Successful Integration of Agile Development Techniques within DISA

Sean Cohan
Pragmatics, Inc
cohans@pragmatics.com

Abstract

The Department of Defense (DoD) and the Defense Information Systems Agency (DISA) have historically operated on 18-36 months release cycles for major IT projects. DISA is now striving towards delivering smaller components in 30-60-90 day release cycles. This paper discusses Pragmatics’ successful implementation of agile development techniques into a non-agile shop, our introduction of agile development techniques to DISA to help them achieve their goal of delivering high quality software on shorter release cycles, and our lessons learned along the way.

1. Introduction

One of the most successful and important contracts for Pragmatics has been the Joint Operational Planning and Execution System (JOPES), a major DoD Command and Control (C2) program. DISA, the project sponsor for JOPES, is responsible for planning, engineering, acquiring, fielding, and supporting global net-centric solutions to serve the needs of the President, Vice President, the Secretary of Defense, and other DoD Components.

In early 2004, I was hired at Pragmatics as the lead engineer responsible for moving JOPES towards the DoD and DISA mandated Service Oriented Architecture (SOA). I knew how to establish J2EE web applications and web services that would meet the desired SOA. In my previous position, I had started using automated tests and automated builds. Realizing the benefit of being able to repeatedly prove that my software worked, I also knew I wanted to establish automated software testing and software builds; he was also well versed in agile development methodologies and practices. During the interview Mr. Gallaher said, “I can help establish the automated software testing and build environment”. As a result, Mr. Gallaher became our agile coach and began establishing the Pragmatics agile development environment.

2. The Agile Beginning

In early 2004, Pragmatics began introducing the agile development practices of test-driven development (TDD), continuous integration, collective code ownership, refactoring, iterative development, and collaboration with customers/end users. The initial agile team consisted of myself, the agile coach, one mid-level developer, and two junior developers. While the traditional non-agile developers, program managers, and customer focused on the current release, the agile team began developing the next release of applications designed to meet DISA’s objectives for migrating to an SOA consisting of n-tiered, thin client web applications and data exposed on the network via SOAP over HTTP web services.

The coach quickly established some basic infrastructure for the agile team: a Wiki for the knowledge repository (we’re currently using DokuWiki), an open-source source code repository (Subversion), and an open-source continuous integration tool (we’re currently using Vulcan).

The coach also introduced the concept of paired programming and open spaces. Initially, I was not convinced upper management would approve pair programming and an open area. I suspected upper management would believe a pair of programmers would only get half the work done that two individuals working separately could produce. I also suspected upper management wouldn’t approve the use of a large room for the open agile area since no rooms were available. The coach was not deterred. As a result, he was able to get upper management’s buy-in for pairing programmers together and the use of a small back corner of office space for the open agile area.
Corkboards, white boards, and pairing tables were brought in to equip the agile area, and the agile team was off and running.

From this beginning, the agile team grew to over 20 developers and the practices they put into place grew to become embraced by the corporation as strategic business practices, as well as by the customer as a means to increase the quality of software and shorten release cycles within DISA.

3. The Importance of the Coach

With the basic infrastructure in place, the team began reengineering one of the existing applications into a Struts, J2EE web application. The coach began teaching developers how to go about testing software using TDD, the use of story cards to document high level requirements, the use of task cards to document story tasks, how to estimate tasks and stories using points, how to hold daily stand up meetings, how to continuously integrate, the concepts of collective code ownership, and how to mercilessly refactor code.

Initially, developers struggled with how to write story cards and then break the stories down into tasks, how to estimate stories and tasks with the use of points (arbitrary values that give a general indication of the degree of difficulty for the story/task). With the help of the coach, most developers were able to grasp these concepts and put them to use.

We found that most developers new to TDD struggled with writing tests. The experience of the coach was essential here as he provided extensive hands-on instruction to individual developers on how and when to write tests. Most developers were able to pick up the basics and perform TDD within their pair. A common tendency for developers new to TDD was to write large amounts of code first and then write test code later. The coach played a key role here retraining developers to write the test code first and the payload code afterwards.

Another tendency for developers new to agile development was to work for extended periods on code without checking it into the source code repository. Here the coach reinforced the necessity for frequent check-ins and continuous integration.

A common tendency for paired developers was that one developer began to dominate the keyboard while the other watched. Another tendency was pairs did not rotate often enough. The coach would look for these tendencies and encourage (nudge) keyboard exchanges when it was not occurring frequently enough, and mix up the pairs to facilitate cross training.

Had we not had an experienced agile coach who could overcome these types of problems, we would not have been as successful establishing our agile development environment. His knowledge of what needed to be done, his mentorship over developers on the proper application of agile development techniques, and his care to ensure the agile development practices did not fall by the wayside was essential to our success.

As important as the coach was, he also attributes our success to having a few highly skilled and productive developers mingled in with the agile team. The coach could rely on these individuals to setup necessary infrastructure and properly apply agile development techniques. These developers led by example and established a high standard of quality for others to follow.

4. Improved Continuous Integration Tool

Early in the establishment of the agile environment, the agile coach implemented a continuous integration environment using an open source tool. The team successfully used this tool for a year, however, they became increasingly annoyed at some of its limitations (e.g., having to wait until all components were built before notification of the success or failure of any one component). As a result, Chris Eldredge, one of our most talented agile developers, took it upon himself to develop the Vulcan open source continuous integration framework. Not only did he correct the limitations of the previous product, he added the capability to report on Test Code Coverage (TCC) for all projects by plugging in the Emma open source TCC tool (we are now using the Cobertura product). These reports are part of the metrics that have helped Pragmatics achieve CMMI Level 4.

The developer also tailored the tool to generate build artifacts and software packages in a format required by our DISA customer. These modifications made it possible for our internal IV&V group to automatically generate software applications in the format required by DISA as well as test the installation, deinstallation, and ensure the software met all DISA delivery requirements. Now once our Pragmatics IV&V manager signs off on a software package, our Pragmatics CM manager delivers the automatically generated package to DISA.

5. Management Support and Time to Mature

When the agile team began establishing the agile environment, current non-agile programmers, their project managers, and our DISA customer were focused on completing and delivering applications that...
they had been working since 1998. Although not their primary focus, management supported the agile team by allowing them to establish an open area, allowed them to try pair programming, perform TDD and continuous integration, and provided them with requested materials (e.g., corkboards, whiteboards, pairing stations, etc.).

Within a short period of time, management saw the benefits of agile development as developers were able to demonstrate automated tests of their software, the continuous integration environment, and initial working versions of web applications that were on their way to meeting DISA’s objectives. Management began to recognize TDD and continuous integration as strategic discriminators against competition and began to market the agile environment as a means to provide potential customers with a greater assurance of receiving high quality software. Pragmatics also began to demonstrate agile practices to potential customers and included agile development methodologies in new proposals.

The supportive approach by management contributed to the successful establishment of our agile environment by giving the agile team the time they needed to not only establish the agile environment, but also truly learn the agile practices. The agile team was able to establish a development rhythm as well as develop self-sufficient agile teams. Management gave the agile team freedom to learn new tools and techniques for developing software, as well as encouragement to improve the agile development environment.

6. Support from our DISA Customer

Shortly after starting agile development, we brought over the JOPES DISA customer to demonstrate our agile practices: we demonstrated TDD by running the automated tests over the payload software; we demonstrated our continuous integration framework showing build and test reports; we explained our iteration boards and the use of story and task cards; and we explained our three-month internal release process even though DISA did not require software delivery for 12 months or more.

Our DISA customer was impressed with the processes we had put in place and encouraged us to continue. Over the next year, we hosted various members of the JOPES customer team for end-of-iteration demo days to further explain our agile practices. We also hosted other program offices within DISA for demonstrations of our agile practices.

7. DISA Moves Towards Agility

About a year into our agile development, a new Director took over at DISA. A primary objective for the new Director was to change the time it took to get a release of software into production. Historically, it has taken DoD and DISA 18–36 months to field systems. The Director, having met with Google representatives, was impressed with their 30-60-90 day release cycle and desired to implement similar processes within DISA. He favored fielding partial capabilities rather than building large, expensive, monolithic systems that took so long to develop and test that once they were fielded, they were outdated. Fielding partial capabilities would allow users to quickly provide feedback where “good ideas will take off and bad ideas will die off”. With the DISA Director calling for shorter, smaller, quicker releases, DISA departments would have to find ways field releases quicker.

During this time, we were demonstrating our agile practices to other entities within DISA including the Branch Chief of the DISA Test Engineering (TE) division. TE was interested in our automated testing capabilities and how they could be leveraged/integrated within the DISA TE group. We discussed how our automated tests could be handed over to DISA TE groups to run in their test environments. With the subsequent JOPES release, we did indeed pass our automated web services tests on to DISA testers. As web services technology was new to them and they were uncertain as to how to best test services, they were receptive to setting up our test frameworks within their test environment.

We also suggested that DISA TE become more involved with software projects while still in development rather than waiting until the software was delivered to DISA. As a result, the TE Branch Chief is taking active steps to integrate DISA testers with development projects. In May 2007, he presented “Wanted: Agile Testers” at the DISA Customer Conference where he discussed the need to integrate testers within development projects.

8. DISA Moves to Shorten the Release Cycle

Up to this point, once we delivered our software, various entities at DISA needed six to nine months before the software would be approved for production. Once test servers were built, the DISA TE group would test the software in what is known as a System Integration Test (SIT). Once the SIT was completed, another test group would perform interoperability tests in what is know as System Acceptance Test (SAT).
Once the SAT was complete, select end users would perform end user testing in what is known as the Operational Test (OT). Once the OT was complete, security engineers would perform security tests. Once all of these stages are passed, the software would receive its Approval to Operate (ATO) at which time it could be scheduled to move into production.

Both the DISA TE group and our JOPES DISA customer were looking for ways to shorten the length of time it took to move delivered software into production. One approach was to convince test groups to work in parallel during the test events rather than serially. Our JOPES DISA customer decided to leverage our three-month internal release cycle and have us begin delivering our web services on a three-month release cycle. In the spring of 2007, Pragmatics delivered its first three-month release. Our JOPES DISA customer’s confidence in our ability to deliver high quality, working software every three months is directly attributable to the implementation of our agile techniques and our internal three-month iterative release cycle.

With our current release, TE testers are working to become integrated with our Pragmatics testers earlier in the test phase. We are hosting our web applications and web services in our Pragmatics lab making them available for TE testers to test prior to deployment in the DISA test lab. In this way, TE can test our software prior to upcoming test events at DISA thereby shortening the time it takes to test at DISA. TE has commented that this has been a tremendous help to them, and they wish more contractors would make their software available in this way so it could be tested earlier.

We are also beginning to see frameworks set up at DISA that support automated testing and certification. DISA is establishing the Federation Development Certification Environment (FDCE) intended to be the ‘sandbox’ where parallel testing and certification events can occur. When deployed, software modules will be registered in the FDCE Portal, testers will be able to perform automated tests on them, certifiers will be able to certify that software is ready for production, and deployers will be able to automatically deploy the software into production. The FDCE will be DISA’s agile framework that enables 30-60-90 release cycles.

9. Successful Deliveries

In March 2006, we delivered to DISA our first applications developed using agile development techniques. These applications were successfully put into production in the fall of 2006. In March 2007, we delivered nine more applications using agile development techniques. These are scheduled for fielding in the fall of 2007. In June 2007, we delivered our first three-month release and are on track for delivering additional three-month releases in August and November of 2007.

10. Challenges

The introduction of agile development has been a success for both Pragmatics and our DISA customer. As a result, our DISA customer authorized more work be performed using our agile development practices. From a small team in the back corner of the office, we have grown to over 20 developers and expanded to virtually an entire floor. Although successful, we faced growing pains that are described below.

10.1 Lack of Onsite Customer/End User

Our end users are geographically distributed so we were unable to have any on site with us. Although we brought our DISA customer over as often as they could make it for end-of-iteration demo days, they were not end users nor could they provide the functional expertise of end users. Fortunately, a number of our testers are former end users and were able to act as pseudo end users. These Subject Matter Experts (SMEs) were able to provide valuable feedback to developers on what the end users wanted and needed.

An approach we took to overcome the lack of onsite end users was to make the web applications and web services we were developing available on the Internet. This allowed actual end user SMEs to evaluate the applications prior to putting them into production. On several instances the SMEs pointed out where we had missed requirements and had not built the right thing. Making the software available for actual end user feedback allowed us to fix missed requirements prior to delivering the software to our DISA customer.

10.2 Integration of Non-Agile Developers

We faced a variety of challenges integrating non-agile developers into the agile environment. Non-agile developers were briefed on several occasions as to what was happening with the agile teams and the practices we were following. Initial concerns for non-agile programmers included the noise level in the open area, not wanting people looking over their shoulder while they developed software, not wanting to lose status as senior engineers and team leads, not wanting applications they spent the last several years developing being replaced, not wanting to lose control
over developmental functionality, and doubt whether new web applications would adequately meet users' performance and responsiveness expectations.

In terms of noise in the open space, we've learned that expanded open areas help keep the noise to a comfortable level. Additionally, developers had to be reminded from time to time to lower their voices. Over time the team learned to stay within an acceptable noise level.

Some non-agile developers are still reluctant to move in to the agile area because they are either not comfortable pairing or they don’t want to lose their senior team lead positions. We have expressed to them that their domain knowledge is very valuable to the new agile developers and that they would be able to make significant contributions to the agile team. At the same time, they would update their current skills and learn new programming languages. As more and more of the legacy software is replaced by new reengineered solutions and more of their peers move to join the agile teams, these developers are coming closer the realization that they will have to give agile development a try.

It is understandable that non-agile developers were not pleased that their applications were being reengineered, especially since these applications hadn’t yet made it into production. However, they realized that DoD and DISA had mandated that applications become web-based and net-centric and that they must accept this fact.

Problems experienced with the first wave of non-agile developers that joined the agile team included confusion as to how to determine and document story cards, lack of confidence signing up for tasks, resistance to pairing, wanting to go off and develop code on their own, developing large amounts of code without writing any tests, and resenting other developers modifying the code that they had written. Here the coach mentored non-agile developers on what was expected of them and reinforced how to go about performing agile practices. Pairing up non-agile developers with agile developers usually alleviated this for most non-agile developers. Some never became accustomed to the agile environment however. For these few cases, the developers either asked to be transferred elsewhere within the company or left. Pragmatics. This was generally best for both the resistant developers as well as the agile team.

A common initial response from non-agile developers was they were somewhat intimidated by all that there was to learn in the agile environment, from new computer languages, frameworks, tools, IDEs, story and task cards, estimation, and TDD. Pairing most certainly helped them learn the technology, the agile practices, and become comfortable with the new environment.

10.3 Endless Iterations

Early on, the agile team was responsible for reengineering applications that would ultimately replace applications that had not yet been put into production. As such, the applications being developed by the agile team would not be delivered to the customer for a year for more. Initially, the agile team worked on an iterative development cycle consisting of two-week iterations. While the two-week iterations did provide good structure for dividing the work into smaller more manageable chunks, they lacked adequate milestones for the developers to realize a true sense on accomplishment. The iterations just keep going on and on.

In order to complete some of the tasks required for an actual release, the agile team put into place three-month internal releases consisting of six, two-week iterations. The three-month internal releases provided the opportunity to perform release planning and allowed the team to perform most of the steps required for a full release of a given application. This included providing all software needed for installing and deinstalling the applications as well as all application documentation required for delivery. The objective for the internal releases was for applications to be installable into production, albeit as a percentage of the completed to-be product functionality. This gave the team a greater sense of accomplishment and experience with all aspects of the application packaging and delivery.

During this time, the DISA Director was also looking to field software on shorter release cycles. Our three-month release cycles supported this objective by providing the process and model for our DISA customer to receive and field software on shorter release cycles.

10.4 Integration of IV&V Group

At the beginning of the move to agile development, our IV&V group (a team of six testers) was busy testing the current release of software developed using non-agile techniques. They had no automated test tools in place and only performed manual testing. Their testing practices consisted of developing test documents containing individual test cases, manually running the test cases, recording results, and creating problem reports. This testing tended to occur more towards the end rather than throughout the development effort. In addition, they were used to...
working from their offices and not as part of the
development team. As such, the agile team experienced
a number of difficulties trying to get the testers
involved earlier in the agile iterations.

Eventually, one of the six testers was assigned to
work with the agile team. This tester attended stand up
meetings and end-of-iteration demo days. Although we
couraged his participation writing functional tests, he
preferred to have the developers write the functional
tests and run them at end-of-iteration demo days while
he continued to perform manual testing.

We experienced problems trying to get the testers to
test on an iteration-by-iteration basis. The agile team
was developing working features with every iteration,
however, the testers were not testing the new features
after every iteration. Instead, they were testing non-
agile applications and hadn’t established a test cycle
that could keep pace with the agile iterations.

As more testers began testing the applications
developed by the agile team, we found testers would
test functionality outside the scope of the iteration and
write problem reports on areas developers had not yet
completed. They preferred to test the entire application
rather than only the features added in previous
iterations. We had difficulty constraining them to test
only features that were signed off as being complete at
the end of the iteration.

Things have begun to improve. The agile and IV&V
teams have gone through a large release where nine
applications were developed and delivered using agile
practices. We have held some reflection sessions where
we focused on things that worked well as well as areas
that could be improved upon. Some ways we are
improving are more testers are attending stand-up
meetings, they are attempting to sync up their test
efforts to test only the features developed during an
iteration, more testers are willing to write functional
tests using tools such as soapUI, and the test group is
now evaluating automated functional test tools to
regression test the GUIs of our applications.

11. Lessons Learned

The most important factor influencing the
successful establishment of agile development within
Pragmatics was the hiring of an experienced agile
coach. His real world experience with and knowledge
of agile development practices, his dedication to the
establishment of our agile development environment,
his commitment to train developers in the proper
application of agile practices, and his care to ensure the
agile practices continue was essential to the successful
implementation of agile development within
Pragmatics. Without the presence of an experienced
coach, agility most likely would not have been
successfully established within Pragmatics.

In addition to the coach, a key to our success was
the support provided by management to allow the
establishment of an agile team and giving them a
chance to prove themselves. Management’s support
allowed the team to apply agile practices, learn and
apply new tools, and evolve the agile environment. As
a result, the corporation now markets the improved
software development practices established by the
agile team as strategic business practices.

Lastly, DISA’s desire to establish shorter release
cycles along with Pragmatics’ TDD, continuous
integration, and three-month release cycles combined
to allow DISA and Pragmatics to broaden the
implementation of agile practices within DISA.

12. Acknowledgements

I would like to thank, Michael Gallaher, our agile
coach for providing the guidance and inspiration for
our agile development; Art Gomez, Vice President of
Pragmatics Software Engineering Division, for
sponsoring and supporting our agile development
efforts; Chris Eldredge, Software Engineer, for his
outstanding development skills evidenced by the
development of the Vulcan continuous integration
framework; and our agile development team for their
motivated dedication to the successful implementation
of our agile practices. Special thanks also to Tammie
Brawley, our DISA NECC Adaptive Planning Branch
Chief, for supporting our agile development efforts,
Dr. Steven Hutchinson, DISA TE Branch Chief, for his
work to shorten test cycles by integrating testers and
development teams, and Lt. General Croom, DISA
Director, for his work changing the DISA culture to
move towards an agile environment where shorter,
smaller releases containing smaller components are
preferred over the large, monolithic releases of the
past.