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 test 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 Agile values 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]

This paper describes our experience building a software testing team using the Agile values. We'd used Extreme Programming on a previous development team [2], and were curious to find out if the Agile values could be extended to testing. When an opportunity came to join a large Internet development company and develop their testing team at their Utah site, we took it. We were drawn to the challenge of adding testing to the development process without bottlenecking their productivity or being rejected by their rather developer-driven culture, all the while maintaining management support.

We thought we were up to the challenge. We were armed with the Agile values which we hypothesized would work in testing as well as they had in development, and with our development background which would help us create the kind of testing team that could fit into an overall Agile development process—specifically, we thought we'd support rapid incremental development by significant automation of acceptance tests. We were wrong about the latter, but right about the Agile values. Not only did they help guide our decisions in the one area we had control over—the actions and attitudes of the testing team—they also affected the way we tried to influence factors out of our direct control—the actions and attitudes of others.

We cared about the actions and attitudes of others because we knew we weren't going to be in a position of power. We would be part of an interrelated community of teams, each team supporting and depending on others to accomplish objectives. Even if we succeeded in creating an effective testing team, we couldn't assume that everyone would

recognize the value of our testing effort or understand and agree with our Agile approach. The risk of being rejected by the organization led us to try some things that seem counter-intuitive from a traditional testing perspective, but which (in hindsight) do seem to be supported by the Agile values. The counter-intuitiveness of these actions is why we call them "mistakes". Since they appear wrong at first glance, we wrote this paper to defend them.

But first, we'll give the defense of our actions some context by describing our more straightforward application of the Agile values to the part that was under our control—the establishment of a new software testing team.

2 Using the Agile Values to Build a Testing Team

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 projects.

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, so naturally the first thing we were asked to do was to document what those processes would be. The company was working with an outside consulting firm to design a total development process to be used henceforth on all projects, and 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 believed our first priority should be team building, not process. We stacked the deck in our favor by hiring known team players, people we'd worked with before and whose skill sets we knew. We removed cubicle walls to create an open workspace, hoping

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

to encourage good "convection currents of information" and give the team every opportunity to collaborate and invent, following the philosophy of Alistair Cockburn [4]. We stalled on every request from the company to formally document our test methodology, feeling that such an action would limit our team's creativity. We set up a WikiWikiWeb[2] to allow everyone on the team to contribute to and modify our own methods and processes.

Without a process to tell them specifically what to do, the team became used to noticing and solving problems. For example, some test engineers developed a simple and powerful test framework in Perl to automate both command-line and GUI tests. Others created user interface standards guidelines. Many different contributions led to our system of task allocation using three-by-five cards and a Wiki-based task tracking system.

Our emphasis on interaction and teamwork helped us gain the respect of other teams. In a recent survey we sent to fellow managers, all replies to the question "What has Testing done right" mentioned our level of teamwork. No one in the organization below the director level even talks about the company-wide process designed by the consultants.

2.2 Working Software over Comprehensive Documentation

While it might appear difficult for a testing team to accept the lack of specifications or documentation that prevails at many Internet software companies, the Agile values reminded us that documentation is a poor way of controlling software delivery. We put the energy we could have spent requesting specifications, requirements documents, design documents and schedules into obtaining the code itself. We wholeheartedly pursued establishing a regular build for testing, our experience concurring with 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]

We worked with the configuration management developer daily, learning about server administration so we could build and maintain our own test environment, and also so we could tell exactly what changes developers were making and when. The developer helped us set up a clean test environment that was capable of accepting new code daily.

We took every opportunity to influence development to practice continuous integration and shorten the develop-test cycle. When developers wanted to wait until they're "finished" before they delivered to testing, we asked them if perhaps there's an order they could deliver the features in. We involved product and project managers and diagrammed the effect on schedule that waiting for a complete product would have. We called attention to the repeated occasions the schedule has been missed due to a particular component that's always developed without testing involvement. Sometimes our influence worked, sometimes it didn't, which is better than might be expected, and we think it's been worth the effort.

---

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

2.3 Customer Collaboration over Contract Negotiation

When two parties have their own goals and interests to protect, contract negotiation is the process they must use to come to agreement. When both parties' goals and interests are the same, and they know and trust each other, they don't have to worry about keeping the equations balanced, freeing them to make the most progress possible on their shared goals.

To get on the same team as the "customer" (Product Manager in our case), we aligned our goals with his. We shifted our goal of bug-free products more toward profitability for the company, and we asked him to share the goal of quality instead of delegating that goal entirely to our team.

We targeted open and timely flow of information as the best way Testing could contribute to collaboration and goal alignment between product management and the rest of the product team. Before the Testing team existed, the whole project team was caught up in a blame game because it wasn't realized until very late that there were problems preventing delivery. When we provided early and regular bug prioritization meetings, everyone remained appraised 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. Our original plan featured test automation as a way to shorten the develop-test-delivery cycle; we did not in fact shorten the cycle, nor have we automated the majority of our testing. We planned to expand our team size and our operations gradually until we were testing every product; this also hasn't happened.

But as Jim Highsmith says, "In a complex environment, following a plan produces the product you intended, just not the product you need."[6] We believe our implementation of testing has been appropriate to the circumstances, even though it didn't go the way we intended.

Our perception that it would be fruitless to try to control the way events unfold kept us optimistic and thriving during some dramatic changes in our environment. Products were canceled; others changed direction several times. Development managers came and went. A new usability/human factors team was thrown into the mix. We lost our manager and after a period of confusion, we were eventually transferred to a completely different management chain. Half our team was lost in company-wide layoffs.

Each change gave us the opportunity to adjust and improve our process. Layoffs brought the remaining testers closer together as we consolidated our separate, product-centered test teams into one team. The task of deploying updates to customers shifted from development to the system administrators, which resulted in our being asked to test the update process more formally. When a product's user interface was completely

redesigned, invalidating all our automated user interface tests, we took the opportunity to write a better test framework.

We realized that some pieces of our strategy were discardable and others were not. Two rounds of layoffs drove the point home: everything else could be flexible, but if we were to succeed in this dangerous environment, there were some things 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 following section describes our strategy for accomplishing these goals.

3 Using the Agile Values to Work With Other Teams

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 (and possibly to us.) Paradoxically, our very weakness allowed us to make several "mistakes" that were essential to our success. In hindsight, these “mistakes” are probably agile practices in disguise.

3.1 We Didn't Protect the Customer

It's Testing’s purpose in life to protect the customer, isn't it? Perhaps in a safer environment you can get away with this approach. We believed our position in the company wasn't strong enough for us to be effective defending the customer by ourselves. We knew from experience that sometimes releases move forward like juggernauts—and a few quality devotees trying to stop the release by throwing themselves under the wheels probably wouldn't even attract attention, much less accomplish anything for the customer. We were afraid that if we undertook the responsibility of protecting the customer, others wouldn't have to.

One of the first things we did was to make friends with the product manager. We deemed it best to get on good terms with him right away, so that later, when the pressure increased, he would continue to value our input even when we refused to make things easier for him by taking the responsibility to fail a release.

We chose to be a “credible, high-integrity reporter of information” (Lesson 159 from *Lessons Learned in Software Testing* [7]) instead of the defenders of quality. We rarely spoke out in defense of a bug (we rarely had to) but when we did, we were careful to phrase our words as providing more information, rather than as making a recommendation. By never joining battle, over time we became unassailable.

3.2 We Didn't Hire Testing Experience

All other things being equal, the more testing experience we could have gotten on our team, the better. Good testers do think differently from developers [8], and at this company the testing thought process was what was missing. So we should have hired some great, experienced testers and made that thought process part of the overall development process.

Not so fast! We were, after all, novices, trying something new that could quite easily be rejected. 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?

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 the testers in any way they could.

3.3 We Didn't Test Everything

We're testers! Our job is to test everything! How could we justify picking and choosing which products we tested? Perhaps in a safe environment, we would have tried to test every product, assured that management would give us plenty of staff and lengthen the development schedules to accommodate us. But back in the real world, we weren't established in the organization, we were unclear about our management support, schedules were aggressive, and we were fighting for every resource we had. The most important thing we could do for the products was to quickly demonstrate our value to the company, so that we could live to test another day.

We understood the importance of early wins when introducing something new. As described in John Kotter’s *Leading Change*, we needed to “win in the short term while making sure [we] were in an even stronger position to win in the future”. [9] We were

afraid that in trying to test as widely as possible, we could only deliver a mediocre (or worse) performance on everything. As we focused our effort more narrowly, we demonstrated 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 we pulled 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 the decision so clearly in advance 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?"

3.4 We Gave In Easily

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, and so on? Isn't that QA's job? We thought, probably not. 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. We 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 have been there for you, but we don't want to hold you up. We trust you!

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.

The make-up of the testing team greatly assisted this. The zeal that the testers showed for wanting to test everything was felt by the other teams. When a version of the product was released without sufficient testing it was truly a sacrifice for the testers. Everyone knew it and a commitment to allow for better testing on the next release was easier to acquire.

4 Conclusion

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.
- Valuing responsiveness to change made us pay attention to what we were learning and adjust our priorities.

Our first strategy on the project was wrong (Lesson 285 from *Lessons Learned in Software Testing* [7]). But because we were able to adjust our strategy as we went along, we became better at working with other teams. We shifted our priorities from the technical (test automation, complete coverage) to the social (securing support for our team, getting others involved in testing). To do this, we used some practices that seemed a little counter-intuitive to us at first, practices that might even seem like mistakes. In retrospective analysis, our actions are in fact supported by both the Agile and the context-driven testing school of thought, as summarized in Table 1.

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

Our "mistake"	Agile value	Context-driven testing lesson
We didn't protect the customer	Customer collaboration over contract negotiation	12 – Never be the gatekeeper
We didn't hire QA experience	Individuals and interactions over processes and tools	150 – Understand how programmers think
We didn't test everything	Ruthless prioritization	10 – Beware of testing "completely"
We gave in easily	Responding to change over following a plan	176 – Adapt your processes to the practices that are actually in use

Although not specifically mentioned in the Agile Manifesto, we contend that "Ruthless Prioritization" is an important part of Agile software development. Of course, prioritization is key to any type of project management, but the "ruthless" part seems to be a signature theme of the various Agile methods.

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 and testers struggling with introducing agile methods in this direction.

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.

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)