

Establishing an Agile Testing Team: Our Four Favorite “Mistakes”

Kay Johansen, Anthony Perkins

Lehi, Utah 84043 USA
kay@xmission.com
anthonyperkins@email.com

Abstract. The authors have spent the past year building a software testing team in the high-speed, high-change environment of an Internet development company. As we'd hoped, the Agile values helped us tackle the difficulties of testing in such an environment. We also discovered that the principles of the context-driven school of testing aligned well with the Agile values, and that both schools of thought helped us with the politics of working with other teams and obtaining support in the organization.

1 Introduction

Over the past year, the authors have learned the truth of Tom DeMarco's statement that "the major problems of our work are not so much *technological* as *sociological* in nature." [1] As former developers, we were drawn to the technological challenge of sufficiently automating QA testing to support the kind of rapid release cycle common in Internet development. As members of the Agile community, we were interested in the sociological challenge of introducing new processes to an organization. This paper reports on our first year of experience as testers.

In January 2000, Kay accepted an offer from a large Internet company to build and manage a QA team at one of its sites, a successful startup acquired two years previously. As a startup and then as part of the parent company, the site had a four-year history of delivering solid technology and enjoying market success with its ad-hoc, fast-paced, usefully sloppy process—a process that didn't happen to include QA.

Kay hired Anthony to help her create and shape the test team. The two of us had already worked together as developers on an Extreme Programming style project [2], and shared an appreciation for the XP practices and underlying values. Together we thought we were up to the challenge of adding testing to the company's developer-driven culture without slowing down their release cycles—and without being rejected by developers or management.

The risk of being rejected by the organization led us to make some intentional “mistakes”—actions that may seem counter-intuitive from a traditional testing perspective, but which (in hindsight) do seem to be supported by the context-driven school of testing

(ref). We didn't really figure out which "mistakes" to make until after we were well underway with our new team, so before we describe the "mistakes," we'll offer some context and explain how we set up an Agile testing team.

2 Situation

The "products" we needed to test were really collections of open-source software customized by our developers, delivered on our company's servers and leased to customers by the month. The operating system and the services offering the basic Internet protocols (http, ftp, pop, imap, smtp, telnet, ssh) were the most customized; typical applications such as PHP, MySQL, Perl, Python, and Java, to name a few, were usually just configured for our system and given a nice install. But it wasn't all open-source: as part of their newest product, the company was developing a layered web application completely in-house. We needed to span several different styles of testing with our new team.

We also needed to support two completely different release tempos. The developers updated existing products weekly, or more often; easy to do because everything was deployed on our company's own servers. At the same time, they were developing a new product on a schedule of about a year of design and development before launch. We felt a surgical, white-box style of testing, heavily supported by automated tests would work for our weekly release cycle, while the new product testing seemed amenable to more familiar (to us) testing cycles of top-downs, alphas, betas, etc.

Our team, fully staffed, consisted of eight testers including the authors. This we divided into three subteams to test three products, testing for a total of approximately twenty developers. This arrangement lasted for perhaps six months, until layoffs hit and we regrouped into a single test team. As the structure and priorities of the parent company shifted and changed, so did the structure and priorities of our test team; the best way to describe our experience is to start at the beginning, when we were setting up a new team.

2 What We Did

We spent a lot of time initially thinking about how to add testing while maintaining or enhancing the company's productivity. We took time to assess the organization's current philosophy and methods before forming our own goals. We noticed that the product development group was essentially agile: self-organizing teams of motivated individuals were releasing software to customers frequently. They didn't have a documented process they followed, or much internal documentation at all, but they were dedicated, skilled and worked together closely. However, they were starting to feel the effects of being a larger,

more ambitious organization, in the form of schedule overruns on their new development project.

Recognizing their apparently chaotic process as essentially an agile approach to real-world constraints helped us focus our testing goals. Instead of trying to "measure and control" a process which was fundamentally not controllable, we found it more useful to learn from the Agile software development community and adopt the Agile values.¹

2.1 Individuals and Interactions over Processes and Tools

We were hired to establish better testing processes. The first thing we were asked to do was to document what those processes would be. At that time, the company was working with an outside consulting firm to design a total product development process to be used henceforth on all projects. They wanted our input on the testing sections.

Recalling James Bach's warning, "If you don't know how to work together, whatever your defined processes, I can all but guarantee that your processes are not being followed" [3], we made team building, not process, our first priority. We stalled on every request to formally document our test methodology, feeling that such an action on our part would limit our team's creativity. We hired known team players, created an open workspace, set up a WikiWikiWeb² to allow everyone on the team to contribute to the team's test plans and processes, encouraged good "convection currents of information" and in general gave the team every opportunity to collaborate and invent, following the philosophy of Alistair Cockburn [4].

Without a process to tell them specifically what to do, the testers became used to noticing and solving problems. Two testers developed a simple and powerful test framework in Perl to automate both command-line and GUI tests; others created user interface standards guidelines; one tester introduced our team's method of task allocation using three-by-five cards. Our emphasis on interaction and teamwork also gained the respect of other teams: In a general survey about the testing team that we sent to employees at our site, all responses praised our team's high level of teamwork. On the other hand, the company-wide process designed by the consultants is a decorative but unused binder on each manager's shelf. (move to end?)

2.2 Working Software over Comprehensive Documentation

While it was difficult for us as a testing team to accept the lack of specifications or other documentation that prevailed at this Internet software company, the Agile values

¹ The Agile values are part of the Agile Manifesto. For more information, see <http://www.agilealliance.com>.

² The WikiWikiWeb is a collaborative online discussion format invented by Ward Cunningham. The original WikiWikiWeb is at <http://c2.com/cgi/wiki>.

reminded us that documentation is a poor way of controlling software delivery. So, instead of spending energy requesting specifications, requirements documents, design documents or even schedules, we worked on obtaining the code itself.

Our experience at prior companies had taught us the importance of Jim McCarthy's statement that "the regular build is the single most reliable indicator that a team is functional and a product is being developed." [5] That's why we first worked with the configuration management developer, learning about server administration and technical details of the company's products, until with his help we could set up a clean test environment that was capable of accepting new code daily. Once our test environment was in place, we took every opportunity to influence other teams to get code to testing early: asking developers in *what order* they would deliver features to us, diagramming for the project manager the effect on schedule of delaying testing, and pointing out how one component in particular, always developed first then handed off to testing, somehow always missed its schedule.

Because our team was finding bugs in real code instead of raising concerns about decisions in design documents, developers and project managers paid attention to our findings. The developers, who didn't have an integrated build of their own, found our environment very useful for troubleshooting. Our team could easily and always tell the state of the code, which helped the project and product managers make tough scope versus schedule decisions.

2.3 Customer Collaboration over Contract Negotiation

Before we were hired to create a testing team, product management and development were caught up in a blame game over the new product because it wasn't realized until very late that there were serious problems preventing delivery. We hoped that by providing an open and timely flow of information, the testing team could begin to patch up this relationship and get people talking again about the best interest of the users instead of the details of the requirements document.

We worked to gain the trust of both teams by constantly sitting down with developers or product/project managers, asking lots of questions, and listening to their concerns. We repeatedly brought up the user's point of view in meetings, and our testers used the user's point of view when writing bug reports. We hosted regular bug prioritization meetings to get the status of the product out in the open and to encourage a productive dialogue between product management and development.

Because we listened to their concerns, developers and product management were willing to work with us. In the bug prioritization meetings, testing refused to assign priorities to bugs, so product management and development had to come to agreement on bug fixes, discussing them in terms of importance and difficulty. Because of the regular bug meetings, everyone remained apprised of the problems and risks, and worked together to find solutions.

2.4 Responding to Change over Following a Plan

If you measure our results today against our original goals, you'd probably say that our implementation of testing was a failure. But as Jim Highsmith says, "In a complex environment, following a plan produces the product you intended, just not the product you need." [6]

Our original goals for the testing team didn't account for the environment of high change we discovered in a company regrouping from a massive acquisition spree. Our manager, and his manager, as well as *his* manager the CTO, were all laid off. Development managers came and went. Products were canceled; others changed direction several times. A usability/human factors team was added. Testing was moved out of development and into customer support. A third of our testers were lost in layoffs.

We quickly learned that we had to do more than just test on short release cycles and write automated tests. Our original goals could be flexible, but if we were to survive and succeed in this environment, there were some things we decided we had to accomplish:

- Add value
- Be perceived by management as adding value
- Gain trust and support from the grass roots of the organization
- Get others to take responsibility for quality

The next section describes our revised strategy for accomplishing these goals. (change?)

3 What We Didn't Do

Introducing change is difficult. It seems that organizations have an almost immunological response to personnel who act in a new or foreign way. Fortunately for us, we weren't in a position of power, which immediately ruled out a heavy-handed, "ram it down their throats" approach, or even a self-righteous, evangelistic one. Indeed, we were a new team, without strong management support, understaffed, trying to do something that was unfamiliar to the company, not to mention unfamiliar to us. Paradoxically, our very weakness allowed us to make several "mistakes" that were essential to our success. In hindsight, what we didn't do turned out to be just as important as what we did (improve?)

3.1 We Didn't Protect the Customer

We wanted to protect the customer. Others looked to us, the new QA team, to protect the customer. The director of product management made it clear he wanted us to be the ones to fail releases if we thought their quality was unacceptable. But we feared our position in the company was too weak for us to be effective defending the customer by ourselves, and

that if we agreed to take on the responsibility of protecting the customer, others wouldn't have to.

So instead of taking sole responsibility for quality, one of the first things we did was to make friends with our product manager. We had many conversations with him, showing him that we were shifting our goal of bug-free products more toward profitability for the company, and asking him to share our goal of quality instead of delegating that entirely to our team. Similarly, to the development team, we chose to be a “credible, high-integrity reporter of information” [7] instead of the Defenders of Quality. We rarely spoke out in defense of a bug, but when we did we took care to phrase it as providing more information rather than as making a recommendation.

By never joining battle, over time we became unassailable. Other teams accepted us as information providers, and excused us from argument when they were frustrated about products releasing too quickly or too slowly. Making friends with the product manager early, when the pressure was low, paid off later as the pressure increased. He continued to seek out and listen to our input even when we refused to make things a lot easier for him by taking the responsibility to fail a release.

3.2 We Didn't Hire Testing Experience

We wanted to hire experienced testers. Good testers think differently from developers [8], and we wanted to introduce that testing thought process to the company. However, this site had never had a testing team, and the developers were used to pushing changes out quickly with no review. Since the startup company had been very successful, we were concerned that developers and others wouldn't immediately see the value of an experienced tester's approach.

Developers ourselves, we put ourselves in the developers' shoes, and imagined a team of people suddenly appearing and criticizing everything we did, without understanding our work or caring about what we thought was important. Would we trust that their decisions about our product would be good? Would we go out of our way to help them? So in the beginning, we looked for people to hire who would have been attracted to the original startup, even though the office now belonged to a company of two thousand people instead of twenty. It was more important for our testers to be capable of discussing the latest build of Apache or the security advantages of FreeBSD over Linux with developers, than to have the most or the best experience in testing.

We found that the developers accepted people with a genuine interest in their product, and would go to great lengths to help our testers in any way they could. (more here?)

3.3 We Didn't Test Everything

We wanted to test everything. Ideally, we would have tested everything. But in the real world, we were limited in our number of testers, there were a lot of projects, schedules

were aggressive, we were unclear about our management support and we were fighting for every resource we had.

We thought the best way we could help the products was to quickly demonstrate our value to the company so that we could live to test another day. Testing was new to the company, and we had read about the importance of “early wins” when introducing something new [9]. We thought that if we tried to test every piece of every product, we couldn’t deliver a clear “win,” we could only deliver a mediocre (or worse) performance on all the things we were testing. Instead, we focused our effort more narrowly, in order to demonstrate the value of our work more clearly. We regularly communicated our intentions to management, forcing them to either agree with our prioritization or to make hard tradeoff decisions of their own. At one point management approved our recommendation to pull every tester off other products to commit completely to a high-profile product.

This wasn’t a pleasant or popular decision, but we had communicated so clearly that when bugs escaped on the untested products, the reaction was “How many more resources do you need?” instead of “How could you have let that happen?” It took a long time, and things got worse before they got better, but in the end management gave us hiring ability to staff our team up to the number of testers we had before the layoffs.

3.4 We Gave In Easily

We wanted to improve the process of product development. Our actual title in the company was Quality Assurance. Shouldn’t Quality Assurance be the voice of reason against poorly documented requirements, changing requirements, scope creep, unrealistic schedules, insufficiently reviewed code, poor version control, and so on?

But reason prevailed. We didn’t want to add more problems to the system. We believed that if everyone on the project team simply remained civil to each other, that would go the longest way toward getting a product out that was of value. With this mindset, we could pleasantly surprise everyone by cheerfully accepting things that are often resisted by QA teams. The requirements aren’t documented? No problem, we can deal with it. You just added seventeen new features? Great, that’s better for the customer! You want this code to go out without testing? We wish we could be there for you, but we don’t want to hold you up. If a fight wasn’t of critical importance to the customer or the organization, we stayed out of it. If we made a request and met resistance, unless it was for something absolutely necessary, we dropped it.

Because we were perceived as very conciliatory and accommodating, people gave us more support when we actually had an issue we wouldn’t give in on, such as requiring a clean test environment. Because of the zeal shown by our testers who wanted to test everything, other teams began also to be disappointed when a version of the product was released without sufficient testing, and a commitment to allow for better testing on the next release was easy to acquire. (one more result would be nice here, and a transition to the Reflections section)

4 Our Reflections

We've described how the Agile values helped us build an effective testing team:

- Valuing individuals and interactions helped us build a self-organizing team.
- Valuing working software caused us to develop a system for obtaining daily builds.
- Valuing customer collaboration brought us closer to the product manager and developers.
- Valuing responsiveness to change made us pay attention to what we were learning and adjust our priorities.

No process we could have invented could have been complex enough to meet the many changing and interacting testing needs resulting from the complexity of our products, the different release tempos, the huge size of our customer base and the shifting priorities of our company during a time of great economic difficulty for Internet software businesses. But the Agile values guided us to build a team of human beings sufficiently complex to comprehend and adjust to the many challenges we faced over the year, a team that created and discarded processes as needed to perform good testing and to consistently provide information that was valued and used by other teams.

We notice among our colleagues in the Agile community a lot of interest in “how to introduce Agile to an organization.” The opportunity to pursue Agile methods from the testing department expanded on our prior experience in development. We discovered more of an emphasis on sociological factors in the testing literature as compared to development, and would like to point developers as well as testers struggling with introducing Agile methods in this direction. (rewrite this paragraph – the purpose is to introduce the testing community in general and context driven testing in particular as a valuable source of experience for those interested in introducing change)

Yes, our first strategy in approaching the problem was wrong. But because we weren't hung up on sticking to our original plan, we were able to adjust our strategy as we went along. We shifted our priorities from the technical (test automation, complete coverage) to the social (securing support for our team, getting others involved in testing). As we did this, we used some practices that seemed counter-intuitive to us at first, because it seemed like we were not doing things that (as non-testers) we thought QA might usually be expected to do. We still believe our actions were consistent with the Agile values. But when we found out about the context-driven testing school of thought, we saw our “mistakes” actually being used as lessons, as summarized in Table 1.

Table 1. Correlation between our experience and context-driven testing

| Our “Mistake” | Context-driven testing “Lesson” |
|--------------------------------|---|
| We didn’t protect the customer | 12 – Never be the gatekeeper 205 – Don’t sign-off to approve the release of a product |
| We didn’t hire QA experience | 150 – Understand how programmers think 151 – Develop programmers’ trust |
| We didn’t test everything | 10 – Beware of testing “completely” 227 – Don’t let yourself be abused |
| We gave in easily | 14 – Beware of becoming a process improvement group 176 – Adapt your processes to the practices that are actually in use |

Much to our delight, we discovered lesson 285 taught in the book *Lessons Learned in Software Testing* is—you guessed it—“Your first strategy on a project is always wrong.”

5 Acknowledgements

The authors wish to thank Bill McLoughlin for his strong belief that people are more important than process, Alistair Cockburn for his “words of encouragement” on our move to the testing department, and Linda Rising for her help and advice as we submitted this paper to XP/Agile Universe 2003.

6 References

1. DeMarco, T., Lister, T.: *Peopleware: Productive Projects and Teams*. 2nd edn. Dorset House, New York (1999)
2. Johansen, K., Stauffer, R., Turner, D.: *Learning By Doing: Why XP Doesn’t Sell*. In: XP Universe 2001 Conference Proceedings. Addison Wesley Longman, Reading, Massachusetts (2001)
3. Bach, J.: *What Software Reality is Really About*. In: IEEE Computer. IEEE Computer Society, Los Alamitos, California (December, 1999) 148-149
4. Cockburn, A.: *Agile Software Development*. Pearson Education, Boston (2002)
5. McCarthy, J.: *Dynamics of Software Development*. Microsoft Press, Redmond, Washington (1995)
6. Highsmith, J.: *Adaptive Software Development. A Collaborative Approach to Managing Complex Systems*. Dorset House, New York (2000)
7. Kaner, C., Bach, J., Pettichord, B.: *Lessons Learned in Software Testing. A Context Driven Approach*. John Wiley and Sons, Inc., New York (2002)
8. Pettichord, B.: *Testers and Developers Think Differently*. In: STQE Magazine. Software Quality Engineering, Orange Park, Florida (January 2000)

9. Kotter, J.: *Leading Change*. Harvard Business School Press, Boston (1996)