A Lucky Shot at Agile?

By Zeger Van Hese

Senior Test Manager
CTG Belgium

A Tester's Tale of Agile Adoption

This paper describes life as a tester in a team of agile rookies: the context - a major healthcare IT company—and what agile did and didn't do for us. It describes the practices we adopted successfully, but also highlights mistakes and missed opportunities. Was our first shot at agile just a lucky shot or one that proved the value of agile methods?

Contents

This paper describes life as a tester in a team of agile rookies: the context—a major healthcare IT company—and what agile did and didn't do for us.

1. Introduction

As professional test consultants, we boldly go where destiny takes us. Or where the customer needs us, depending on which way you look at it. In our line of work, job content and work environment are things that are mostly beyond our control. Most assignments are challenges in one way or another, but once in a while a project comes along that is a really different—in a good way. A project that forces you out of your comfort zone. One that lets you experience both sides of the development spectrum, from waterfall style to agile development. These projects can mean a major change in testing mindset as well.

What do you do when an opportunity like this presents itself? You grab it with both hands, and don't look back. Well, actually, I *am* looking back now. I was lucky enough to be a tester on a project like this, a while back. And although tales of agile adoption are now pretty common, a couple of years ago there were not that many resources available about the subject. Our whole team was new to agile, which meant "learning by doing," all the way.

2. Goal

This paper describes life as a tester in a team of agile rookies: the context—a major healthcare IT company—and what agile did and didn't do for us. I will describe the practices we adopted successfully, but also highlight mistakes and missed opportunities. Was our first shot at agile just a lucky shot or one that proved the value of agile methods?

3. The Context

3.1. The Customer

The customer in question was a fairly big player in Healthcare IT by the name of Agfa HealthCare. Agfa is Europe's 10th largest software vendor—44th world-wide—and provides diagnostic imaging and healthcare IT solutions: radiology information systems (RIS), hospital information systems (HIS), picture archiving and communication systems (PACS) and appointment scheduling software, to name but a few.

Agfa became ISO 13485:2003-certified[1] in 2006, which basically meant that most procedures, tools and processes were already well established and that no additional action was needed to set up the test environment:

- A document management system with electronic approval workflows for all deliverables (Livelink)
- A bug-tracking tool (Rational Clearquest and an in-house developed tool called Qfindings)
- A test Management tool (Quality Center)
- A requirements management tool (Requisite Pro)

3.2. The Assignment

I started my test assignment at the Agfa HealthCare Scheduling department in the fourth quarter of 2005 on a project called IPlan—an intuitive, completely customizable web-based solution that allows hospital staff and healthcare providers to manage appointments quickly and easily. With IPlan, multiple resources can be planned across different departments, or appointments can be scheduled over longer time periods, for example in medical technical departments. It can be used by the hospital's central booking department and the entire hospital staff, but also by healthcare providers making referrals from outside the hospital. Patients benefit from improved pre-exam information, shorter waiting times, optimized appointment schedules and reduced hospital stays. The core of the application is the scheduling engine, which calculates possible appointment solutions taking into account optimal resource usage and constraints, both clinical and resource-related.

The main task I was given was basically "Test! Make sure that it works and is of high quality".

The main task I was given was basically "Test! Make sure that it works and is of high quality". Actually the second part of the expression could well be the result of some embellishment of my part—it's been a while. This may not sound too challenging but initially there was enough to worry about. It was the very first release of this new product, on a new platform that was quite unique within the company. The product was meant to replace the—hugely successful but not very user friendly—household scheduling brand in the long term, so we felt as if the future of scheduling was lying in *our* hands. I deliberately say our hands, since I was sharing my plight with a full-fledged verification team.

Agfa was by no means a barren testing wasteland. During the preceding years, a strong testing culture had developed within the company. There were quite a number of professional testers distributed over several business units. It was a 50-50 mix of external test consultants and 'native' Agfa employees. Our verification team consisted of seven testers, who divided their time between different projects. Three of us were considered the designated IPlan testers, but the actual number would vary when the workload in other projects got higher. I found myself alone more than once during the course of the first three major releases.

At the time, the team's development life cycle was pretty much sequential. We approached testing according to the V-model where possible. We prepared our scripted tests upfront while design and coding was ongoing. During the coding phase there were frequent internal releases, but these builds were not vertically integrated. As a result, we could only start testing newly developed features once they were completely finished.

The main release milestones were the following (Figure 1):

- Pre-alpha release: the first release delivered to testing. A release that is not feature complete but installable and (relatively) testable.
- Alpha release: the first release that is feature complete
- Beta release: a stable, feature-complete release ready to be tested at pilot sites
- Release candidate: a first releasable candidate

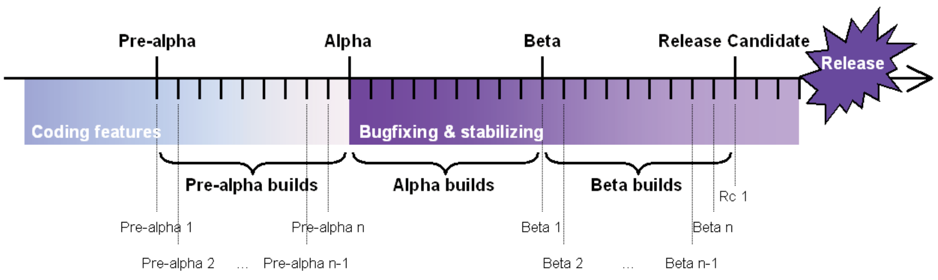

Figure 1: Release process

The verification team's testing approach worked—in the sense that we managed to control the chaos at first. We were able to execute our manual tests, but the amount of tests was exponentially growing as the application grew richer in content.

4. The Problem

In the year and a half after the start of my assignment (November '05 till June '07), there were three major releases of the IPlan software. The verification team's testing approach worked—in the sense that we managed to control the chaos at first. We were able to execute our manual tests, but the amount of tests was exponentially growing as the application grew richer in content. By the time the third release cycle was started, several issues came to surface:

- **Hidden assumptions.** There was an enormous backlog of use cases in our requirements management system. There was a common agreement within the team that this list was simply too large to implement in the release, so only the use cases with priority 'Must' were going to be developed. We only used these specific use cases as input for our scripted tests, which were well prepared in advance. It turned out that the developers didn't use the same input (Requisite Pro) as we did. They mainly based their development on face-to-face talks with the product manager that we didn't know about. It was as if there was a built-in aversion against using a tool. They regularly frowned when being confronted with the contents of our bug reports—taken literally from requirements—saying "but where did you get all this information?" They kept forgetting our answer. So there were a lot of hidden assumptions floating around. Product management thought we knew about the requirement changes, and we assumed the developers used the use cases as primary input.
- **Ever-changing requirements**. The requirements weren't very up-to-date, and the updates that were done regularly occurred without us knowing. In theory there was a synchronization tool in place between the requirements system and the test management tool, which flagged all requirements that were changed directly into Quality Center. In practice, we gave up on this synchronization after a database restore reset all revision numbers and all our tests and requirements kept being flagged as "newly updated".
- **Scope creep.** Lower priority use cases were regularly upgraded to 'Must', even after the first "feature-complete" alpha build. That meant features were developed of which we weren't aware. Most of the time we stumbled upon them by chance.
- **No test automation.** There was no test automation in place. No commercial test tools were available. An in-house automation tool was being developed at the time but for other platforms than the one we were using. Manual regression tests were our only option.

- **Alpha release was not feature complete.** The team didn't want to miss one of the first milestones, so we received an alpha build that was far from complete. Some features weren't fully finished until a week before the release date. We couldn't even perform exploratory testing on them upfront since the few things that *were* present didn't work at all. As a result, the beta testing phase was virtually inexistent, which had a huge impact on quality.
- **Quality issues.** The lack of proper beta testing resulted in sub-par quality. At the end, even major defects were postponed to future releases.
- **Delayed release.** The release candidates turned out to be pretty unstable due to regression issues. In the end, it took us 12 release candidates before the final version was released.

The team's software architect had been investigating some agile approaches and thought that the team might benefit from a methodology called ***Scrum****.*

Many of these issues were caused by poor communication between all parties involved and the fact that the team adhered to the original planning, even when the initial estimates were no longer realistic. Needless to say that in the end, the whole team was frustrated. We had this very promising product and we somehow felt that we were not unleashing its full potential.

5. The Trigger

In April 2007, in the midst of a frantic release process, our product manager organised a 'team summit' for all developers and testers involved with IPlan. The concept was new to us, but we liked the idea: a full-day of informative sessions to get a better understanding of each other's way of working. There was some time foreseen for brainstorming as well. All this took place in a rural setting far away from the office. It was a good occasion to reflect on what was going on and to propose some improvement ideas. There were many, but one proposal especially stood out from the rest. The team's software architect had been investigating some agile approaches and thought that the team might benefit from a methodology called **Scrum** (Figure 2). He described Scrum as an iterative incremental framework for managing software development, as a possible wrapper for our existing engineering practices. It could also be a way to improve communications and maximize co-operation in environments where requirements are rapidly changing. He proceeded to give us an overview of the common practices and the predefined roles within a Scrum team[2].

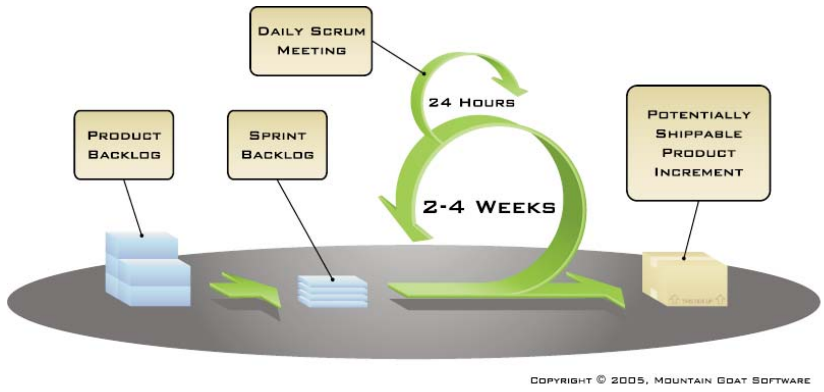
Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software

Figure 2: Scrum overview

Practices:

- Customers and testers become a part of the development team
- Frequent (2-4 weeks) intermediate releases ("sprints") with working functionality. This enables the customer to get working software earlier and enables the project to change its requirements according to changing needs.
- Transparency in planning ("sprint planning") and module development
- Frequent stakeholder meetings to monitor progress ("daily scrum meeting")
- No problems are swept under the carpet. No one is penalized for recognizing or describing any unforeseen problem.
- Workplaces and working hours must be energized—"Working more hours" does not necessarily mean "producing more output."

Roles:

- ***Product owner.*** The product owner represents the voice of the customer. He ensures that the Scrum team works with the "right things" from a business perspective. The Product Owner writes requirements (typically user stories), prioritizes them and then places them in the product backlog.
- ***Scrum master.*** Also known as the facilitator. He removes impediments to the ability of the team to deliver the sprint goal. The Scrum master is not the leader of the team (as the team is self-organising) but rather acts as a buffer between the team and any distracting influences.
- ***Team***. The team has the responsibility to deliver the product. A team is typically made up of five to nine people with cross-functional skills who do the actual work (design, develop, test, etc.).

We liked what we heard, since it seemed pretty easy to learn and it looked like it required little effort to start using. About halfway through this explanation everyone was already thinking about how we could make this work for our team. We ended the day with a nice meal. By that time, some agile seeds were planted—we only needed time for them to germinate.

Management had heard some good feedback about our Scrum information session and was willing to let us try agile development—with Scrum being the methodology of choice. And although becoming agile is often perceived as an abrupt paradigm shift, the team's resistance against this drastic change was miraculously low.

6. The Start

Two months after the team summit, a post-mortem meeting was held to evaluate, extract lessons learned, and formulate recommendations for the future. The general consensus was that there was room for improvement and that things needed to be different for the next release. Management had heard some good feedback about our Scrum information session and was willing to let us try agile development—with Scrum being the methodology of choice. And although becoming agile is often perceived as an abrupt paradigm shift, the team's resistance against this drastic change was miraculously low. That didn't come as a surprise, since we already made the switch (be it mentally) months before. It was July 2007, and armed with nothing more than the best intentions and a good deal of enthusiasm we braced ourselves for a wild set of sprints.

7. The Agile Momentum (Sprint 1-8)

The circumstances for the change were certainly right, organisation-wise: the rest of the original verification team had in the meanwhile vanished into thin air:

some members resigned, others were transferred to another business unit. This made the transition to one Scrum team a whole lot easier. The remaining testers and the product manager were integrated into the newly named "scheduling team."

Our new Scrum team consisted of:

- 2 testers
- 3 developers
- 1 software architect (who took up the Scrum master role since he was the original instigator)
- 1 Product manager (who took up the Product owner role; he also was the project manager and team lead ad interim)

*The team figured that **12 four-week sprints** would be enough to get the job done.*

There was no office space available for the whole team to be co-located at first. The developers were still located elsewhere, but within close distance. The product manager *did* move in with the testers, which I think is a luxury not many test teams can afford.

We organised daily Scrum meetings ("stand-ups") from day one, which earned us a fair share of odd looks from passers-by. The sight of men standing around in a small circle while taking turns in talking and sipping coffee *was* quite unique back then, it's become more common. But the benefits were clear from the beginning. There was instant communication going on. We knew what the developers were working on; they knew where we were at. The product manager was also attending most of the time, while it used to be very hard to reach him before.

The product backlog was assembled and some preliminary sprint planning took place. The team figured that **12 four-week sprints** would be enough to get the job done (see Figure 3). The testers were not invited to this planning game, but at the time I didn't really mind because we had enough work on our hands.

No one wanted to rush anything, so the first four sprints were fully dedicated to:

- Analysis
- Complete code refactoring (moving configuration part and application to a single codebase)
- Improving performance
- Improving memory usage
- Improving tracing and logging

There were good reasons for this approach. This would start us off with a clean slate and it would also improve code consistency—things got a bit messy with all these different developers implementing different modules over the years.

The priority of the testers in these four sprints was to get an automated test framework up and running, which we thought was absolutely necessary to make the incremental approach work. It turned out that the in-house developed test automation tool that we were never able to use before—by the prophetic name of triple A (Anonymous Automation Android or A^3)—*would* be able to support the IPlan platform with only a few minor changes. The tool was developed and maintained outside of our team, but the responsible automation engineer was very helpful and responsive. Within a day in sprint 1, we were making our first automation scripts.

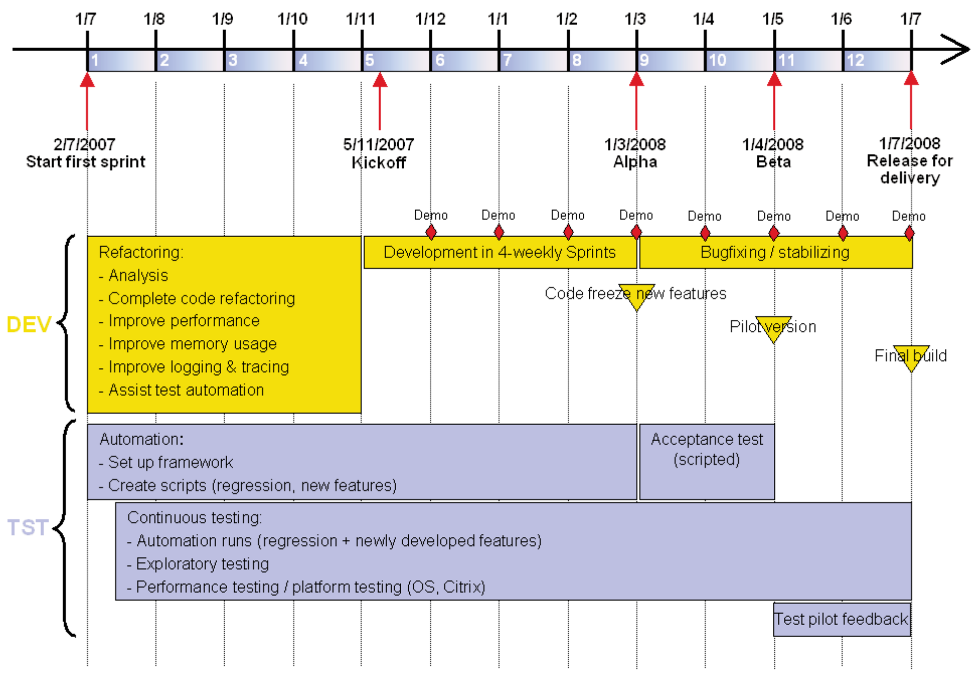

Figure 3: Sprints overview

By the end of the second sprint we already had a well-balanced regression set of around 300 scripts. We ran these scripts on the daily builds in full refactoring period, providing quick and valuable feedback. Refactoring ended with only 2/3 of the lines of code left, but performance and stability had improved significantly.

Triple A is a keyword-driven automation tool, which was ideal for us in the given situation. The philosophy behind the tool was that even non-technical users would be able to write scripts. So we could immediately start automating tests without real programming knowledge. Technical know-how was only required by the automation engineer that implemented the keywords. But it was just as easy to add customized C#-scripts when we needed special features, e.g. file operations such as comparisons or move/copy/delete. And we didn't have to wait for the first pre-alpha build either, we simply started creating a whole set of regression scripts against the latest released version.

We *did* encounter problems while scripting. Lots of them. We needed developer or automation engineer interventions regularly to be able to script actions on certain controls. But since the developers knew that the issues we encountered were blocking, they were given absolute priority. By the end of the second sprint we already had a well-balanced regression set of around 300 scripts. We ran these scripts on the daily builds in full refactoring period, providing quick and valuable feedback. Refactoring ended with only 2/3 of the lines of code left, but performance and stability had improved significantly.

The actual "new" development started in sprint 5. Still, no one from testing was invited to the sprint plannings, which I found a bit strange considering the fact we were all one team. When I mentioned this to the rest of the team they did not deem it necessary since the subject matter was mostly technical. By sprint 7 everyone got tired of my asking to be included, and the testers finally became a real part of the team. Not that we could contribute to any of the technical discussions, but it was valuable for us since we were really involved now. We were taken seriously and knew things first-hand instead of getting to know them incidentally by hearsay. It was also valuable for the team since our presence resulted in less misunderstandings and requests for clarification afterwards.

Knowing the ins and outs of what exactly would be worked on in each sprint made it possible to establish a certain routine in every iteration. In a sprint we typically performed following tasks:

- writing new automated test scripts for—stable—functionalities developed in the previous sprint
- launching our complete automated test set (regression + new features added in previous sprints) regularly and analyzing the results
- exploratory testing on new features as they became available. This approach enabled us to simultaneously explore and learn about the delivered software. It also gave the team an indication about the risk that was present in the software
- writing and executing manual acceptance tests for the newly developed features. We did this to have some coverage data for all new functionalities, since there was no traceability between requirements and our automated tests
- finding and logging bugs
- taking part in sprint retrospectives

We were almost feature-complete, and these new agile methods really did make progress visible. We delivered clear value at the end of every iteration.

When a large enough room became available to accommodate all of us, we didn't hesitate and got co-located immediately. The testers didn't have to rely on meetings anymore to get critical information. The information was now floating around, waiting to be picked up. Not much later a weird kind of euphoria kicked in: we felt like we had a momentum going, we were really on a roll. We were almost feature-complete, and these new agile methods really did make progress visible. We delivered clear value at the end of every iteration. At the end of each sprint, a validation session was organised for all stakeholders (application specialists, professional services). These demos were very well received.

8. The Agile Reality Check (Sprint 9-12)

Were we overconfident, hit by a false sense of security? I'm still not sure. What I *do* know is that things changed after the first—feature complete—alpha build was delivered. We weren't able to keep the momentum going. Our newly acquired Scrum habits started to fade away. Very subtly at first, but more apparent later on.

Reality dawned upon us. For every agile practice we had embraced before, there seemed to be another practice that was carelessly neglected or even abandoned. The sprint planning meetings were no longer organised, probably since there was no "new" development going on anymore. Stand-up meetings were no longer held daily. After all, everyone knew that it was more of the "same old, same old": bug fixing and stabilizing, while testing was working to get some coverage for the acceptance tests. The daily routine became a weekly routine, at best. The retrospectives disappeared, too.

We were working towards a stable beta build, to get it tested at a customer site by the beginning of the summer. Our product manager—who was also playing the customer, project manager, and team lead roles—had a hard time finding a suitable pilot site. Most sites could not free up any resources with timing being close to summer vacations. The customers that were available and willing turned out to be not a good match content-wise. We wanted them to specifically test our new features, and the proposed departments would only make limited or no use of that part of our product. The result was that the beta release was postponed for a month.

One potential pilot site finally agreed, but they demanded additional functionality, which the developers provided swiftly. After all, we were agile and that was what agile was all about. And we had some extra time on our hands after the beta was postponed. But we had lost a fair share of our agility when we started giving up on some basic Scrum practices. We were coding again when we were supposed to fix bugs and stabilize. We were embracing change a little too hard—the new elements were not even incorporated in any sprint planning. Testing was left out of the loop again. Developers were having a great time “being agile”, but we just felt confused.

Although the release date slipped, the overall results were quite impressive. The software had never been this stable, even the first internal builds were of an unprecedented quality. We actually had achieved more with less.

9. The Aftermath

At the time, the release for delivery seemed to trail forever. There were many release candidates due to the fact that even in the very last days new functionality was being added and changed on demand of the pilot customer. Results of such a practice could have been dramatic, but we eventually missed the final deadline by only a month. No harm was done on the quality side: the automated regression tests proved to be invaluable once more.

Looking back now, I think we did a pretty good job. Although the release date slipped, the overall results were quite impressive. The software had never been this stable, even the first internal builds were of an unprecedented quality. We actually had achieved more with less. More quality, more stability, more features, more satisfaction in less time and with fewer resources than before. The product reached the desired quality and contained the features that were stated in the project contract.

10. Hits & Misses

In moving from waterfall style development to Scrum, we had to adapt our style quite radically. Some agile practices were easily picked up by the team, while others never found their way into our work processes. Our attempt at agile was certainly not agile by the book. Here are—in random order—some highlights and practices we adopted successfully, but also our mistakes and missed opportunities. This is our list of hits and misses—the good, the bad, and even the ugly.

10.1 Hits

- **Quality**
  We had very high quality, from early stages on. We were able to improve significantly on performance, maintainability and extensibility, not only of the current version but also future versions. It was particularly striking that our first pre-alphas were exceptionally stable; this had certainly not been the case in the preceding years. I think the primary reason for this was that we resisted the urge to start implementing new features as soon as possible, but invested the time and resources in an extreme makeover.

- **Communication**

  The increase in communication played a major role in a successful release. From the moment we started with Scrum, much more cooperation and communication was going on. You could say that significantly more time was now being spent in meetings, but at least we were forced to speak up; there was no room for hidden assumptions.

- **Test automation**

  We had a pretty agile test automation process in place. Daily builds were created at the end of each day. The installation files were copied to our automation server and a trigger file launched our automated tests, all unattended. This proved to be a tremendous help. It ensured that the refactoring went smoothly in the early stages, and gave us very quick and valuable feedback about regression issues later on.

- **Refactoring**

  I guess that every programmer has a list of things he would have designed or coded differently for every project has participated in; this was no different with IPlan. Over the years, our original programmers had to make some compromises in order to get releases out the door in time. These compromises often took the form of less elegant or maintainable code. While these compromises won developers time up front with a quicker release, they cost us dearly in the long term, as it became harder to add or modify features. While the benefits of refactoring might be clear to programmers, it is not easy to demonstrate the business benefits of reengineering old code[3]. Rarely will a business have the luxury or willpower of choosing refactoring over new development. But in our case they luckily did, and it clearly paid off in the end.

We kept things simple, honoring the principle 'go for the simplest approach that works'. As the agile manifesto states: Simplicity—the art of maximizing work undone—is essential.

- **Continuous integration**

  From day one of the project, the team was able to put continuous integration into place. Our developers integrated their work frequently—usually each person integrated at least daily—leading to multiple integrations per day. Each integration was verified by an automated build, which detected integration errors as quickly as possible and provided valuable early feedback for us. Without our automated tests and continuous integration, the frequent releases would have created a huge manual testing burden.

- **Simplicity**

  We kept things simple, honoring the principle 'go for the simplest approach that works'. As the agile manifesto[4] states: Simplicity—the art of maximizing work undone—is essential. We didn't try to do more then we had to. We didn't engage in speculative planning, which usually just means rework. We basically worked for the needs of the current sprint, not for the sprints to come.

- **Use of tools**

  We used simple tools that did the job—or developed our own ones when other tools meant too much overhead.

  - We created a wiki, where all project data was centralized: work instructions, how-to's, test databases, a concise automation manual, etc. We pointed to existing documents where possible to avoid duplication.

- We made good use of James Bach's free tool Perlscript in support of our exploratory testing efforts. Perlscript is great for input attacks, but also to assess input fields really quickly.
- One of our developers created his own tool (aptly named TinySprint) to be used for sprint planning, task planning, and burn down charts. He decided to make one himself after other tools were just not doing the job for us. He literally generated it in a day.
- We started using an existing automation tool—no investment was needed. And the return was almost instant.

- **Exploratory testing**

  We had used exploratory testing in the previous releases, but not as conscientiously and focused as we did now. Back then we used it when there was some time left after our manual test runs, while now we used it as our main testing approach. The exploratory testing showed us the unexpected, unpredicted, and emergent behavior that went hand-in-hand with the system that was delivered. Our disciplined exploratory testing was an effective way to gather and provide feedback to the team, not only about finished features, but about work in progress as well.

Thanks to our automated regression tests and exploratory testing, we were now having a feedback loop in terms of days, while previously it was weeks or even months.

- **Validation session ("demo")**

  The project stakeholders felt much more involved than before when we started giving demo sessions at the end of every sprint. To make sure that they would attend the demo, we deliberately planned it every month, immediately after our Project Control Meeting, where we knew everyone would be present. These were no "pretend-demos", everything was really working. These demos also acted as validation sessions for the validation team. The benefits were clear: everyone knew what we were working on and what they could expect from the end-product. They knew the state the application was in at every point in time—transparency, also outside of the team. We saw their genuinely enthusiastic reactions and knew we were doing a good job.

- **Better feedback, faster delivery, less waste**

  As Elisabeth Hendrickson already pointed out in 2006, the key to becoming agile is to adjust your practices with three key ideas in mind[5]. In retrospect, without knowing these principles, we did honor them:

  - *Increase the rate of delivery.* Implementing Scrum automatically increased our rate of delivery. Internally, the deliveries were on a daily basis. Externally, we delivered a tested and releasable product every four weeks.
  - *Increase the rate and quality of feedback.* Try to shorten the feedback loop—this is basically the time between when a programmer writes a line of code and when someone or something executes that code and provides information about how it behaves. Thanks to our automated regression tests and exploratory testing, we were now having a feedback loop in terms of days, while previously it was weeks or even months.
  - *Reduce waste.* In Lean terms, waste is anything that does not add value for the customer[6]. We reduced our number of byproducts significantly. No scripted tests that weren't used, no excessive or unmaintained documentation: we put everything that we deemed important enough on a wiki, where we pointed to existing documents rather than just duplicating. We did make a test plan, but it was a living document with many links and sprint descriptions in the appendices.

- **Agile enthusiasm**

  Our team was full of agile enthusiasts, probably because all members had already switched to an agile mindset the moment Scrum was first introduced to us. There was no organisational resistance to agility either. Needless to say that this was an important factor in a smooth transition. We didn't have to invest any energy in convincing anyone of our new way of working.

Our team was full of agile enthusiasts, probably because all members had already switched to an agile mindset the moment Scrum was first introduced to us.

10.2 Misses

- **No user stories**

  Our requirements were not made in user story format. Our old (use case) format was kept because of the huge use case backlog that was already in the requirements database. We did search for an alternative to document user stories in digital format but no suitable solutions were found at the time. This was indeed a missed opportunity since user stories are generally regarded as *the* agile way of describing requirements, much more than use cases, because these often include details of the user interface. Including user interface details causes definite problems, especially early in a new project when user interface design should not be made more difficult by preconceptions[7].

- **No unit tests**

  Unit tests are generally considered as a must in agile projects. We certainly did not embrace that practice. I honestly do not know why there were no unit tests in place. I asked the developers several times, but all I got was an indifferent shrug and some mumbling about it being too late now. This is another good example of the developer testing paradox[8] in full effect. The paradox is the following: how is it possible that the practice of developer testing, which is so obviously right and so widely acknowledged as beneficial, and which could improve software quality and economics more than any other alternative, is still a rarity in software development organizations?[9]

- **Testing not involved in sprint planning up till sprint 6.**

  Although the test team was quickly integrated in the new Scrum team, we were apparently not considered as full-blown team members. Sprint planning was something for developer and customer roles only, because they didn't want to waste our valuable time and slow us down with estimates about development work and technical analyses. It took them six sprints to realize that testers can also contribute in these sessions and that there is a lot of crucial information to be found in planning meetings. The more the whole team comes to a shared understanding of the work to be performed, the better. Sprint planning meetings are about far more than just identifying tasks and putting an estimate on each. Sprint planning meetings are about discussing the work to be performed, understanding what the product owner wants, how we might collaboratively deliver that and how we can collectively address risks[10].

- **No test-driven development.**

  Test-driven development requires developers to create automated unit tests that define code requirements before writing the code itself. The tests contain assertions that are either true or false. Passing these tests confirms correct behavior as developers evolve and refactor the code[11].

In order to implement test-driven development, a solid unit testing practice should be in place. The two cannot exist without each other. The day the team decided not to make unit tests was also the day we gave up on a potentially test-driven development cycle.

In order to implement test-driven development, a solid unit testing practice should be in place. The two cannot exist without each other.

- **Too many combined roles.**

  Our product manager found himself between a rock and a hard place; he was combining several roles, often with conflicting interests. His principal role was that of the product owner, but he was also the team lead and project manager ad interim. We were promised a new project manager but it was really hard to get one with the right qualifications for the job. This was clearly not an ideal situation—it put a heavy strain on the product manager's customer duties, and also made him a biased project manager. The slack in the final release for delivery could have been avoided if the project manager was someone less partial. He wanted to accommodate the wishes of the pilot customer while his main focus should have been a timely release. This is logical—in his heart he always was a product manager; project management was a role he only agreed to play for a short time.

  The Scrum master was also a software architect and developer in the team. He took on important development activities as well as representing the team to stakeholders, facilitating Scrum activities and escalating issues to management. He wasn't able to put in as much development time as he wanted to. His development activities suffered from this double role. In hindsight, we needed a developer more often than we needed a facilitator.

- **Scrum and agile planning abandoned in last sprints.**

  We were a bit inconsistent when it came to agile conformity. In the beginning, we adhered happily to all Scrum practices, but all our goodwill vanished into thin air once all original product backlog features were developed. With some very important stabilizing, bug-fixing and beta-testing sprints left, we stopped having daily stand-ups, planning meetings and retrospectives. We fell back into our old routine of assuming that everyone knew what needed to be done and what everyone else was working on. I think the majority of the team saw Scrum as a tool to only manage pure feature development activities. But stabilizing and bug-fixing are also development activities, and everything surrounding pilot site testing and its follow-up should also be taken into account.

- **Manual tests still used.**

  There's truly a bit of overkill here. This is not considered agile at all, but at the time we couldn't establish any traceability between our automated tests and our requirements coverage due to some technical restrictions in the test management tool. We wasted precious time creating and executing these tests, but we needed some indication of coverage. We tried to get the problem solved for months. It was only fixed long after the project was finished.

- **Embracing change is not equal to creating chaos.**

  We embraced change, but in the end we also created chaos when we were supposed to consolidate and stay focused. We tried to stretch the limits of agility by adding new features that were not even in the original product backlog. Toward the project's end there was no real sprint planning, which meant there was no control over development activities anymore.

This gave way to some agile renegade behavior in the last sprints. Here's another missed opportunity: I think the release wouldn't have slipped without all those last-minute feature requests.

11. Conclusion

We started this adventure with the best intentions but with no agile experience. Our software architect was the agile advocate that started it all. Everything happened very fast, without much preparation or training; there were only three months between the first introductory Scrum presentation and the actual start. This was mainly possible because it was the right time and the right place for such an agile shift. We all wanted some kind of change, and implementing Scrum seemed the right thing to do at the time.

The overall consensus was that we did a great job. We provided more for less. Customer and stakeholder satisfaction were on a higher level than before. Yes, we could have done a better job. Looking back, we missed several opportunities to do better, but it still was an approach that worked for us in the given context. Important here is the notion of context—context is everything. The fact that it was beneficial to us in this context is no guarantee for future successes.

Were we agile? That's a tricky and possibly senseless question. People usually tend to get into the argument of trying to qualify a team as "agile" or "waterfall". I'm not getting into this argument—what matters to me is that it worked. I agree with Naresh Jain on this: agile is certainly not about merely following a set of practices; it is a culture or value system[12]. Our agile processes certainly weren't agile by the book, but a kind of real world semi-agile that fell into place quite naturally. We didn't necessarily follow all the right processes, but we changed the way we worked in the right way to get a positive change out of it. It's as Matt Heusser said: "Agility is not yes-or-no; it's more-or-less—and don't let anyone tell you otherwise"[13].

We tried and eventually got what we needed, not all we initially wanted. It's safe to say that we were lucky, but I like to believe that we were able to push our luck by providing a good agile culture and letting common sense prevail; by resisting the urge to start implementing new features and sticking to refactoring first.

One question remains: was our shot at agile just a lucky one? I'd like to answer that with the 20th century existential philosophy of Jagger and Richards:

You can't always get what you want
But if you try sometimes well you just might find
You get what you need

(M. Jagger / K. Richards, 1969)

We tried and eventually got what we needed, not all we initially wanted. It's safe to say that we were lucky, but I like to believe that we were able to push our luck by providing a good agile culture and letting common sense prevail; by resisting the urge to start implementing new features and sticking to refactoring first. The creation of a solid and reliable code base was instrumental in the success, as was the team. If we were successful, it was also because of the people, not necessarily just because of the processes we used.

In a way I am sorry that we will not again experience the same enthusiasm, excitement, and naïveté from our first agile adventure. Learning by doing all the way. It was a great experience with excellent results for our customer.

12. Selected References

- Wikipedia on Scrum (*http://en.wikipedia.org/wiki/Scrum_(development)*)
- Mountain Goat Software on Scrum (*http://www.mountaingoatsoftware.com/scrum*)
- Perils and pitfalls of agile adoption, Matt Heuser (2006)
- The Agile Manifesto (*http://www.agilemanifesto.org/principles.html*)
- Elisabeth Hendrickson—Agile QA/testing (2006)
- Alberto Savoia—The developer testing paradox (2005)
- Zeger Van Hese—Software testing—profession of paradoxes? (2007)
- Joe Lax—Make time to refactor (2002)
- Mike Cohn—Advantages of user stories for requirements (October 2004, InformIT Network)
- Poppendieck, M. & Poppendieck, T. (2003)—Lean Software Development
- Mountain Goat Software—Better together (July 2006)
- Wikipedia on Test Driven Development (*http://en.wikipedia.org/wiki/Test-driven_development*)
- Naresh Jain, Managed Chaos—Agile mythbusters (2007)

[1] ISO 13485 is an ISO standard, published in 2003, that represents the requirements for a comprehensive management system for the design and manufacture of medical devices.

[2] Wikipedia on Scrum

[3] Joe Lax, Make time to refactor (2002)

[4] http://www.agilemanifesto.org/principles.html

[5] Elisabeth Hendrickson—Agile QA/testing (2006)

[6] Poppendieck, M. & Poppendieck, T. (2003)—Lean Software Development

[7] Mike Cohn—Advantages of user stories for requirements (October 2004, InformIT Network)

[8] Alberto Savoia—The developer testing paradox (2005)

[9] Zeger Van Hese—Software testing—profession of paradoxes? (2007)

[10] Mountain Goat Software—Better together (July 2006)

[11] Wikipedia on Test Driven Development

[12] Naresh Jain—Managed chaos—Agile mythbusters, 2007

[13] Matt Heusser—Perils and pitfalls of agile adoption, 2006

For more information about CTG's Testing Services, please contact:

Zeger Van Hese
Senior Test Manager
CTG Belgium N.V./S.A.
Woluwelaan 140A
B-1831 Diegem
Phone : +32 (0)2 720 51 70
Fax : +32 (0)2 725 09 20
E-mail: *zeger.van_hese@ctg.com*
Mobile : +32 (0)497 45 95 94
www.ctg.com