Agile practices in a traditional environment

Markus Gärtner

September 5, 2009

Abstract

While Agile Practices and their underlying principles and values improve Agile teams, their use in non-Agile environments also help traditional teams improve their process and working-habits. This paper gives an overview of Agile practices that helped in a non-Agile environment. From my two years of experience as a QA group lead in a more traditional environment I will report about the change process towards more successful projects motivated by the testers. Practices like test-driven development, exploratory testing and Agile planning applied in the right way help testers of traditional teams succeeding on a technical basis. Improved communication through personal touch as well as whiteboard discussions help overcoming human bottlenecks. Based on examples from a real world case study, I show how we overcame process thinking and contract negotiations without explicitly using the term Agile.

1 Introduction

In order to introduce you to the situation in more depth and make the following approach reasonable, this section outlines the historical view right before introducing some Agile practices.

During 2007 my company was bringing a new product to the market. Over two years had been spent on developing the product. It consists of three major sub-components written in three different programming languages and using different architectures. It has the ability for extensive customization. During our first customer project, I was a member of the testing team for the customizations that provided plug-ins and customized API calls.

My company was following a waterfall-like approach to software development. Development activities were planned separately from the testing activities in a phased way. Our programmers wrote the applications and used some developer tests before the software got to the tester. The testers were responsible for testing the software applications, bringing in test automation, which was then used to exercise the software before and after it has been built by another group which was responsible for creating the releases in order to deliver them to the customer. Over time this had led to a high separation of the programming and the testing process.

Based on my previous work we had stuck

to an approach using chained tests that prepared test case preconditions using other tests. Since this strategy led to slow tests we had an 18 hours painful process each time we needed to execute the regression test suite. We compensated this by having the tests run in parallel over night. The chained test approach also resulted in hard to reproduce test cases. Our mean time for having one test executed using this automation approach was down to around an hour. The feedback time for our developers was therefore very long since usual regressions involved at least ten tests to execute before a conclusion could be made. Additionally the tests were fragile due to data-sensitivity on the external interfaces of the application. Since most of the external communication to other sub-components was simulated the tests had to deal with preparing the simulators properly prior to triggering the system. Afterwards all received requests were checked from the simulators as well. This led to the situation that for example whenever something changed regarding balance handling, we had to adapt a vast amount of test cases that involved communication to the subsystem responsible to maintain the balances. Most of our testing was achieved using test automation by shell scripts. Since we were able to test behind the GUI, there were little to no manual testing activities.

My company decided to implement a new department for customizations for the product and I was asked to be a group leader for testing in the new department. Until then we had twelve testers busy with maintaining our test automation on that project. After the switch to the new department I was going to keep four testers plus myself. Therefore it was clear to me, that I needed to research improvements regarding our approach to testing. In addition our old approach was suffering due to a lack of design. This issue needed to be addressed in the new approach as well.

2 Decision making

Immediately after being appointed I started to read about different approaches. Since our legacy test automation had a lack of design I began with the well-known *Design Patterns* from Gamma et al [GHJV95]. While it gave me a very good starting point, I came to the understanding that test automation is software development. Therefore we would need to apply established development practices. I became increasingly excited with Agile methodologies and the underlying practices. Coming from a waterfall-like approach, the question that drove me was: "If Agile teams deliver successfully and often then how can Agile testers keep up the pace?"

Researching over this topic I went directly into the *Framework for Integrated Tests* (FIT) [MC05]. After reading the book, I wasn't quite convinced about it, but it seemed promising. On the other hand there had been about a quarter of a year passed since I started to research new opportunities in order to get our current testing approach from a ten person maintenance job to an approach that barely four to five people could handle. So, time pressure was slowly increasing.

During early 2008 I found a book which was being written at that time. It dealed with an answer about the initial question that drove me. Lisa Crispin and Janet Gregory had put early draft chapters from their upcoming book *Agile Testing* [CG09] on the internet and I volunteered to give them feedback on the content while I was learning from the experts.

From the insights I got from these early draft chapters, I was able to realize, that I needed to raise the point to the whole team. Therefore I sat back together with my four colleagues on the issue we were facing and we started a conscious decision process. There were three alternative approaches that we had identified and considered:

1. Implement a test automation approach using the legacy approach anew thereby overcoming the problems we introduced initially resulting in high test maintenance cost and the like.

2. Stick with a solution provided by one of the external companies we were working with. Based on JMeter they had brought up a testing framework for our product, which had benefits over the shell-script based approach, but also some drawbacks.

3. Implement a test automation approach on the basis of FIT.

Just shortly before I had been introduced into a systematic way to make decisions. We identified relevant aspects for our new testing approach. Among these were issues of traceability, maintainability, reproducibility and test execution times. Each aspect was weighted against the other and after that we tried to assign numbers of fulfillment for each of the three identified alternatives. In the end we had a numerical indicator for each of the three alternatives

Based on our bad experiences the first alternative ended up with the worst results. The drawbacks of the second one seemed to be too big in the medium- or even long-run. The approach based on the Framework for Integrated Tests had the best results. Since our company is working mostly globally all across the planet it seemed to be a wise decision to introduce FIT using FitNesse [FN]. The wiki-style editing and the possibility to execute tests over the web browser was a promising choice.

At that time it was clear to me that just bringing in some green field test automation would not be enough in the long-run. Improvements on the team structure and how we had been working together were necessary as well. On the other hand we just had one teammate with a background in programming. No one had real experiences with the framework, so the conversion effort would be a challenge for the whole team.

3 Introducing improvements

Having read enough about teams and working together I raised a point about the testing issues we were facing. The Agile testing quadrants (see Figure 1, [Mar04]) were a good basis for this. First of all I introduced the quadrants as two axes of a graph to the team. Then I asked the question where on this graph our current approach to test automation was positioned. Just considering the technology-facing to business-facing axis the team responded we were more technology-facing than business-facing. For the next step I asked the question what they thought we should do. The answer was a clear movement into the business-facing direction (see Figure 2). However, this discussion led to some drawbacks. If my team was focusing on business-facing tests, there probably would be a gap of technology-facing tests. This made me introduce our intent to our developers.

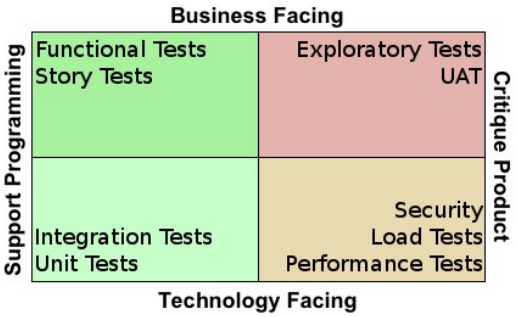

Figure 1: The Agile Testing Quadrants

Unfortunately I could not sell the implications of the business-facing approach to them.

When focusing on more business-facing tests, there is a high risk regarding the inner quality of the software. The Agile Manifesto prescribes "Individuals and interactions over processes and tools" and focuses on team thinking. Therefore I would have liked to intensify closer collaboration between developers and testers in the overall improvement project. However, I discovered that the developers were not going to support our new testing approach since they had enough work to do with developing the customizations and did not want the extra burden with our testing issues. In addition my suggestion for intensifying low-level unit tests kept unaddressed. We needed to establish this new approach exclusively with with testers in the team.

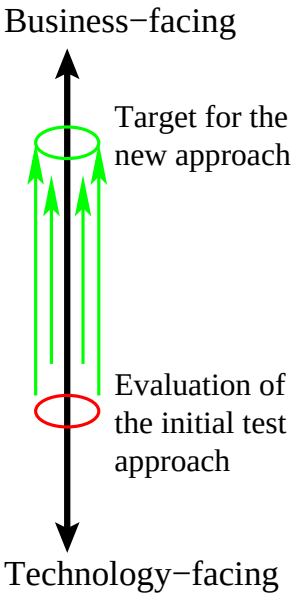

Figure 2: The identified target for the new test automation approach

The mission at hand was to work more effectively regarding test automation. There was a clear problem with the feedback times for our development team to succeed and the first set of improvements needed to address this issue. During my research I had crossed an add-on component for our product which was initially used for migrating old data from a legacy system to our new one. The interface makes use of describing a whole data structure in XML and thereby is able to replace seven or eight calls to the finer grained standard API. Using a spike solution I had already shown that we could improve significantly by making use of this additional component.

In addition we also needed to change how we had worked together in the past. Since I was dealing with *The Art of Agile Development* [SW07] at that time, I was influenced by the eXtreme Programming practices that are described in the book. This led to the first set of practices we followed. The book describes several improvements on the team morale that these Agile practices address. The team had just been working through a hard project and I wanted to protect my colleagues from this kind of situation in the future.

3.1 Iteration 1 – The Beginning

A team-based approach was the only alternative I considered. From the past I knew that we ended up with specialized areas of the business logic. We had one testing expert for balance-related behavior, one for the lifecycle-related areas, etc. In the past year this caused severe problems during vacations. In addition the project team helped out onsite at the user acceptance test team. For instance during one vacational leave of our balance-related specialist the behavior was overworked. During that time we had to keep our development groupleader patient until our expert got back from vacation. In order to reduce such bottlenecks we agreed on the Collective Code Ownership paradigm. Everyone should be able to get introduced to a certain area of the system as easily as possible and make corrections if the situation demanded for it. Human singletons should never occur. The expressivity of FIT-based tests helped a

lot here, too.

We already practiced version control usage and a build that executed on a single command. However we didn't execute the whole regression test suite automatically since it took about 18 hours to finish in the legacy approach. Therefore we also did not do continuous integration in an optimal way. In order to get an Informative Workspace, I ordered a huge whiteboard for our office. We were already sitting together, so there were just little improvements necessary regarding osmotic communication [Coc06].

In the beginning we started off by identifying the most risky parts of the system. We knew that maintaining these risky parts in the legacy test system was a cumbersome task and as soon as we could free ourselves from the high maintenance costs in these areas, the biggest time-savings would be possible. Since we knew the system pretty well after having dealt with it for more than a year, we sat together and went through a risk analysis. We started the overall progress of the new test approach by focusing for the first six weeks on the most risky parts of the system. We were able to identify three areas, where the biggest improvements could be achieved thereby freeing us from the burden of maintaining the old test cases in parallel. After two weeks we held a retrospective and identified how to improve our process.

After the first six weeks we had grown a testing framework, in which we were able to cover most of the risky use cases of our system. In order to get upper management buy-in I decided to hold a demo to my direct superior and some colleagues on the same management level as I am. I presented the system and received some skepticism about it. One reply was about language issues like slow test execution times. This one we had already dealt with by using the add-on component. We were able to create all necessary data for one test in less than ten seconds which had taken us in the previous approach about an hour. Another problem addressed the raised skill-profile regarding testers. By sticking to an approach that involved development work using a programming language the currently employed testers possibly could not be able to make necessary adaptations on the underlying fixtures. During the next few weeks we tried to compensate for this in our work.

3.2 Iteration 2 – The Cooperative Game

Early on I had noticed some potential problems with our approach. We practiced regular refactoring but without the support of unit tests. Since we were not practicing test-driven development up to that point we possibly were running into the same problems as before. It was clear for me that we needed to communicate our intents to the next generation of testers using unit tests. *The Cooperative Game* [Coc06] had made me aware of this issue.

Therefore I started to look into *xUnit Test Patterns* [Mes07] and was surprised to find descriptions of our previous flaws in the chapter on Test Smells. After being introduced to JUnit I was able to write clear tests which communicated my intentions even for the next person to work on the code. Slowly we improved our code applying the boy scout rule ("Leave the campground cleaner than you found it."). Regularly refactoring the code helped there, too. Among the team were about two and a half practicing programmers by that time. We stuck to a collaboration mode in which the less experienced programmers were preferably writing test pages. Occasionally we practiced Pair Programming with less experienced colleagues when introducing new features into the fixture code. Thereby we were

able to involve everyone from the team based on their particular skill profile.

In addition we automated our unit tests and incorporated them into Cruise Control [CC], the continuous integration framework that was set up about one year earlier by the group responsible for releasing the software packages.

In the meantime we had been introduced to another major project. Since we were not finished with our test conversion we had to deal with the project organization and the conversion in parallel. This increased the pressure on each colleague in my group. We made the decision to implement all up-coming tests for the project using our new FIT approach. In parallel we put aside one *sacrificial lamb* that was responsible to execute the legacy tests. If time permitted we were working on converting other tests, but this was the exception.

By that time we had already converted a majority of our old test cases, but we still needed to run our slow shell-script based tests with every new delivery to our customer. About each second week there had been a delivery to production. This meant we needed to execute them at least twice in two weeks. By focusing on the risky parts of the system and also the tests that had the biggest problems in the old approach we were steadily improving, but the sooner we could finish converting all test cases we would save that time. The legacy test suite wasted a vast amount of our time based on efforts necessary for test result evaluation. For each delivery we were put under high pressure to make an informed decision about whether or not to deliver.

3.3 Iteration 3 – Wrap-up

At some time in the middle of the third block of six weeks there happened a miracle. The project that had come across was put on hold as we had to wait for a major delivery from our product development department. There were about twelve gaps which needed to be filled before we could go on with it. At this time we decided to track the remaining conversion efforts on our whiteboard. We identified all the open points that still had not been converted. By bringing these up in a visual way it was also more motivating to deal with. In the last week I remember one colleague finishing the cards on the board in almost no time. Every time I came back from my meetings there was another card finished by him. In the end we were able to finish the conversion in about 18 weeks overall.

The team-based approach significantly led to a success. On the technology side we had exchanged a script based approach by one based on a programming language. By sticking to a simpler API for introducing test-relevant data into the system we were able to create all preconditions in about ten seconds which took previously about an hour. But the fashion how my colleagues had been working as a team made us responsive to the miracle of the project that was put on hold. We already had some business-facing tests automated that verified the risky parts of the system. The unrisky parts needed to be exercised from time to time when a new maintenance patch was delivered to our customer. The team took the opportunity to work down the backlog in order to be prepared for the time when the project was resumed. Using a visual story board for this was the best thing to do.

Before the summer vacation times, I realized that there were some open ends, which needed to be filled. First of all in order to get everyone into practicing test-driven development on the testing team I decided to pair program with my colleagues more regularly. We followed ping-pong programming, where the keyboard gets handed over whenever the previous unit test was made passing and the next one had been written. Beside the fact that this

is fun it also helps to introduce the programming language. On another side this improved our unit tests and I was able to introduce my colleagues into JUnit as a side effect.

The next thing I had to wrap-up was the way we had built our system. Next on our agenda was to convert our efforts in a way to make them applicable for the next customer. Early on we had been growing a framework by putting functions that were most likely to be useful for the next project into a reusable component. Therefore we already had been growing a test framework which was ready to be taken out of the box with just some minor efforts for the next project.

4 Outcome

4.1 First-class citizens

During the summer vacations in 2008 we were able to run the group for about one week with just one and a half persons, while the remaining colleagues left. The business-facing aspect of our test approach had made us agile so we could keep the pace. It was during this time that the developer's group leader had a question on how the software currently was implemented. He had prepared seven examples for a particular use case in our system, where he wanted to get customer feedback about the correctness of the current implementation. He came with this list of examples into our office and asked if someone could give him answers on these. One year earlier it would have taken us about one to two weeks to get results on these seven examples. With our new approach it was possible to answer his list in less than 15 minutes. We testers eventually had become first-class citizens compared to our previous struggles.

4.2 Serving the project

During 2009 I realized the impacts of our approach. My team was brought into a project which had been working for about half a year by then. It was an internal project and the requirements were pretty stable. Our customer was part of the marketing department so we could regularly get together and clarify open points. At the time we started to work on that project the requirements were well-known which gave us an advantage. There was practically no test automation done at that time and the project was under pressure to conclude with a user acceptance testing phase together with the customer. There had been many attempts previously to finish this off. We were given three weeks until the project should be brought to an end.

In order to get familiar with the business model of our customer and to bring the project forward we started with pure manual exploratory testing activates. This stressed out the still open bugs and kept up with the bug fixing rate of our development team. On the other hand this gave us the opportunity to get to know the business rules and understand the context of the project better.

At the end of the first week I was getting impatient. The tests I was exercising were pretty much the same all the time and test automation would really help us. Therefore I sat together with the other tester that was assigned to this project and we agreed to start the test automation approach by the beginning of the next week. Since we knew pretty well which elements needed to be tested and how much remaining work was waiting for us we were quite confident to finish this with about one month of effort. The only uncertainty for us was the new context.

At the beginning of the next week we sat together and identified elements of the project we were going to test. We decided to plan test

development work and fixture development work separately. In the end we estimated and prioritized the stories we had identified and compiled together our story board for the next few weeks. We decided to track our progress on a burn-down chart [Coh05]. Initially we had 24 story points of work in front of us. While working on the stories we had to extend the scope by another two story points. In the end we were able to use our previously built framework as a basis for the testing activities. We got test automation in place in just two weeks having only one and a half people working on it.

5 Conclusion

Agile practices and thinking help even in a more traditional environment. Local changes using Agile practices can improve the situation. This may raise attention for the Late Majorities [RM04] as well. Early feedback and regular reflection over the course build the basis for Agile methodologies. These principles also help in other contexts. Taking little steps like refactoring and test-driven development and small process improvements help to address impediments step by step and manage risks wisely. By making sure to involve everyone the best results can be achieved. This is even true if the feedback you get is more skeptic than supportive. The introduced case study shows that Agile practices can be used even for non-Agile teams. Even if the approach did not attract other teams immediately the learning I was able to make during this time was worth the whole effort.

A team can make itself amenable to *miracles* or not. By addressing high value/high risk, working with feedback and getting to stable points often, a team can appear to *make opportunities* when in reality, those opportunities are always there. A team just needs to be able to have just enough breathing room to be able to see them and take advantage. Regular reflection and involving everyone into decisions and improvements aids in the quest.

References

[CC] Cruise control. http://cruisecontrol.sourceforge.net/.

[CG09] Lisa Crispin and Janet Gregory. *Agile Testing - A Practical Guide for Testers and Agile Teams*. Addison-Wesley, 2009.

[Coc06] Alistair Cockburn. *Agile Software Development - The Cooperative Game*. Addison-Wesley, second edition, 2006.

[Coh05] Mike Cohn. *Agile Estimation and Planning*. Addison-Wesley, 2005.

[FN] Fitnesse - the fully integrated standalone wiki, and acceptance testing framework. http://www.fitnesse.org.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Mar04] Brian Marick. Agile testing directions. http://www.exampler.com/old-blog/2004/05/26/, 2004.

[MC05] Rick Mugridge and Ward Cunningham. *FIT for Developing Software - Framework for Integrated Tests*. Prentice Hall, 2005.

[Mes07] Gerald Meszaros. *xUnit Test Patterns - Refactoring Test Code*. Addison-Wesley, 2007.

[RM04] Linda Rising and Mary Lynn Manns. *Fearless Change - Patterns for Introducing New Ideas*. Addison-Wesley, 2004.

[SW07] James Shore and Shane Warden. *The Art of Agile Development*. O'Reilly, 2007.