Scaling Agile Methodologies for Developing a Production Accounting System for the Oil & Gas Industry

Harprit Grewal
CGI
harprit.grewal@cgi.com

Frank Maurer
University of Calgary
maurer@cpsc.ucalgary.ca

Abstract

This report discusses using Agile methodologies in what can be described as a medium to large scale project [1]. In this report, we will discuss the impact Agile methodologies had on the project over the period of two and half years and the lessons learned while scaling agile techniques to a relatively large team. We will discuss some interesting experiences – good and bad – encountered during the course of this project.

1. Introduction

Oil and gas companies use "production accounting" to account for products produced and the need to allocate the correct shares to various stakeholders. Production Accounting tracks oil & gas and their by-products as they travel through a complicated network of Wells, Batteries, Gathering System, Gas Plant, and End-point facilities.

This paper discusses the Production Accounting Solution (PAS), an enterprise application developed by CGI together with four upstream oil & gas companies, which automates many of the tasks of the production accountants. The PAS project is in its 3rd year of development. The current software development team consists of approximately 80 developers, business analysts, and QA staff.

The large number of stakeholders involved in the domain and the practices carried out by various players make the successful development of such a system very challenging. The company has made a number of prior attempts (each using non-agile methods) to commence the PAS project, but each failed. Given the increasing number of world-wide Agile success stories, the architects of the PAS project decided to use Agile practices in developing the solution.

The Production Accounting Solution was born out of a need to will replace four legacy systems, each of which were developed over a decade ago. None of these systems had all the features that would totally automate the tasks of the production accountants. More importantly, getting new features added to these systems and getting support for the existing features was getting increasing difficult. Some of the major players decided that it was time to agree on a common solution that would ultimately become the industry standard.

This project is an interesting case study as there are few projects of this size that have used Agile in its purest form. There are numerous reports on employing agile methodologies, but most of them have focused on improving a failing process or using agile from scratch on smaller sized projects [2]. What makes this project different is that it is one of the few projects of this size that has followed agile practices from the beginning and continues to do so.

2. Technologies and Methodologies

During the initial stages of the project (end of 2003 and beginning of 2004) we made a number of technical decisions: the project would be developed using the Java 2 Enterprise Edition platform; we would use Oracle as the database backend; Eclipse was adopted as the development platform; and we chose Concurrent Versioning System for version control.

Since the project was large and complex in nature and most of the areas were new to the project development team, Agile methodologies were suited to mitigate the potential risks. We believed that the biggest risk on the project was understanding the domain and delivering what the production accountants wanted. Having the

AGILE 2007
0-7695-2872-4/07 $25.00

business representatives work with the development team mitigated this risk by decreasing the feedback loop. We chose *eXtreme Programming* as the development process because the idea of developers communicating with the business users through stories and developing the system in small chunks appealed. However, there was some initial opposition to pair-programming as it was deemed to be expensive. The development team had to convince the sponsors that this was indeed beneficial and would pay in the long run. After some negotiation, the approval for pair-programming was given.

Each of the four sponsor companies provided between two to four full-time production accountants to work with the development team. These business users were co-located with the development team in an open environment where they shared the *pods* (a collection of work stations arranged in certain configuration) with the developers and were, essentially, part of the development team. We referred to these as *cross-functional* teams as it was the implied job of the production accountants to impart their knowledge to the developers and for the developers to explain, at a high level, to the production accountants what was involved in developing the stories so that the business can understand that some stories that look easy on the surface can have ripple affects on the system and they need to treat those stories as non-trivial.

We decided very early in the project to hold daily *Scrum meetings* to manage the progress of sprint and product backlogs. To ensure that the meetings were brief we insisted that team members only talk about relevant information and anything that needed extended discussion be taken off-line.

Test driven development was a major part in the development process. One of our development rules was that no source code be checked in without its accompanying tests. The tests were mostly functional tests and were mostly resistant to implementation changes. Initially, the tests were executed against the database, but as the number of tests grew, running the tests was becoming a bottleneck. In order to reduce run time and speed up the feedback loop from tests (currently sitting at about 16000 JUnit tests), we developed an in-memory version of the tests which gave developers feedback in less than fifteen minutes. Every effort was, and is, made to speed up the tests even more. The time spent in writing and running these tests was a worthwhile investment. Cruise Control rebuilt the system each night and ran the test suite. Any failures resulting from the nightly build were fixed as soon as possible.

3. How Agile Played Out

Large-scale Agile was a new experience for most of the development team. Most staff had worked on smaller projects with varying use of agile methodologies. PAS was not one hundred percent XP, as some tasks were trivial and did not need pairing.

We decided at the start of the project not to release the system into production after every sprint. The reason for this was that PAS was a replacement system intending to replace systems that have been in production for ten to twenty years and while there are strategies available to phase in the new system and phase out the old one, they all come with heavy costs of implementation, training. Plus, the sponsor companies wanted to avoid their users having to juggle between two systems. We decided that PAS would be made available after some milestones were achieved, the most important of which was that the system should be able to perform all the tasks that the current systems were performing.

For the first year, the development team was small comprising of eight to ten developers with the same number of business people. New developers were often hired in small batches, of usually 2 or 3 developers, in order for them to pair with the more experienced developers and assimilate the patterns used on the system. The developers were selected based not just on their technical abilities but their willingness to be part of a dynamic team and learn a radical new way of developing software. Due to this selection process, the turnover rate on PAS was extremely low. Even those who left the project said that they were happy with the project, but had to leave for other reasons. Overall, the development team was highly motivated and wanted the project to succeed.

4. Adding Resources to the Project

The development started in March 2004 when the two team leads and the development manager started implementing the framework on which the system would be developed. The first release of the system happened on April 2nd, 2007. Most of the development for the first release was completed by the end of December 2006.

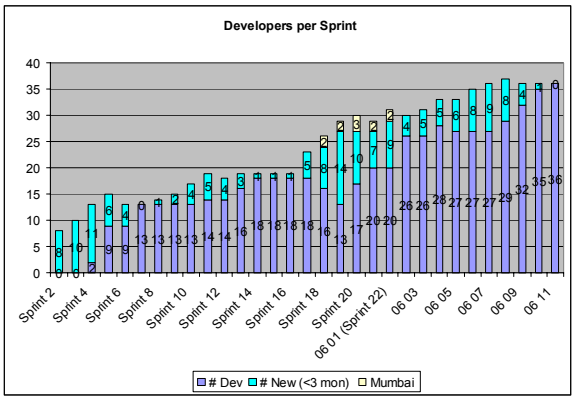

Figure 1. Developers on PAS

Figure 1 shows the number of developers for every sprint from the beginning till November 2006. For the first year, the development team was small comprising of eight to ten developers with the same number of business people. The project reached its maximum number of developers in July 2006. Hiring happened mostly in small numbers so that the new team members could more easily be integrated into the development effort. Not all the developers on the project did necessarily write Java code all the time. There are some developers who would develop tests using Ruby and some build WebFocus reports. The numbers also include the developers from Mumbai (India) who were part of this project for about 5 months.

The project was divided into three main streams focusing on three areas of the domain. The product owner was responsible for coordinating the activities between the teams.

A GUI team was used by the main teams as needed; and there was one team responsible for refactoring the system architecture and design.

This last team's main goal was to maintain a backlog of items that need to be cleaned up and to make the codebase more intuitive and manageable. We chose to have a separate team for refactoring because the developers did not understand the domain well in the beginning and there were parts of the system that were used by other sub-teams and could have had affected them. The refactorings team (usually senior developers) would look at parts of the system and complete the refactorings with minimal impact on the progress of the development teams.

Towards the end of the project, a performance team was formed to identify and improve performance bottlenecks of the system.

5. Scaling Agile Practices

5.1. Scrum of Scrums

As the team grew, the daily standup meetings were getting longer and losing focus. It was decided that each team would have its own individual stand-up and the team leads would have a Scrum of Scrums to discuss the cross-team issues coming out of their teams' stand-ups. This freed up the teams to focus their backlogs. Scrums again were meaningful for the developers as they could actually make sense of what the other members of their team were working on.

The teams did not run totally independent of each other. For instance, each team gave end-of-sprint demonstrations to the entire project, which was useful for sharing knowledge about new functionality recently added to the system by each teams.

The Scrum of Scrum was a little more formal than a daily standup. A team lead would tell others team leads what stories/bugs their team members were working on. They would also discuss the burndown charts and discuss issues that would prevent them meeting their sprint goals. This would allow other team leads to know if they needed to re-prioritize something in their own stream or if it would impact anything what they were working on. Each team lead would go back and share relevant information with their team members.

It was suggested, at some point, that a team member (not necessarily the team lead) from each team would go to other teams and relay what his/her team was working on so that they can be aware of any impact on their work. This was rejected by developers as most of the time the information shared was not useful. Therefore, we decided that the team lead would share highlights of what the other teams were working on and it was up to the people who might be affected to contact and talk to the relevant developers.

5.2. eXtreme Programming – When and How

PAS team leads strived to disseminate knowledge about the system by rotating team members between teams and mandating pair swapping every few days. As the project progressed and neared its deadline, teams sometimes chose to drop some agile principles. As

a result of this, some people started to work individually either to work on trivial features and/or bugs. Sometimes this became a habit. One of the team leads noticed that code quality was starting to degenerate as a result of lack of pair-programming.

Teams were asked for alternatives to pair-programming in order to maintain the code quality. The alternatives such as code inspection were rejected as boring, time consuming, and biased. All teams agreed that pair-programming was the best approach going forward.

Individuals sometimes worked alone for trivial changes. The senior members were given the implicit task of ensuring that *enough* pairing was going on in the team and that developers were pairing with not just the same person but with all the developers on the team.

5.3. The Workings of Agile Teams

As the project grew, the teams were still located on the same floor but worked in separate team pods. Each team had their own set of business representatives and testers. However, the developers were free to go to other pods and talk to other business users if they thought they would get a better answer there. As they arrived, new team members were introduced to the project during boot-camps which explained what the system was trying to achieve and what patterns were used in the system. During the course of the project, a testing course was offered three times and attendance for the new members was mandatory while the experienced developers were free to join if they needed a refresher.

We initially intended for team members to swap between teams in order to pick up knowledge from other areas, but, this did not happen. Each team was implementing a very specialized area of the system and the learning curve was too steep for the developers to leave their own field and efficiently pick up the workings of a different area. Team members focused on implementing functionality within their speciality and resisted swapping between teams.

6. Agile, PAS, and Offshore Development

In order to save costs while increasing the productivity of the team, management decided to recruit the services of the developers from CGI's India office. A pilot was started where three team members from India, (one development manager and two developers) joined the project in Calgary for three months to understand the methodology and the domain. They then returned to India and tried to work remotely. However, due to large time difference between Calgary and India (approximately 12 hours), it was difficult for the Indian team to contribute. Effective communication, one of the main pillars of agile approaches, was difficult to sustain as India developers had no access to the business people during their normal business hours unless they worked night shifts. Also, since there was no written documentation, conversation by phone for every little detail seemed awkward.

We tried many ideas to make use of the resources in India. For instance, India team was given the task of working on bugs. However, we found that fixing bugs was sometimes as complicated as writing new stories because the developers there needed information to fix the issue without making any assumptions. That too needed access to the business users, which was the most difficult task. It is to be noted that PAS used bug-cards to log bugs, rather than storing them electronically. Using the India team to fix bugs meant that we would have incurred the overhead of storing and maintaining them electronically, as well the possibility of the attached documentation becoming outdated.

Even though the developers in India were extremely talented and experienced, they found themselves unable to contribute fully as they did not have the necessary resources at their disposal. Therefore, we decided that the project would not use this facility.

Having the client close to the developers not only helps in developing code without making assumptions, it also helps productivity as the feedback loop is considerably shortened. Keeping the project completely localized in Calgary meant some higher costs labor compared to moving a part of it to India, but it was worth keeping the system as close to reality by getting continuous verification and feedback from the business users.

7. Effects of Scaling Agile on Productivity

Data was collected for the 32 sprints [1].

Sprint	Test Code Lines	Source Code Lines	Test Source Ratio
01	15889	9773	1.6

02	23094	14738	1.6
03	32149	21898	1.5
04	36611	26539	1.4
05	55028	30890	1.8
06	67741	39737	1.7
07	82498	51505	1.6
08	94690	55034	1.7
09	109534	64051	1.7
10	121446	71539	1.7
11	131968	80309	1.6
12	139354	85539	1.6
13	153269	97306	1.6
14	163423	106762	1.5
15	173817	114921	1.5
16	190395	122263	1.6
17	204126	151493	1.3
18	217843	160541	1.5
19	233657	170826	1.4
20	183442	183442	1
21	269634	194728	1.4
22	287591	208509	1.4
23	291345	217079	1.3
24	315160	227663	1.4
25	337281	235987	1.4
26	346227	248247	1.4
27	367101	262384	1.4
28	372190	265886	1.4
29	385159	270948	1.4
30	406993	279785	1.5
31	419903	296380	1.4
32	432932	309595	1.4

Table 1. Test lines and source lines and ratio

As can be seen from Table 1, it is clear that the amount of test code was always more than the production code. For every line of production code, there was anywhere from 1.3 to 1.6 lines of test code written. The only change in the consistency was from sprint 16 to sprint 17 when a lot of tests were removed that supported only a small amount of production code (and thus were deemed unnecessary and/or redundant). The number of tests and assert statements increased consistently over all sprints.

When new members were added, team productivity was decreased and less test and production code was created per developer hour. Experienced developers would have to pair with the new developers that caused their productivity to decrease. After the new team members familiarized themselves with the project, team productivity went up again to the average level.

Agile methodologies work better with a limited turnover in the team. On a project of the size and duration like PAS, people will leave. One intermediate developer left the project in September 2005 (Sprint 18) and four other senior developers left at the end of October 2005 (Sprint 19). This had an impact on the project as new developers were brought in and it took time to train them in the domain as well as the methodology that we were using. Not only were five developers lost, another five developers effectively were slowed down in training the new hires before they could contribute to their fullest ability.

Not every phenomenon in the data can be explained with other empirical data. Motivation can also play a big part in affecting productivity. In Peopleware, DeMarco and Lister note that "the major problems of our work are not so much *technological* as *sociological* in nature" [3]. According to them motivation is one of the key success factors in software development along with Communication, Staffing, and Low turnover. As developers work on a project for an extended period of time, their motivation tends to be reduced. Therefore, some decline in the productivity can be attributed to the drop in motivation of the experienced developers. Another explanation is that the system is becoming larger over time and that makes it harder to incorporate changes.

Effort or velocity was never really tracked formally for the most of the project. Story count was the only metric that was relevant for longest time on the project. A story was a piece of functionality as perceived by the business users. If the functionality seemed too large, it was subdivided in to sub-stories. The total numbers of stories were divided by the duration of the project and that was the goal of every sprint.

8. Summary and Conclusions

PAS demonstrates that Agile methodologies namely e**X**treme **P**rogramming and Scrum can be scaled to teams of 40+ developers. XP was adapted by allowing the developers to do trivial tasks individually. Sometimes Pair Programming was used as a training tool while at other instances it was used a powerful tool to turn ideas into high-quality code.

When Scrum for the whole team was getting unwieldy, it was divided into a Scrum of Scrums where the team leads would get together to discuss

the results of their individual teams' scrums. The quality of the system, as perceived by the customer, is of high standards. The motivation of the developers was very high. Even though this was the most difficult project that most of our developers ever worked on, they found it to be very challenging and personally rewarding to be associated with it. The sponsor companies' representatives would come over to the development site and express their appreciation for the system.

The project has been a refreshing, challenging, and a very good learning experience for a lot of people. It has proved that software development does not have to be chaotic process where developers have to burn their midnight oil (no pun intended) to get some functionality in the system with sparse or no testing - leading to even more chaos. One of the things that were desired towards the end was the need to develop a different metric for tracking the velocity of the sprints.

Test driven development has played a big part in helping prevent chaos from entering this project. It is not hard to image, for a project of this size, what sort of panic would have engulfed the team had there not been tests to keep the codebase stable.

PAS team members have embraced agile practice. This was mainly due to the selection of the team members. Developers were hired in smaller numbers so that each individual could slowly acclimatize into the team and in turn become the mentor for the next new member.

Using off-shore was a great idea – if it had worked - as it could have sped the development on the project while reducing costs. Off-shoring did not work out for PAS due to difficulties with synchronous communication due to different time zones.

It is important for the business decision makers to understand that software development is a unique and challenging process. Agile approaches bring radical changes to the way software has been developed. It may seem that pair-programming is expensive or that having no documentation is a problem, But when every thing is factored in, the long term gains are substantial – not only in terms of dollars and cents but also in building better relationships between the development and business fraternities.

9. References

[1] Grewal, H. S – "An Experience Report on Agile Development Practices in Developing a Large-scale Software System" (December 2006), A Term paper as part of Master of Science Program at the University of Calgary, Canada..

[2] Mann, C – "An Exploratory Longitudinal Case Study of Agile Methods in a Small Software Company" (June 2005), Masters Thesis, Department of Computer Science, The University of Calgary.

[3] DeMarco, T. and T. Lister – Peopleware: Productive Projects and Teams. Dorset House, second edition, 1999.
http://www.cs.wm.edu/~coppit/csci780-fall2003/presentations/22-peopleware.pdf