

2008

PACIFIC NW SOFTWARE
QUALITY
CONFERENCE



COLLABORATIVE
QUALITY

OCTOBER 13-15, 2008

CONFERENCE PAPER EXCERPT
FROM THE

CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

Distributed Agile

An Experience Report Joy Shafer

Abstract

This paper details our journey from chaos to concord while developing software as a service with a globally distributed team using Agile methodologies. It highlights the challenges and successes we experienced as we concurrently developed two small, first-version, online services using a development team that was located in Redmond and Moscow, and a test team that was located in Redmond and India. We encountered many interesting and difficult challenges, but were able to successfully overcome them and release high quality software on time, delighting our stakeholders. This experience report highlights our learnings, focusing on several critical success factors, such as well-defined processes, cultural awareness, continuous integration, open communication and relationship building.

Introduction

About eight months into our first Agile development projects, the Development Lead, the Program Manager Lead, and I, the Test Lead, attended a seminar titled “Agile meets Offshore.” The presenter, Ole Jepsen, who has considerable experience in this arena, told the audience not to try Agile offshore on “... first releases of complex and high-technology-risk projects, if your onshore development process is not in place, [or] if you don’t have any onshore Agile experience.”¹ My colleagues and I looked at each other, and the Development Lead said, “No wonder this is so hard. We’re doing all of those things!”

Six months later, we had fallen into a cadence of releasing both of our online services on time every two months. These were the easiest releases I’ve experienced in my fifteen-year career in software testing. We were even able to release a week early in one case. Our software quality and team morale was high, we worked productively in ‘round-the-clock’ shifts, and the offshore teams were truly an extension of the core team.

Some of us were enthusiastic about using an Agile software development methodology from the beginning. Our management was sold on Agile and sent us all to Scrum Master training. The basic tenets of Agile made sense to me. I had been in software development for many years, and shipped many different products. I had long been an advocate of involving test in the process as early as possible, and keeping the bug counts low. The concept of short, iterative sprints was new to me, but one I thought held promise.

Once the class was over and we went back to our office and tried to implement Agile, we struggled. There were many on our team who were rather vocal about how silly the methodology seemed (“Pigs and chickens? Please spare me!”). They did not want to track their hours. They did not want to spend the time needed in meetings to plan a sprint fully, and they did not want to be held accountable daily for the work they were supposed to be doing. There were also struggles with bringing the offshore team members into the process.

As a management team we patiently, and sometimes not so patiently, met every objection. We persevered and found solutions to most of the problems, eventually creating a very effective Agile process that the whole team embraced.

How We Used Agile

Wikipedia defines Agile Software Development as a conceptual framework for software development that promotes development iterations, open collaboration, and adaptability throughout the life cycle of the project.

Agile comes with its own set of vocabulary. We used a method called ‘Agile with Scrum.’ We did not use paired programming, test driven development or story points, although some of us were strong proponents of these methods, and we may eventually have adopted some of them if we were still together as a team.*

Agile is characterized by:

- Short iterative ‘sprints’ which generally last 2 to 4 weeks

Each iteration includes a detailed planning session at the beginning of each sprint, and at the conclusion of the sprint there is a retrospective to pinpoint areas for improvement, and a demonstration to stakeholders of what was accomplished during the sprint. By the end of the sprint, all features under development should be completely finished. That means not only coded, but also tested, and all bugs fixed and verified.

- Agile has its own terminology

ScrumMaster: the person who facilitates meetings, unblocks team members, and runs interference against those who would randomize team members

Product Owner: the person who makes decisions about what features should be built

Product backlog: a prioritized list of features or other desired or required work items, such as infrastructure improvements

- Intensive communications: daily ‘Scrum’

The Scrum meeting is a fifteen minutes daily meeting whereby the team members are held accountable for the work they committed to completing the day before. The scrum is also an opportunity for the ‘Scrum Master’ to discover and remove blocking issues.

*Our group was reorganized last summer into a larger non-Agile team, and many of the team members have moved on to different projects. One of our services was shut down even though it was successful with customers, because no one could figure out how to make money from it.

- Small teams and team focused control

A typical Agile team is about eight people. Ideally the team is co-located and able to communicate easily about their work throughout the day. Because of the structure of Agile, the team holds each other accountable for the work, and management should be less involved in making sure day-to-day work is completed.

- Division of work by product functionality rather than by task

As the Agile team works together to accomplish their objectives, and given the short timeframe of a sprint, traditional roles of development, testing, project management, technical writing, and so on tend to be shared across the team. Some sprints may be 'test heavy' in which case developers or doc writers or other non-test team members may lend a hand to the testing. In some sprints, testers or other technical team members may fix bugs or work on tools that would typically be done by a developer.

Agile works well when requirements are evolving, as new requirements can easily be accommodated at the beginning of every sprint.

Project Details

We were a newly formed group, charged with building two brand-new, version-one online services.

Windows Live VoiceMail is a service that forwards your telephone voicemail to your Windows Live (Hotmail) account. We were responsible for building the middleware piece that receives the voicemail from the Network Operator, translates it into a different format and forwards it on to Hotmail. It also kept the Hotmail inbox in sync with the Voicemail inbox.

This team consisted of Developers, Testers and Program Managers. We had six based in Redmond, five in India and three in Russia.

We worked with four internal Microsoft partners and two external partners.

Windows Live Call for Free was a service that could be accessed through a link from the Earthlink site (<http://maps.live.com>). It connected users with merchants by phone, either through a VOIP call or by connecting the user's phone line with the merchant's line through the PSTN (publically switched telephone network). Although the UI (user interface) was simple, extensive backend rules were implemented to make sure the service was secure and that there was no possibility for users or merchants to be spammed.

This team had twelve members based in Redmond, four in India and three in Russia. We interfaced with two internal partners and two external partners.

The offshore development team, which consisted of a lead and seven developers, worked for a small company in Russia. The vendor relationship with this company had been obtained along with the acquisition of a company called Teleo. All of the Microsoft developers on the

two teams had joined through the Teleo acquisition as well. The Teleo development team had been successfully working in a geographically distributed model for several years.

The Indian test team, which initially consisted of twelve people across both projects, was brought on board because there were no other test resources. Qualified testers are scarce, and there were few internal headcounts available to hire against. By the time the offshore test team was in place, the rest of the development team had been working on the services for four months and the target date for the first beta was only two-months out. Test was already considerably behind.

Challenges of Implementing Agile in a Distributed Team

Certain facets of the Agile methodology make it challenging to implement successfully in a distributed team environment. Some of the challenges we faced included:

Information Flow

One of the tenets of Agile is lean documentation. Things are only documented when there is a clear need and purpose for that documentation. When we started doing Agile, we found that the lack of detailed development specifications caused problems for the offshore team. A lot of detailed information was shared during our four-hour long planning meetings. Because of time zone differences, it was impractical for the offshore teams to engage in these planning meetings. The onshore team was too busy to relay all the detailed information to the offshore teams, leaving the offshore teams confused and ineffective.

Technical Communication Difficulties

In addition to the logistical problems with communication, there were other more basic issues. Most of the interactions with the offshore team took place over the phone. Initially we had a very difficult time understanding what the offshore team was saying. The voice quality of phone calls was often very bad. The offshore teams had unfamiliar accents and speech patterns. There were cultural and expectation differences which sometimes made it difficult for us to understand each other.

Round-the-Clock Work?

The differences in time zones were a big problem at first. Our main method of communication was email, and it would sometimes take days to resolve blocking issues because twelve hours passed between each question and answer.

If the onshore team did not complete something or get some piece of information to the offshore team by the end of our day, we would delay progress by a full 24 hours.

No Shared Vision

Partly because the teams were offshore, and partly because they were vendor teams, we found it difficult to share the business reasons behind the project activities. The offshore team was often working in the dark without the knowledge they needed to make decisions independently. Some of the tasks we asked them to do made no sense to them.

No Opportunities for Socializing

Trusting relationships are the foundation of team productivity. One of the best ways to build these relationships is through casual interaction, sharing details about our lives that let people know who we are and why we do what we do. It was impossible to get together with the offshore team in a non-business setting, therefore these relationships did not get built.

Additional Challenges

In addition to the above challenges, which are characteristic of a distributed team, we had other difficulties to overcome.

Initial Lack of Support for Offshore Test Team

I was hired as the first Microsoft test person on the team. When I was brought on board, the twelve-member Indian test team had already been working for a month, but they had accomplished almost nothing. They had no access to the Microsoft environment and they had spent their energies trying to build their own lab to mimic our facilities, but the environment was complicated and difficult to setup. After a month of frustration, they still did not have a working test environment and we were only five weeks away from our first scheduled beta release. Not a single bug had been filed.

The Microsoft contact for the Indian test team was a very busy developer who did not have the disposition or desire to help them. He was frustrated that the offshore team was demanding so much of his time, and the offshore team was stymied by his terse answers and lack of follow-through.

Lack of Access to Resources:

Initially, neither the Indian testers nor the Russian developers had access to the Microsoft corporate network. This created a huge overhead for our team, as someone had to send them files of everything they needed to work on, and the onshore team was required to spend a lot of time merging the Russian development teams' code changes into our code base.

Ineffective Onsite Coordinator

The plan for the Indian test team from the beginning called for an 'onsite coordinator,' a member of the Indian vendor team who could act as the liaison between the offshore test team and the onshore team. However, visa issues prevented the chosen onsite coordinator from being able to come to Redmond. (He shared his name, Amit Kumar, with millions of other Indians, making the visa process very time consuming. Amit Kumar is the Indian equivalent of Mike Smith.)

The delay in having an onsite coordinator caused considerable frustration for both teams, so the vendor company sent an alternate. Unfortunately, the person they sent did not have the temperament to deal with the strong-willed people on the Microsoft team and he was lacking in basic troubleshooting skills and systems knowledge. He could code, but he didn't

know anything about SQL administration or Web Services, and he lacked the ability to unblock himself when he became stuck.

Communication Channels Not Yet Established

When I started, there were two half-hour meetings each week with the offshore test team, one for each project. These meetings were ineffective because of the difficulties in understanding each other and the need for them to ‘try out’ our suggested solution before they could come back and ask more questions.

Initially, even after the offshore team was set up with Microsoft email accounts, they were not in the habit of using them and would miss important emails about the project.

Little Visibility into What Offshore Test Team was Doing

I had twelve unpronounceable names reporting to me on the organization chart, but there was little visible productivity from the offshore team. This is particularly a problem for test, which traditionally is difficult to measure.

Lack of Trust

Going along with the lack of visibility into what the offshore team was doing, came a lack of trust. I was under a fair amount of pressure to make sure the services were tested and ready for the rapidly approaching beta dates. The communication difficulties, cultural differences, and lack of a real relationship with the offshore team, led me to mistrust them. I wondered if they were even honest. The list of people working for them included ‘Amit Kumar’ and ‘Kumar Amit.’ I thought they were trying to pull the wool over my eyes, but I later found out these really were two different people. I also questioned whether they knew what they were doing. Most of the documents they sent were not what I was expecting. I’ve since learned that their documentation was done in a very typical Indian style, but my initial conclusion was that they did not understand test at all, and were trying to cover up their lack of expertise with volumes of fluff.

Best Practices for Distributed Agile

We found solutions to most of the problems we encountered. In this section, I’ll highlight our learnings.

Build Offshore Team Incrementally

Don’t staff up until you’re ready to support an offshore team. When you have someone onboard who is willing and able to support an offshore effort, you can bring offshore workers on effectively. However, if there is work that needs to be completed before all team members can be productive, such as building a lab environment, bring only a few people onboard at first, and only add the rest of the team when you are able to support them and keep them productively busy.

Bring Offshore Onshore

The offshore test teams did not start to show results until we had good technical communicators from offshore come to Redmond to become dedicated conduits of information for the offshore team.

After about two months of working with the offshore test team, I was fed up. The onsite coordinator they sent was ineffective. The offshore team was finding few bugs and I had little confidence in their test results. The documentation they sent was subpar. I was ready to sever our relationship and start over with a new vendor team.

Due to earlier complaints, the vendor company with which we were working had already sent a second onsite coordinator to assist the first one they sent. She was a little better at communications, but she didn't know test. We sent the first onsite coordinator back to India, and the second followed shortly thereafter for personal reasons (family emergency).

I set up a face-to-face meeting with our vendor representatives in Seattle and we laid out our grievances, rather bluntly. They listened to us, thanked us for being candid and promised to try their best to set things right. We were skeptical, but willing to give them another try, especially since our beta date was fast approaching and we knew that starting the process of engaging a new vendor would take time. The two projects were different enough that I realized we needed a dedicated resource for each project to handle the communications with offshore, someone who could do a deep technical dive and really understand what we were doing,

At this point in time, Amit Kumar's visa finally came through and he arrived in Seattle, followed shortly by Amit Sharma. Both of these men were the test leads from the two offshore teams. Although the initial contract had only called for one person from offshore to be housed in Redmond, our vendor company did not charge us for the additional onsite resource. They wanted our business and were willing to eat the extra expenses to make things right for us. Both of the people they sent were bright, hard-working and technical. They set up a system whereby they called the offshore teams every night and spent a half hour or more going over the day's events and making sure the offshore teams knew what was expected of them.

The offshore teams summarized their days' work and sent it via email to the onsite coordinators, who acted as the offshore teams' proxies at the scrum meetings.

The most important thing we did to make the project successful was bringing dedicated technical resources onsite for each project. These people already had relationships with the offshore team members and were able to build relationships with the Microsoft teams. Both of these men were technical enough to make the offshore team understand the complex details of the software under development.

Build Relationships

Another thing that happened about the same time that helped with my acceptance of our vendor company was that their offshore project manager came to Seattle for a visit. I had spoken with her many times over the phone, but my impression of her changed considerably when I met her in person. Over the phone, long distance, with her unfamiliar accent and

unusual verbiage, I assumed she was young and inexperienced. Once I met her in person, I realized she was a competent, accomplished professional with goals similar to mine. In fact, we have built the roots of a relationship that will probably outlast our professional association.

Throughout the lives of these projects, we maintained onsite representatives from offshore. These typically switched out every three months (we insisted on telephone interviewing the replacements to avoid the first couple of disasters). Because our relationship lasted for a number of years, most of the offshore team members had the opportunity to work onsite. They came to understand our culture and 'how work gets done' at Microsoft. We learned that the Indian team overall was smart, honest and hard-working, and they brought their positive impressions of us back to India with them as well.

Unfortunately, due to budget constraints, the Microsoft team was unable to visit the team offshore, however, I think it would have given us a deeper understanding of and appreciation for our offshore workers, and it is something that I highly recommend.

Multiple Communication Channels

Establishing multiple communication channels was very helpful in fostering productivity. Much of our communication was done via email; however, many team members would send Instant Messages (IMs) to each other with questions if they found their counterparts online. We had regular telephone meetings, and occasionally scheduled video conferences as well. Sometimes we would take photos of whiteboard diagrams to send to our offshore teams.

Cultural and Time Zone Awareness

About three months into the projects, I hired three Microsoft testers for our teams. Although I did not specifically look for Indians, two of the three people I hired were Indian. By chance, one of our developers was Russian and we also hired a Russian Program Manager (PM). These folks were able to communicate effectively with the offshore teams, and to explain cultural nuances to the onsite team members.

The time-zone differences became less of a problem once everyone got into the habit of making sure their work was ready for hand-off by the end of the day. Simple awareness of the time zones made a big difference. The overlap in time zones between the Russian developers and the Indian testers was a bonus. They often instant-messaged back and forth with each other.

Common Code Base / Document Repository / Tools

Having a common code base and document repository was essential to making the offshore teams effective. It took a very long time to get Microsoft corporate network access for the Russian developers, but our new Russian PM made this happen shortly after he came onboard.

The problem of lack of visibility was solved by having the offshore team members enter their hours directly into our scrum tool. Every day we could see exactly who spent their time doing what, and along with tangible evidence of their productivity such as test cases, bug

reports, test automation, and metrics reporting, my confidence in the offshore team increased dramatically.

Partners Must be Flexible

If our vendor company had not been so willing to go the extra mile for us, we would not have been successful. The vendor company must be willing to try new processes and procedures and not be too dedicated to one way of doing business.

Allow Time for Processes to Mature

Assume that it will take several months for problems to surface and solutions to be found. Plan time in the schedule for communication channels to be worked out and for working processes to be put in place. Don't assume instant productivity and a smooth transition for an offshore team.

Open, Honest Communication

We were blunt in our communications with the vendors. We told them our grievances without sugarcoating them. We encouraged feedback from them as well. Both vendor teams were receptive and responsive to our complaints.

Other Best Practices

In addition to the problems we encountered in working in a distributed Agile model, there were other problems we ran into that are likely to crop up even with co-located teams. I will not go into the details of the problems, but I would like to highlight a few additional best practices that we discovered.

Continuous Integration

Once we finally got both the offshore teams connected to the Microsoft corporate network and we invested in infrastructure so that we could automatically build, deploy to a test environment, and run build verification tests (BVTs), our velocity increased dramatically. Before releases, we built twice a day. The morning build would include all of the Russian developers' check-ins from the night before and would get tested by the team in Redmond. The evening build would include the check-ins done during the day by the Redmond team, and would get tested by the offshore team over night.

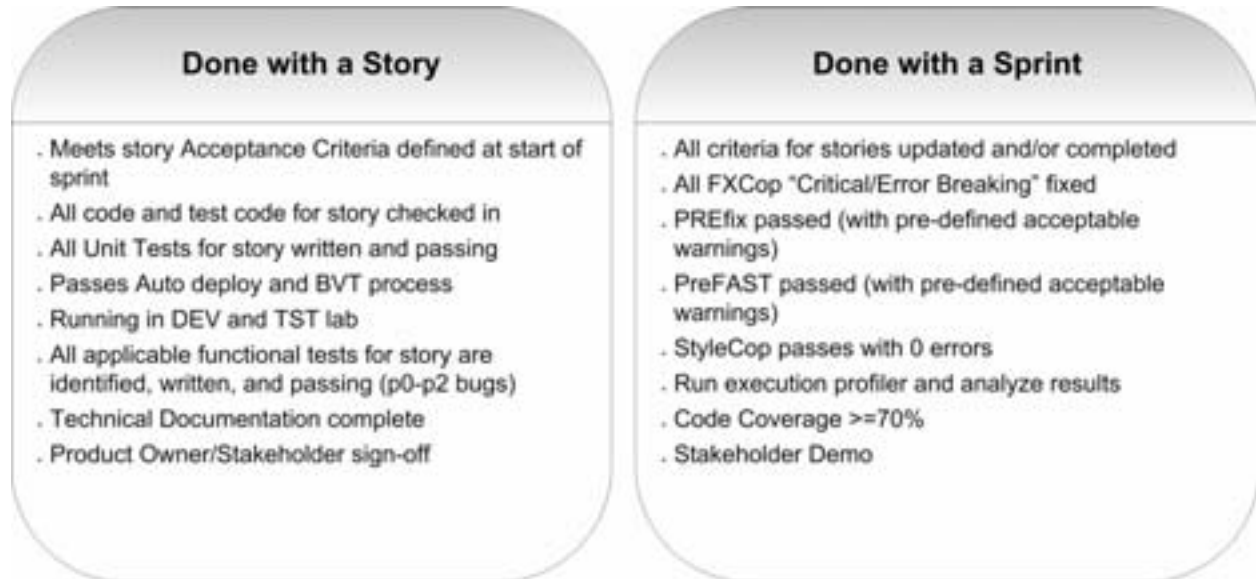
Continuous integration allowed us to create and maintain a very stable code base, making both new feature development and subsequent releases much easier.

Understanding 'Done'

Our team struggled with the meaning of 'Done.' Early on, if the developer declared the code finished, the story was deemed complete. However, we then found ourselves continuing to work on tasks related to the same story into the next couple of sprints. It seemed impossible to both code and test a feature completely within the same sprint.

Our velocity increased, which helped, but the main thing that brought us to the point where we could finish a complete story within a single sprint was getting really clear on the done criteria for the story, and making sure that all of the attendant tasks would fit within the sprint. We defined our done criteria (acceptance criteria) within our sprint tool and our PM Lead became very particular about making sure the criteria were complete and detailed for every story on the first day of every sprint. Of necessity, we had to break our features up into smaller pieces so that the entire development cycle could happen in a two-week timeframe.

Defined 'Done' Criteria



Notes: P0-P2 refer to bug priority, P0 meaning 'fix immediately.' FXCop, PREFIX, PreFAST and StyleCop are all internal Microsoft tools which check for code and security flaws.

Continuous Improvement

Retrospectives: Early on in our adoption of Agile, our retrospectives were hour-long meetings that yielded volumes of suggestions for improvements. Some items, such as 'there was not enough time to finish testing,' appeared sprint after sprint. As time went on and our processes matured, we had fewer and fewer things to discuss at the retrospective. Our retrospective shrank to fifteen minutes, and we usually did them at the very beginning of our sprint planning session rather than in a separate meeting. Because the team was committed to continuous improvement, we never abandoned the retrospectives, and even once things were going very well, there were sometimes great improvement suggestions that came out of the retrospectives.

Investing in infrastructure and process: Early on in the formation of our team, it was very difficult to make forward progress on software development. It seemed that for every hour spent productively, at least two hours needed to be spent on support tasks. We realized that we needed to invest in automation and process improvements in order to cut our overhead time. We took this back to our Product Owner and convinced her that we needed to invest

some of our energies in infrastructure otherwise we'd never be able to reach the productivity levels that she wanted. Some of the investments we made were:

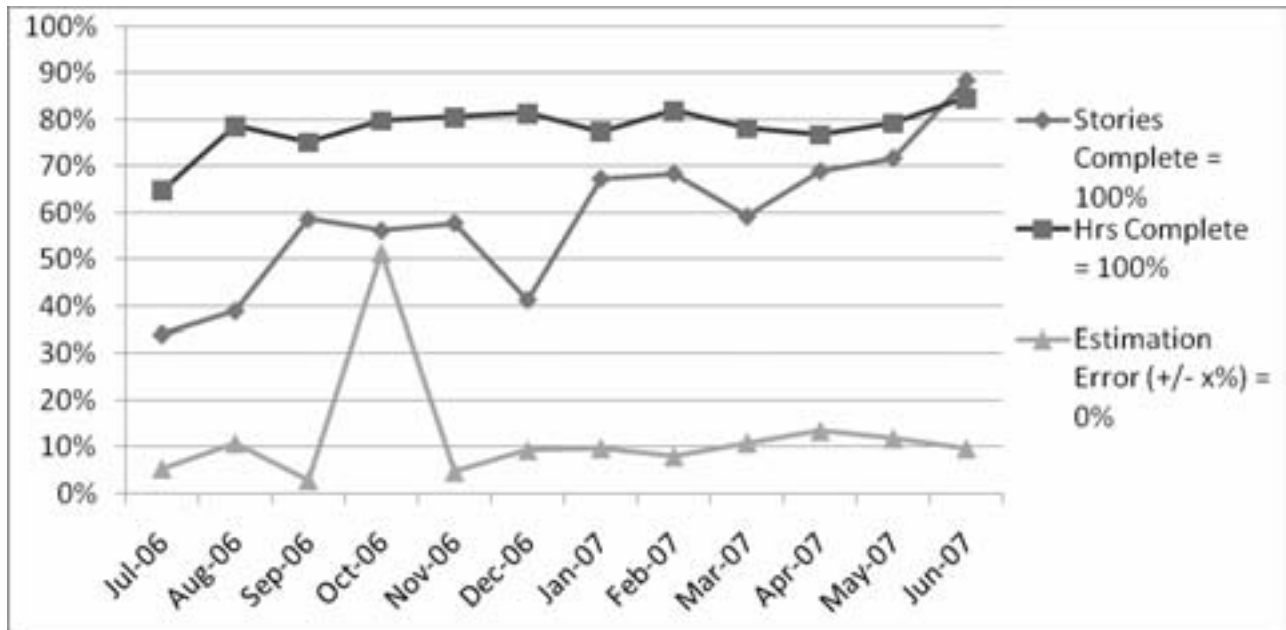
- Building an automatic build/deploy/BVT system
- Building emulators for all of our external dependencies
- Investing in lab infrastructure to increase our flexibility and minimize resource contentions
- Hours spent in meetings hammering out what 'Done' meant
- Hours spent in meetings discussing process improvements

Metrics for tracking progress: Our Product Owner believed strongly in tracking progress through metrics, and most of the team agreed with her. We defined a basic set of metrics to track and report. Some metrics were reported for every sprint, some for every month, some for every release, and some we only tracked quarterly. For each release metric, we set a minimum bar that we needed to meet before we could release, as well as a goal metric for which to strive. We began to see improvements in the metrics almost immediately and this lent us motivation to try even harder.

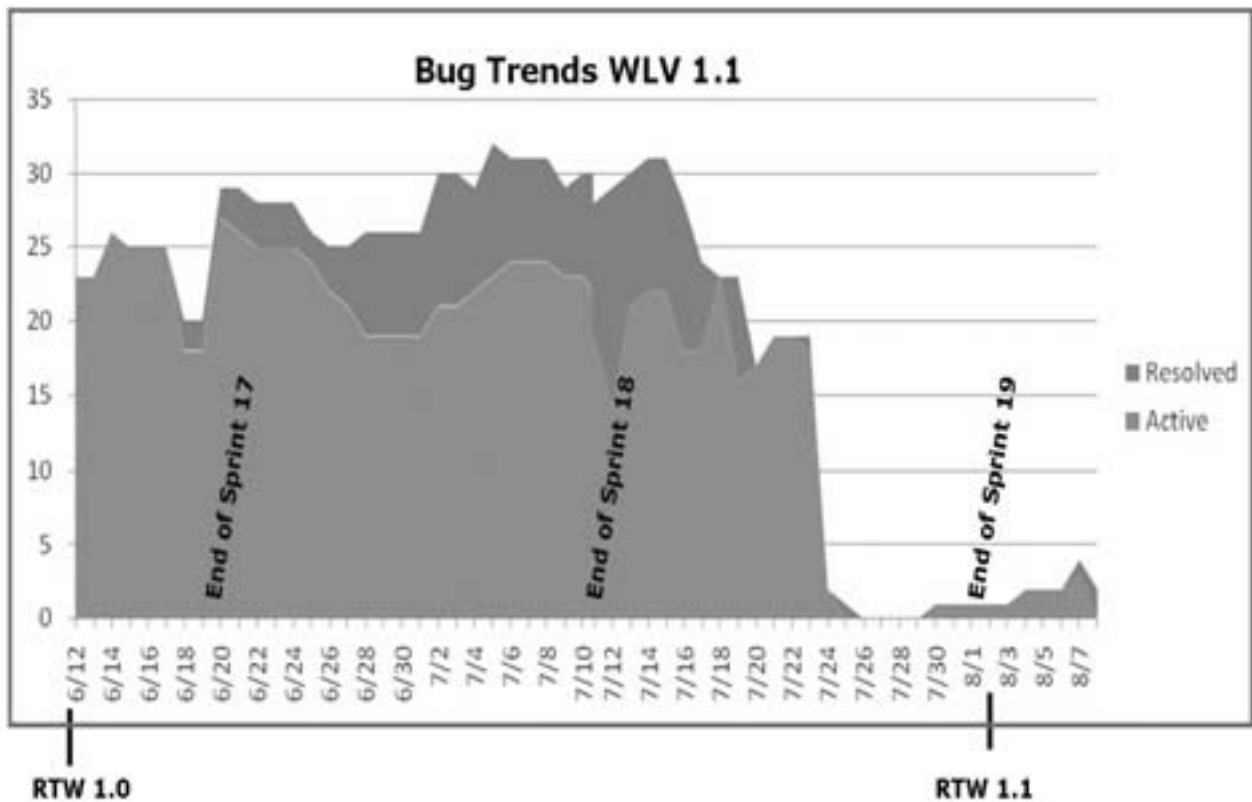
Some of the metrics we tracked included:

- Bug metrics: bugs found per release, resolution of bug, priority of bug, feature area of bug, etc.
- Code coverage
- Test automation percentages
- Test results
- Code complexity
- Unit test coverage
- Percent of stories completed in each sprint
- Percent of committed hours completed in each sprint
- Customer metrics: % up-time for the services, % positive feedback from customers, average latency of transactions, etc.

Sprint Estimation Metrics

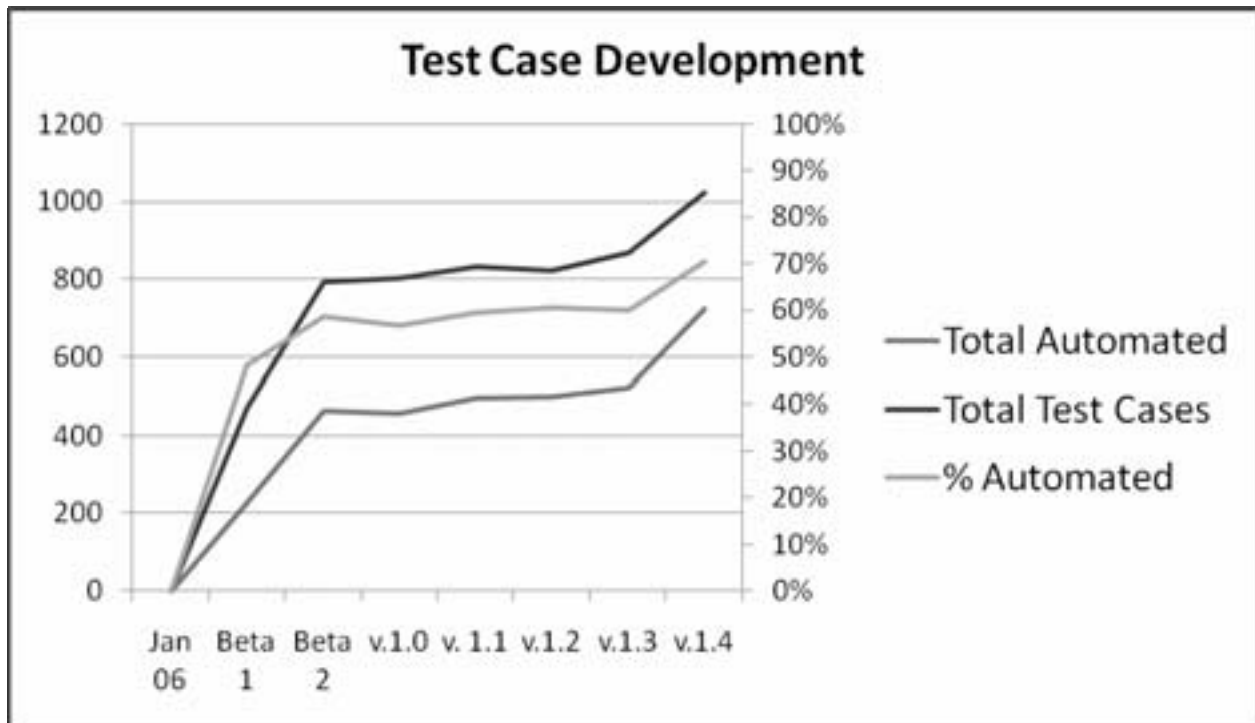


Bug Statistics



Note: RTW means 'Release to Web' (or, sometimes, 'Release to World')

Test Automation Statistics



Summary

Agile solves many of the problems associated with the Waterfall model. It incorporates requirements change as part of the model, and it leads to more dependable schedules and higher quality releases. There are many benefits to Agile in general.

But what about Agile offshore? Does it really make sense for a geographically distributed team to adopt Agile processes? We found that doing Agile in a distributed environment is difficult, but so is doing Waterfall in a distributed environment. Waterfall relies on detailed requirements and specs, but documentation is not generally a good method for communication. Once you solve the communication and relationship problems, working in an Agile fashion on a distributed team can be very successful.

I'll summarize what we learned using Agile with Scrum in a distributed development environment with the following four points:

- Have patience—expect several months of chaos before relationships are built and processes evolve to create a smoothly working team.
- Choose a flexible partner—your offshore counterpart should be willing to work the way you want to work.
- Plan for onsite rotation—set up a system whereby your team members work at the offshore facility and/or offshore team members work at your onsite location. Rotate team members out occasionally so everyone gets a chance to know the offshore team.

- Build relationships—the heart of a solid team is trust. Building a solid working relationship will provide a foundation for you to work through issues as they arise and maintain mutual respect for each other.

We made tremendous improvements in our processes and productivity over the course of a year-and-a-half. After our initial betas came in months late, we were finally able to reach a cadence of releasing updates to each service every two months, so effectively every month we had a release.

Releases became easier as we gained more experience doing them. Our partners came to believe us when we gave them a release date, and also to have confidence in the quality of our systems.

The offshore teams truly became an extension of core team. Due to a hiring freeze and turnover, our team ended up being 80% vendor/contractor resources, and only 20% full-time Microsoft resources. We were able to work very effectively with this model.

Our metrics continued to improve, and the VP of our group began pointing to our team as the ‘poster child’ for productivity.

Early on when we were struggling so hard to get a release out, we would work evenings and sometimes weekends. As we made improvements that increased our productivity, we found we could accomplish considerably more in less time. Once we reached the cadence of releasing every two months, we never needed to work extra hours. The team was happy and relaxed, and we all enjoyed working together in the Agile model. Best of all, we felt good about our contribution.

Endnotes

¹Jepsen, Ole “Agile Meets Offshore,” *Agile2006 Conference*, Minneapolis, MN, Handout, p.3.