When You're Evil: Building a Credibility-based Relationship with Developers

By Curtis Pettit

Curtis.Pettit@gmail.com

Draft 6
7/11/16

Intro

Software is complicated, in most cases, it takes several people working together to create it. This paper focuses on the relationship between you the software tester and the developers you work with. It's a relationship that's often strained by conflicting priorities, egos, and missteps. It doesn't have to be that way. Ultimately your goals of delivering good software are the same -- you are here to make them look good.

As a tester, every interaction you have with people on a project is a credibility transaction. You start with a tiny pile of credibility, when you meet someone, like chips at a poker table. This starting credibility is mostly based on those whom they have worked with before; small and almost entirely out of your control. Every time you file a bug, ask a question, or make a statement, you are wagering some of your credibility that what you are saying is valuable. How much you need to wager will depend on how likely they judge that outcome to be. If you don't have enough credibility to cover the wager, you will just be ignored.

I was in a sprint planning meeting once. The project was just starting out… and the team just assembled that day. While we were estimating the backlog, we came to a user story that was moderately complex and I estimated a "5". The dev lead turned to me and said, "If this is a 5, we will never finish this project on time." By this point in my career, I was savvy enough not to say, "Duh! That's what I told you an hour ago." Instead, I actually said something like, "Ok, I can use this as a baseline for a 3 if you like since it's a relative scale, but that doesn't mean that the story is any easier. " I was ignored, and when the project eventually ran over it was surprising only to him.

Most of the papers you read about software testing will be focus on improving the practice of testing, helping you win more often. Either by increasing the odds of winning or making more

bets with the same odds. Having a high credibility relationship with the developers will help with that, but this paper focus on maximizing the result of winning and minimizing the result of losing. Over the past couple of decades social psychology has learned a lot about how to influence and motivate people. If we are going to build a relationship with developers let's take a look at what make them tick.

Quick Tips

Probably the most popular book on the subject of getting people to like you is Dale Carnegie's How To Win Friends and Influence people. In it he wrote "one of the simplest, most obvious and most important ways of gaining good will was by remembering names" (Carnegie 1981) I am admittedly bad at this. At professional events, I typically wear the previous year's conference badge on my back so people can identify me from behind. I don't remember other's names well, so I don't expect them to remember mine. When I'm working with a new set of people, I'll review the list of people invited to meetings so that I have some names I can associate with faces. I'll check the company's social network or HR system looking for pictures of them. Sometimes you can find them on external social media sites, too. I'm not stalking them, at this point, just trying to associate names and faces. Make sure you get the correct pronunciation. We tend to work with a lot of people from exotic locations, and people take their names very seriously. "The name sets the individual apart; it makes him or her unique among all others. The information we are imparting or the request we are making takes on a special importance when we approach the situation with the name of the individual." (Carnegie 1981) In college, it took me over a dozen tries to say my Greek roommate's name correctly. I'd forgotten that lesson by time I took my first job after college. There was a Chinese man on the team whose name seemed simple enough to pronounce, but I wasn't pronouncing it right at all. I am pretty sure I said his name incorrectly in front of others including him probably several times before my boss corrected me. I don't think my Chinese colleague noticed, and he wasn't the type to hold grudges, which is good because he became my boss several years later.

I use small talk as a way to discover common interests. Common interests are the basis for most friendships, and thus a good target for in-group bias. I play computer and tabletop games, both fairly common hobbies within a technology crowd. You probably won't have common interests with everyone on the dev team, but most people have at least one really weird hobby that you can quiz them about. "Really? How did you get into that? How does that work?" I knew this one dev who collected Tom and Jerry cartoons. Apparently, it's not the sort of thing you can just go to the store and buy. Some episodes are just no longer available when they touch on sensitive subjects like the World Trade Center.

People like to talk about themselves and their hobbies, so it is a low ante credibility bet. The more you know, the more likely you are to be able to connect it back to part of your experience or start a new conversation later about something that happens afterwards. That's how you win. Several of the developers in my current job are interested in gardening. I am not on their level, but I show off pictures of the shrubs around my house and complain about the invasive plant species.

Motivation

The first things I look at when I am trying to influence someone or to diagnose the issues in an organization are the motivations of the people. In a work context, the most obvious motivator is money, which is usually tied to some review process. A lot of times this is unhealthy but you can only rail against it or ignore it for so long and still be employed.
Sometimes metrics are used as part of the reviews. Bug counts, thousands of Lines of Code Written (kLOCs), unit test coverage, number of stories or features delivered. When you know what those metrics are, you can understand how your team members are gaming it, and game the numbers a little bit yourself. I have worked on teams where developers are rewarded for delivering features but are not allowed to work on them once their bug count reaches a certain level. On other teams, the developers are hassled if they have bugs assigned to their features. They find it incredibly valuable if I sit on a bug for a day or two or just let them fix a problem without entering it into the tracking system. This is a powerful technique for two reasons. One principle called the "rule of reciprocation" (Cialdini 2009), which is the social pressure to return a favor. Since I have done a favor for them, it is much more likely they will do a favor for me in the near future. "The rule possess awesome strength, often producing a yes response to a request that, except for an existing feeling of indebted-ness would have surely been refused." (Cialdini 2009) The other is that it makes them feel like you are on the same team, which allows you to take advantage to the in-group bias. "In-group bias is a central aspect of human behavior. Across a variety of scenarios, people tend to be more helpful to members of their own group rather than to those of other groups." (Fu et. al 2012)

While it is your job to critically evaluate the developers work, there are probably other people who are also evaluating their work such as managers, peers, clients, etc. The developer cares more about looking good in front of these people than in front of you. If you couch your feedback in terms of avoiding looking bad, but speaking credibly for the others; they will value your input.

Developers also tend to be proud of their work. Each developer is proud of a different thing. For one it will be performance or security; for another, elegance. You can figure out what by seeing

what they choose to fix or to work on in their own time. I've known a developer to spend hours tweaking a line's gradient. He put up with a terribly performing virtual machine because that technology supported hardware video acceleration and therefore more accurate colors. This wasn't a fancy video game. It was a graph, and a solid line would have worked. In fact, we changed it to a solid line before shipping because it made a clearer graph. It impressed upon me, however, how much he cared about color changes. From then on, I took color related bugs to him first because they automatically had more credibility there. Another developer was really proud of the MSAA accessibility tree he built into his features. Knowing this, I can direct feedback to the developer who cares about it the most. An informal, "Hey, did you see what someone else did?" and that team member will lean on the other developer for you as well as appreciating that you care about the things that they care about.

Other developers are looking for a puzzle to solve. If you can get a divide by 0 error in a program that doesn't do math or a race condition in a single threaded application, someone will want to know more. A reputation for making the impossible happen is a very useful thing for a tester to have and a sign of high credibility. One of my developers said he was going to get me a cape because my bug-finding abilities were super human. At that point, I could change the design of a feature with just an evil grin. I don't have to know why the design won't work. If I detect some latent uncertainty, just the smile of someone who breaks everything they come across is enough to make the team members think more about the problem, sketch it out on the whiteboard, and find the bugs for me.

Both the pride and the puzzle solving are expressions of what Daniel Pink calls mastery "the desire to get better and better at something that matters. "(Pink 2009) Once you understand their motivations, you can couch your feedback as helping them achieve their goals. It will risk less credibility to wager and have a higher return.

Common Enemy

A team is built around a common goal. The members have to accept that they share that goal, which no amount of organizational structuring can do. I've never come back from a team-building exercise and felt that I was really any closer to someone than when it started. However, at the end of a tough project or release, I feel closer to the people who made it happen.

Software projects are hard. Something is almost always a barrier to progress, and that something typically translates into a someone. It often results in an us versus them mentality. You really want to be on the us side, but how do you show that you are on the same side when your purpose is critiquing? Find the other things on the project that are making it difficult.

People who set deadlines, people who add requirements, changing or broken dependencies are probably not actively working against you. They have a range of responsibilities and priorities that you know nothing about, but it sure does feel like it. The developers feel the same way. If such unreasonable requests are squeezing you, they're probably squeezing the development team, too. They already know who is causing them grief, so you need to let them know that the enemy is common.

You need to show them that you hate the things they hate to get them thinking, "This person thinks like I do." Make them believe you share their values. Once they do, you will be a member of their team allowing you to take advantage of in-group bias.

My favorite way to do this is with snarky comments. I am going to make them anyway because that's who I am, but carefully applying them can put you on the developer's side. A snarky, "This will be a fun project" made to the dev after a tough meeting lets them see that you share their concerns and can relieve some of the tension. The pitfall is that you may snark about someone who is more popular than you are. Stick to attacking ideas, at least until you know the developers well enough to say who they are close to.

Another way is to anticipate the developer's objections and raise them first. Problems with development will definitely affect you. Voicing their concerns lets them know that and shows you as someone who is on their side. The developers may remain silent because they don't feel they have the credibility to speak up. Turning to them and asking, "If we do it this way, will you even have time to finish programming?" says to them I will back you up and you can borrow some of my credibility to bet against this objection. Even if they don't win the battle, they will appreciate your efforts on their behalf. It does not have to change the course of the project to improve the relationship. Even if you find out about things later in a water-cooler conversation, you can raise the objections before the developers get a chance to say them. Finally, you can always ask how you can help. There is usually a little bit of gaming the system you can do to bail them out, and it make you look like a team player in either case.

Overcoming Obstacles

Not everyone will be easy to gain credibility with. Some will be hostile or decidedly indifferent. You may hear things like, "Works on my machine." Chances are you are doing something subtly different. Every time you round trip with a guess as to what that is, you lose a little credibility in addition to time. My first move in that situation, is to go to their office and offer to show the issue on their own machine. Failing that, I invite them to see it on my machine and allow them to debug it while I watch.

If you are having trouble getting past an us versus them mentality, you could try making your good intentions explicit with a big visible promise. That is, taking something that says directly, "I am on your side" and making it visible. Personally, I like the Testers Commitments that James

Bach wrote, but it doesn't have to be that detailed. Print it out, sign it, and hang it in your workspace. Then, live by it. It's hard to be hostile to someone who is openly trying to help you. Once, I was on a project with a developer who was a bear to work with; even the other developers thought so. I wasn't really responsible for testing his area, so I let it go. He got burned a few times releasing bugs to the wild and eventually asked for my help. Basically, I built up so much credibility that he couldn't ignore me anymore, but that took over a year. Ultimately, you are here to help the developers and they will be more successful working with you than against you. On a long enough timeline, they will have to cave from the pressure of the other developers. The final tip from social psychology is "…the principle of social proof. This principle states that we determine what is correct by finding out what other people think is correct." (Cialdini 2009)

Conclusion

If there is only one thing you take away from my paper, it should be seeing every interaction with developers through the lens of credibility. It becomes much easier to have an open dialog if the developers trust you. I have even had developers tell me about bugs in their code, as they were handing it off, because they trust that I am trying to make them look good. They also believe that I will find the bugs anyway, which may or may not be true. You can't win over everyone, but if you make a conscious about your credibility and aware of the science of influence you and have a much stronger relationship.

Bibliography

Carnegie, Dale. *How to Win Friends and Influence People*. New York: Simon and Schuster, 1981. Print.

Cialdini, Robert B. *Influence: Science and Practice*. 5th ed. Boston, MA: Pearson Education, 2009. Print.

Fu, Feng, Corina E. Tarnita, Nicolas A. Christakis, Long Wang, David G. Rand, and Martin A. Norwak. "Evolution of In-group Favoritism." *Scientific Reports* 2 (2012). Web. <http://www.nature.com/articles/srep00460>.

Kahneman, Daniel. *Thinking, Fast and Slow*. New York: Farrark Straus And Giroux, 2011. Print.

Machiavelli, Niccolò. *The Prince*. Trans. David Wootton. Indianapolis: Hackett Pub., 1995. Print.

Pink, Daniel H. *Drive: The Surprising Truth about What Motivates Us*. New York, NY: Riverhead, 2009. Print.

Sun Tzu. *The Art of Strategy: A New Translation of Sun Tzu's Classic, the Art of War*. Trans. R. L. Wing. New York: Doubleday, 1988. Print.

About the Author

Curtis graduated from Florida Institute of Technology (FIT) with a Bachelor's of Science degree in Software Engineering. During his studies at FIT, he focused on software testing while working at the Center for Software Testing Education and Research. He spent eight years as an SDET at Microsoft, where he lead a transition from scripted manual test cases to session-based test management. Currently he works as a Senior Software Test Engineer for HUGE, a digital advertising agency, testing websites and mobile apps for a variety of brands.