream in the software process over which they have no control. When the work finally reaches the tester, it is often already late and over budget. This means the tester will be critiquing software from a team whose nerves are already frayed.

If a tester is working on an agile team with continuous builds, they will have more control and autonomy over when they can do their testing, however, they may not be sensitive to the fact that the developers are working on their creative edge. This causes further sensitivity on a team.

While some testers are fortunate enough to work on a team with a low developer to tester ratio, many testers work on much smaller teams or even as the solo tester. When a tester is working alone or lacks enough support, it can be challenging to be heard. In environments where developers are encouraged to be "cowboys" it is even more difficult for testers to have a balanced relationship with the rest of the team. This can lead to the tester feeling ostracized and powerless. Even when testers work on larger teams, they may live in a silo with very little opportunity for communication with their developer counterparts. Testers can be separated from the development team by being in a different building, a different location or a different time zone. In this case, the tester lacks a connection to the rest of their team making it a struggle to feel that they have any voice at all.

Aside from looking at the environment in which testers work, it is worth considering what testers do. Testers must critique an application. This includes asking probing questions, finding ways to break an application and reporting bugs about the breakage. In the case of user interface testing, testers must report if something about the user interface makes it awkward to use. The basis of all of these activities is finding information about how an application is broken, not working as intended or confusing to a user. All of this information is negative.

The reality of testing is that the task of producing negative information about an application is given to the person on a team who may feel ostracized, powerless and alone. This creates an unfortunate situation. Testers are placed in the most emotionally precarious position of anyone on a software team and blamed if they don't handle the situation correctly every time. This is the tester's paradox. Just as each and

every negative interaction packs five times the punch of a positive one, a study has shown that it takes approximately 5 interactions to get over one bad interaction. 100% perfection is not a fair request to make of any human being, including software testers.

5. Crucial Conversations in Testing

In, *Crucial Conversations*, *Tools for Talking when the Stakes are High*, Patterson defines a crucial conversation as, "a discussion between two or more people where 1) the stakes are high 2) opinions vary and 3) emotions run strong. Software testers face these types of conversations every day in a myriad of different situations:

- Discussing whether or not a discovery is a feature working as designed or a bug
- Participating in a discussion about which bugs should be fixed before a release
- Asking whether or not it is ok to ship a release
- Discussing why a bug is reproducible in one environment and not another
- Discussing the flexibility of release dates
- Asking a developer if a feature is ready, especially if that feature is late

Crucial conversations can happen at any time, and in the compressed, intense nature of agile iterations, they happen frequently and are most often spontaneous. Opportunities for crucial conversations in an agile project include post-standup conversations, retrospectives, and any daily interaction. Agile office spaces are designed to encourage team interaction and discussion. Sitting at a desk on an agile team frequently means that there are no walls and the rate of discussion is much higher. The purpose of this is to create and maintain a pool of shared meaning on the team. While this is a great way for testers to be more integrated with their team, it also creates a challenge for testers, along with everyone else on the team, to have a net positive effect in the many conversations they will have.

The beginning of any crucial conversation is, what Patterson calls, "starting with heart." When approaching any conversation, it is necessary to know what you want from it. This can be as simple as wanting to know if we have gotten the correct meaning from a story or as complicated as having to answer the question, "is this software ready to ship." Starting with heart is important because no matter where the conversation goes or how messy it gets, it helps us to retain focus on what we wanted in the first place. If we don't know what we want from a conversation, a simple, "let me get back to you on that," should be acceptable. If it is not, someone in the conversation will begin with an unfair advantage and creative safety will likely be compromised.

There are a few questions to ask before engaging in a crucial conversation:

1. What do you really want for you?
2. What do you really want for the team?
3. What do you really want for the software?

Once you've answered these questions about what you want, it is also worth asking yourself how you would behave if you really wanted these results. This will help you to focus on your goal and get your brain ready for the conversation. It is worth noting that if the only good reason you can come up with for having or staying in conversation is to one-up someone else or to "win," it is time to reassess your true goal.
Sometimes crucial conversations take the form of arguing. When a developer insists that software "works in her environment" or a tester is faced with answering the question, "is this software ready to ship."
In his book, *Good Boss, Bad Boss*, Sutton lists a few guidelines for having a constructive argument:

- “Don’t begin the fight until everyone understands the challenge or problem at hand.” This is in the same vein as starting with heart. In software, it is particularly important to know that everyone is arguing about the same problem. Even if the discussion is about “feature x”, the argument might be going in the wrong direction because each person is arguing about a different nuance of “feature x.”
- “Don’t argue while generating ideas or solutions - make it safe for people to suggest crazy or controversial ideas.” For testers, this means holding back from torpedoing an idea too early or in the wrong forum. Common forums for brainstorming include developer “spikes” or having a “lab week” where everyone works on something new and experimental.
- “Once the argument is resolved, make sure that the conflict and criticism ceases. It is time to develop and implement the agreed upon ideas. If compromise is reached on bugs that will be fixed for a release, it is not helpful to continue complaining about the ones that have to wait until a bug fix release.
- “After the fight is over, do some backstage work. Soothe those who feel personally attacked and whose ideas were shot down.” it is important to know when to reach out to a developer or product manager who may feel beaten by the QA stick. Reaching out is addressed in it is own section of this paper.
- “If people turn nasty, take a time-out and ask them to turn off the venom. Pay special attention to comedians who deliver devastating insults via jokes and teasing.” Although this guideline is written for a manager, it is important, for us, as testers, to know when our stress level and a team’s stress level has hit it is limit. Cultivating this awareness and using different approaches to create space is addressed in the next section.

6. The Physical Challenge of Crucial Conversations

There are good reasons why a crucial conversation is challenging. The primary reason is biology. The human brain was not designed for having successful conversations in stressful situations. In any stressful situation, the brain will send us signals to do one of three things: fight, flight or freeze. The moment we perceive that a conversation has turned crucial, the flow of blood to our brain is decreased while the flow of blood to our arms and legs is increased. Instead of engaging in reasoning, the brain physically triggers us to try and “win” the conversation or to end it.

Aside from biology’s attempts to sabotage a high stakes conversation, our own skills can also lead us astray. Negotiating stressful conversations is not taught in most academic programs. Most people learn conversational skills from their parents who may or may not have had a great skill set. Because these skills are not intuitive, a lack of training can render a person defenseless.

Despite human biology and despite a lack of training, good intentions help set the foundation for a good discussion. Bad intentions set a course of disaster for a conversation. If the conversation is high stakes, the effects can be devastating. We must always be aware of the reasons for having the conversation as well as the other party’s reasons for having the conversation.

7. Creating Space in Crucial Conversations

Part of knowing how to successfully navigate crucial conversations involves developing an awareness of when we need some space in a conversation. Physical signals come from our body responding to stress. Each person’s body responds to stress differently. For some people, they will feel stress in their

shoulders while others will feel it in their chest. Cultivating this self-awareness means that when we are faced with a stressful decision we will be able to recognize that and give ourselves the space we need to make the decision.

A conversational sign that it is time to make some space is what Patterson describes as the "the sucker's choice." A tester who finds something that will negatively impact customers and but is told that it is a feature or "working as designed" faces the sucker's choice. A classic sign of the sucker's choice is being presented with two choices that are equally bad as the only two options. On the one hand, if the tester agrees, then software will be released that might have a negative impact on the customer. On the other hand, it is implied that the tester is being a nuisance to the team by suggesting that there is a problem at all. "Those offering up a Sucker's choice," writes Patterson, "either don't think of a third (and healthy) option - in which case it is an honest but tragic mistake - or setup the false dichotomy as a way of justifying their unattractive actions."

Patterson offers a strategy for setting up new choices in these situations.

- "First, clarify what you really want." In the case of our example, we don't want to release software that will negatively impact the customer.
- "Second, clarify what you really don't want." We don't want to waste anyone's time by fixing something that won't matter to the customer.
- "Third, present your brain with a more complex problem...combine the two into an and question." In our case, we might say, "I don't want to release software that will negatively impact the customer and really don't want to waste anyone's time with a fix that won't matter to our customer.

In our example, the conversation could be shifted finding out how important the potential problem will be to customers.

8. Mindfulness in Testing

Mindfulness is a concept found in many of the world's religions and has made its way into modern psychology through mindfulness based stress reduction. According to the Bob Stahl in *A Mindfulness Based Stress Reduction Workbook*, It is a practice of, "being fully aware in the present moment." This involves developing our ability to acknowledge our emotions and suspend judgment in what we observe. Mindfulness training involves learning how to acknowledge our emotions in much the same way we acknowledge what we see and hear without allowing them to completely take over our actions.

As an example, a tester finds that a web page he is testing does not allow navigation with the tab key. "I should be able to hit tab," he thinks. Since the tester is practicing mindfulness, he acknowledges that he is frustrated and that this frustration is his personal judgment. If he is doing exploratory testing, he begins thinking more objectively about the frustration and what it was about the application that caused him to feel that way.

Anytime we preface a statement with "should," or "I feel that" we are about to express a judgment. These have their place in testing, but it is important to understand the difference between making judgments and observations. Having an awareness of the emotions leading us to making judgments can also give us clues about our own inner rules. Knowing about our inner rules, especially for testing, allows us to break out of them.

While it is important to be able to observe our emotions when we're using an application, it is also important to have some outside perspective on our emotions and to be able to balance them with the goals and priorities of a software team. Learning about mindfulness based stress reduction is one way to learn how to acknowledge our emotions without letting them get the better of us in our testing.

9. Recover and Repair

It is inevitable that as testers we will make mistakes and steps on toes in the course of our work. We are, after all, human. Mistakes happen. Sometimes we are overly harsh when we critique work that took someone a long time or we judge a feature to be lacking before the developer has had a chance to finish it. It is important to know how to reach out and work on repairing the damage with what relationship expert John Gottman describes as, "repair attempts."

Sutton outlines steps for making an effective apology when it is needed.

1. Take the blame fully. In the case of judging to harshly, we might say, "I'm sorry that I was so harsh in my critique of your feature."
2. Take control over what you can. In the case of our example:
3. Explain what you've learned
4. Communicate what you will do differently: "I will be more mindful of my critique in the future."

Testers are paid to find flaws and in doing so, we will inevitably "break some hearts." It is important however, to keep in mind that bad feelings can linger beyond the current release. A study pointed to in the "No Asshole Rule" claims that one negative interaction equals 5 positive ones. It is, therefore, worth putting forth the effort to find the 5 good interactions to offset the times when a negative one happens. This corroborates what Gottman says about the effectiveness of repair attempts.

The effectiveness of repair attempts, says Gottman, has less to do with the format of the repair attempt and more to do with, "how much emotional money in the bank you [sic] have with that person." It has, indeed, been our experience that positive reinforcement can go a long way towards a better relationship within a team. If a developer produces a feature that has problems, but is, nonetheless, an exciting feature, it is worth telling the developer that despite the problems you expect them to fix, we are excited about the feature.

10. Conclusion

The consequence for software testers of bringing agile into the mainstream is that we can no longer afford to ostracize ourselves on a software team and expect to succeed at testing. Bad tester behavior

can shake a software team's confidence and reinforce the "us vs. them" mentality between testers and developers. This rift will, eventually fracture a software team.

We have shown why it is important for software testers to contribute to a psychologically safe workplace environment and which types of tester behavior are counter-productive to this effort. In exploring the "tester's paradox" we have shown the challenge faced by testers in overcoming negative tester behavior patterns. To help testers re-frame their interactions on a team in a more positive way, we've applied the crucial conversations mindset and what can go wrong with them. We've offered guidelines for arguing constructively and introduced mindfulness as a way to clarify our emotions and judgments in testing. For those inevitable times when things might go wrong we've suggested ways to recover and repair relationships. It is the sincere hope of the authors that the ideas in this paper assist software testers in their journey of self-reflection.

References

Austin, Rob and Lee Devin. 2003. *Artful Making: What Managers Need to Know About How Artists Work.* Financial Times.

Beck, Kent and others. 2001. *The Agile Manifesto.* http://agilemanifesto.org/

Copeland, Lee. 2004. *A Practitioner's Guide to Software Test Design*. Artech House.

Gottman, John. 2010. *Relationship Repair that Works.* The Gottman Institute. http://www.youtube.com/watch?v=SqPvgDYmJnY

Patterson, Kerry and others. 2002. *Crucial Conversations: Tools for Talking When Stakes Are High.* McGraw-Hill.

Stahl, Bob and Elisha Goldstein. 2010. *A Mindfulness-Based Stress Reduction Workbook.* New Harbinger Publications.

Sutton, Robert I. 2010. *The No Asshole Rule: Building a Civilized Workplace and Surviving One That Isn't*. Business Plus.

Sutton, Robert I. 2010. *Good Boss, Bad Boss: How to Be the Best... and Learn from the Worst*. Business Plus.

Whittaker, James. 2002. *How to Break Software: A Practical Guide to Testing.* Addison-Wesley.