This Would Be Easy If it Weren't for the People

Danny R. Faught

danny.faught@gmail.com

Abstract

Soft skills are important for software professionals to learn. This paper explores soft skills using a series of vignettes drawn from the author's experiences -

1) "Collaborative design"
2) "But you didn't fix my bug"
3) "You're getting in my way"
4) "You have a big problem, wait, no you don't"

Biography

Danny R. Faught is a 21-year veteran of the software industry, focusing on software testing and customer service. He is a graduate of Jerry Weinberg's PSL and Change Shop workshops.

Danny is currently a Senior Software Developer in Test at AgileAssets, Inc. He has a B.S. in Computer Science from the University of North Texas.

1 Introduction

Software professionals encounter frustrations on the job just about every day. Many of these frustrations don't come from the technology that's at the core of their work, but from their interactions with the people they work with. This paper will present a series of vignettes based on real experiences to illustrate a variety of soft skills that have made me more effective on the job.

The vignettes are:

1) "Collaborative design" – accepting design input from a co-worker, and using Myers-Briggs to figure out how.
2) "But you didn't fix my bug" – understanding how our ideas about what's important often differ from those who make the decisions, illustrated by a security bug that didn't get fixed.
3) "You're getting in my way" – avoiding inflicting unwanted help on a co-worker who was in the critical path.
4) "You have a big problem, wait, no you don't" – when following the compulsion to tell a programmer about a major design flaw, "reply all" is not the way to do it.

Jerry Weinberg taught me much about using soft skills on the job. I especially remember the concept of "center, enter, turn" approach that he first learned from Aikido. George Dinwiddie offers a nice description (Dinwiddie, 2007) -

> "Center: Be aware of yourself, who you are, and what you want to accomplish.
>
> Enter: Be aware of the other. Enter their world and stand beside them, rather than in opposition.
>
> Turn: Together with them, turn their energy in a more effective direction."

I often find, however, that I'm redirecting my own energy as often as trying to change someone else.

The stories in this paper are drawn from various companies I've worked with. Some details have been altered in order to avoid sharing proprietary information, and all of the names of the people have been changed.

This talk builds on ideas first published in "Testing, Zen, and Positive Forces" (Faught, 2004).

2 Collaborative design

In this story, I had to deal with a challenge to one of my designs.

I was trying to convince the QA team to change their existing functional test automation framework so that it used a keyword-driven approach. These are the things I wanted to accomplish:

- Have automated tests that were maintainable. I felt that this was critical, after seeing so many automation efforts fail miserably because of maintainability issues.
- Use an incremental approach, so that we wouldn't break what was already working, and so that we would get frequent feedback on whether the design was feasible (in short, be agile).
- Develop a good working relationship with my new co-workers and earn their respect.

I was brainstorming the design with John, another test developer on my team, and we were stuck. We were writing our first test in the not-yet-developed keyword framework as a way to illustrate the design. The simplest test we could think of just tested the login feature. After some debate, I had written something like this on the whiteboard for the test case:

```
browser start
app start
app login user-type=admin
```

I was thinking about the big picture, wanting to encapsulate as much as possible into each keyword, which helped with the goal of better maintainability. The validation would be built in to the keyword implementation as much as possible, without having to explicitly put it in the test case. I liked the fact that it was only three lines long. But John didn't like my test, and said that he didn't even think that using the login feature for the test was a good idea.

I valued John's opinion, so I wanted to understand the basis of his objections. I felt that the design I had sketched out was a solid plan for our implementation, but it wouldn't be wise for me to dismiss his concerns if I didn't really understand what they were.

Earlier in the day, I had discussed the test framework design with John and our co-worker, Bill. I knew that Bill's was an "extravert" according to the Myers-Briggs Type Indicator®, and so was I (see Weinberg, 1994, for more on Myers-Briggs). That means, among other things, that we tend to say things out loud as a part of processing our thoughts. At one point, in response to something Bill directed at him, John asked for a moment to think and process it, which is typical of a Myers-Briggs introvert (introverts prefer to process a thought fully before saying it out loud). Out of respect for that, I asked Bill that we put a hold on our discussion to give John his moment of silence, and I resisted our habit of continuing to think out loud any time there's a pause. This interaction put the Myers-Briggs model at the forefront of our thoughts, and that's what helped get us past our test design impasse later in the day.

I realized during our test design discussion that I was thinking top-down, looking at the big picture, without fleshing out any of the details. That's typical of a Myers-Briggs "intuition" thinker. It occurred to me that John might be on the opposite end of the Myers-Briggs scale in that regard ("sensing"), which would mean that he would want to start with the details of something that is clearly implementable before trying to see the big picture. That led us to produce something more detailed on the whiteboard:

```
browser start
browser go $start-page
ui type username user-type=admin
ui type password user-type=admin
ui submit submit_button
```

Now, I feel strongly that most of our tests should not go into this level of detail. I want to encapsulate the interaction with the user interface into higher-level keywords like the "app login" line of my first draft test case. John, however, wasn't able to evaluate this more detailed view of my idea, and frankly, neither could I, until we fleshed the low-level details. In fact, we decided put off implementing a feature that allows for aggregating keywords into the "subroutine" calls that my first attempt at a test case relied on, and this meant we were taking a more incremental approach. We agreed that we would not develop a large number of tests at this low level, but it would be beneficial to write the first several tests this way.

I was glad that I was able to swallow my ego and accept feedback. My goal of earning the respect of my co-workers had led to a better way to achieve my more important goal of following an incremental approach.

I didn't ask right away what John's Myers-Briggs type was. Like many models, it wasn't important if I was right about what his preferred way of thinking was. If the Myers-Briggs model helped me to find a better way to interact with someone, that's really all that matters, and I didn't need to explicitly apply any labels. I've found this to be true with many psychological concepts.

3 But you didn't fix my bug

What are we here for, if no one bothers to fix the bugs we're finding? This story explores why we should understand our stakeholder need, and why we shouldn't assume our bug reports aren't useful.

Once, I was doing a quick quality survey of a web-based software application developed by a startup. I found that I could get to the screen to edit the account profile for any account on the system, without having administrative access to the system. I sent the CEO a screen shot of the profile edit screen for his own account. I was surprised to find out that the organization did not plan to fix the security hole in the foreseeable future. It's not that my bug report was ignored – it just wasn't the kind of bug they wanted to know about. This was an early phase startup, focused on building enough of a prototype to attract investors, but they were not trying to build production quality code.

I have been reminded many times since then that I have to tailor my deliverables and my expectations to the current context (see Kaner et al., 2002 for more on context-driven testing, e.g., p. 2, "Your mission drives everything you do"). When I begin testing, I gather information about what kinds of bugs my stakeholders most want to know about. It may be difficult to get a good answer this this in a vacuum, so I will also look in the bug database to see what kinds of bugs have already been fixed, and which ones are languishing in the queue. As I proceed through the project, when I get concrete examples of bugs, I will occasionally get feedback on whether the bug is important to report.

Though I generally prioritize my bug finding activities so that I report bugs that are likely to get fixed, I don't stop there. Usually stakeholders want me to report most of the bugs that I find, even if they're not going to be fixed for a long time, and perhaps never fixed at all. The information provided in the bug reports can be used in a variety of ways, including looking for clusters of minor annoyances in the software, planning bug fixing resources for future releases, including information about known issues in release notes, and handling technical support calls.

There is certainly a place for bug advocacy – sometimes you should be the champion for a particular bug and advocate for it to be fixed. Choose wisely when you do this. See Cem Kaner's "Bug Advocacy" presentation for more on this topic (Kaner, 2000).

Don't be discouraged if the bugs you reported aren't getting fixed. There are several uses for the information you provide in a bug report. Find out all of the things your organizations uses bug reports for, and cater to those needs.

4 You're getting in my way

This is a story about trying to inflict unwanted help.

I had a deliverable that had dependencies on the work of two other teams. I got a report from a technical contributor on one of these teams, Jack, that his part was going to take longer than anticipated because of the complexity of the task. I was fairly sure, based on Jack's brief description of the problem, that his solution was overly complicated. I thought that I could make his job easier and get the solution more quickly by giving him a better understanding of what I needed.

I tried to meet with Jack to discuss this with him. However, Jack asked me to include a manager who was involved with the project in the meeting. That would mean I should also invite my manager. I was hoping to have a technical discussion just with Jack, and then give a recommendation to the affected management afterward. I was nervous about turning the meeting into a big affair, having seen some dysfunctional meetings in the organization before. I prefer to solve problems in small meetings. So I consulted with my manager, who requested that I not stir the pot. He was willing to accept a delay in my deliverable, so we agreed that the help I wanted to offer was unlikely to be helpful. I'll never know exactly what kind of brouhaha I avoided, but I'm glad I avoided it nonetheless.

For further reading on inflicting help, see Weinberg, 2011.

5 You have a big problem, wait, no you don't

I was sure I had found a big bug, but should I shout it from the mountaintop?

A software developer, David, send an email out to a mail alias to all of the developers and QA staff asking for feedback on a database query he wanted to change. Though I wasn't qualified to answer the question he was asking, I looked at the code and thought I saw a glaring security hole, making the code vulnerable to a SQL injection attack.

I replied to the email and expressed my concern about the security hole and suggested how to fix it. David replied and explained that this was static SQL code, so there was no way for someone to modify it externally, and thus there was no vulnerability. Upon further reflection, I realized he was right, and my suggested fix couldn't even be done in pure SQL anyway.

It was at this point that I heaved a sigh of relief that I hadn't done a group reply. My mistake could have hurt my credibility with the entire development organization and on my own team, and perhaps made it less likely for anyone to pay attention to other issues I tried to raise later.

While it was embarrassing to admit to David that I was wrong, I'm glad I did speak up, because it was a learning experience for me. I think that the better understanding of our product's implementation that I gained was worth suffering a small gaffe in an interaction with just one other person.

Of course, this also reinforced my policy of not doing a "group reply" for an email without carefully considering the ramifications.

6 Conclusions

A few common themes emerge from these vignettes –

- Build relationships with your co-workers. Use each interaction as a stepping stone to build on. When a difficult situation arises, you can use those relationships to find your way to a solution more easily.
- Do no harm. Carefully choose your audience when sending potentially bad news or corrections. Sometimes it shouldn't be sent at all. Be bold when there is good reason to share the news.

I've found that the biggest challenges in my career have not been the technical challenges, but working with the people. Whatever technical work you're doing, you're almost certainly going to need to interact with other people along the way. It's not a "them vs. me" sort of challenge. It's not often a simple case of incompetence. It's just other people like me, who are trying to do their job just like I'm trying to do mine. The more you can refine your soft skills, the more effective you'll be on the job.

References

Dinwiddie, George, July 22, 2007. “Blocking.” http://blog.gdinwiddie.com/2007/07/22/blocking/ (accessed June 16, 2014).

Faught, Danny. March 11, 2004. "Testing, Zen, and Positive Forces". http://www.stickyminds.com/article/testing-zen-and-positive-forces (accessed June 16, 2014).

Feathers, Michael. 2004. *Working Effectively with Legacy Code*. Prentice Hall PTR.

Kaner, Cem. 2000. “Bug Advocacy”. http://www.kaner.com/pdfs/bugadvoc.pdf (accessed June 16, 2014).

Kaner, Cem, James Bach, and Brett Pettichord. 2002. *Lessons Learned in Software Testing: A Context-Driven Approach*. New York: John Wiley & Sons, Inc.

Weinberg, Gerald M. 1994. *Quality Software Management*, Vol. 3: Congruent Action. New York: Dorset House Publishing.

Weinberg, Gerald M. March 29, 2011. “Knowing What to Leave Alone.” http://secretsofconsulting.blogspot.com/2011/03/knowing-what-to-leave-alone.html (accessed June 16, 2014).