Agile in the Federal Government: Improving Technical Execution

ccpace

Abstract

As discussed in the first paper in this series Agile in the Federal Government: Scrum and Beyond[1], the history of Agile adoption in industry took the path of usage of eXtreme Programming (XP) followed by the combination of Scrum and XP. In the government, the adoption of Scrum alone has become the de facto method of Agile adoption with the use of the term Agile and Scrum even becoming synonymous in some agencies. In this paper we will discuss why we believe the use of Agile Engineering Practices such as XP is crucial to the success of Scrum projects for the Federal government and that the government can again benefit by combining Scrum with XP as is already in use by industry. We believe this is because Agile Engineering Practices enhance the empirical process control (Inspection, Adaptation, and Transparency) of Scrum.

Background

In December of 2010 the government introduced the **25 Point Implementation Plan to Reform Federal Information Technology Management**, now most often referred to as simply, the 25 Point Plan. The primary goal of the Plan is stated in the introduction;

"Government officials have been trying to adopt best practices for years – from the Raines Rules of the 1990s through the Clinger Cohen Act and the acquisition regulations that followed. But obstacles have always gotten in the way. This plan attempts to clear these obstacles, allowing agencies to leverage information technology to create a more efficient and effective government." [2]

The Office of Management and Budget, (OMB), has recognized the value of one of these "best practices", an incremental approach to software delivery, which it terms Modular Development in the 25 Point Plan:

"Modular development delivers functionality in shorter timeframes and has long been considered best practice in the private sector and in some areas of government; in fact, both Raines Rules and the Federal Acquisition Regulation (FAR) advise agencies to plan programs in this way. Successful organizations using modular development base releases on requirements they define at a high level and then refine through an iterative process, with extensive engagement and feedback from stakeholders." [1]

OMB subsequently provided a set of guidelines for Modular Development in the **Contracting Guidance to Support Modular Development** released in

June of 2012.[3]

Outside of the federal government, the core principles of this Modular Development approach are most often categorized under the umbrella term 'Agile'. The most popular Agile method being implemented is Scrum. Scrum defines the management processes and roles in a software development project.

In July of 2012, the GAO released a report on Software Development called **Effective Practices and Federal Challenges in Applying Agile Methods**.[4] In it they discuss the key challenges to applying Scrum within agencies and the practices which help Scrum succeed.

With these policies and guidelines, many agencies and departments have begun adopting Agile software development with the expectation of seeing higher quality software delivered sooner and at a lower cost. Meanwhile, other agencies and departments have retained use of traditional software development methods.

Most recently, the highly visible, flawed Healthcare.gov website roll-out has sparked much controversy; include a wave of dialogue about whether use of Agile practices could have helped to avoid some of the problems that were encountered. In particular, there appears to have been no use of Agile engineering practices. In the next sections we discuss the benefits of these practices when coupled with Scrum.

Initially Not Meeting Expectations

Many agencies have engaged in Agile software development via a top-down approach because their CIO or other high level management are following the guidance in the 25 Point Plan and OMB guidelines. They have identified programs in their agency to apply Agile within. They have sent some of their staff to a couple of days of training (compared to many years of non-formal training in more sequential development methodologies such as Waterfall). In addition, they've awarded the work to contractors with some form of Agile software development credentials. This may mean the contractor has some staff with a certified Scrum Master certification (again, a couple days of course work). With this model the first few projects produced limited success. Often many of the ideas of Scrum are not fully used and there is a large difference between what was discussed in their training and what is actually occurring during the projects. This is the realization that understanding Agile is easy; whereas, practicing it is difficult. We have seen

a similar realization arise in the private sector when it first began to adopt Agile software development. In this whitepaper we will discuss an approach to solve these difficulties that has succeeded in the private sector and which we believe will work effectively in the public sector as well.

Empirical Process Control

Three of the most important principles of Scrum are Inspection, Adaptation and Transparency. Referred to as the empirical process control model, these principles provide control for processes that are imperfectly defined and generate unpredictable and unrepeatable outputs. These principles are the key to the concept of always improving. Retrospectives and the repeated planning activities of Scrum offer numerous opportunities to inspect what is happening in order to keep improving the process. The team then makes changes (adapt) that will improve the process and outcomes. In order to do this, the process and outcomes must be made transparent to all involved.

Initially many of the improvements are made to external influences or involve personnel issues on the team. In order to keep improving their Scrum process, the team will eventually need to dive deeply into how the work is being executed. This is when issues of software quality, design quality, reliance on a single team member and other technical issues arise. All of these issues introduce risk to the project success.

The next sections address how to improve the technical delivery skills of the Scrum team and allow them to reduce the risk associated with the rapid changes that occur in an Agile project.

Agile Engineering Practices

How can an agency increase the chance of success with their Agile projects and become more Agile themselves? One approach is to bring in superstar, experienced Agile practitioners who know how to bring the academic lessons of Agile into the pragmatic practices of a project. Unfortunately there are only a limited number of these practitioners available. An alternate way to take your Agile projects to the next level is to introduce Agile Engineering Practices (AEP) from Extreme Programming (XP). While Scrum defines process management and roles, XP is an Agile framework for the development techniques. Table 1 has a brief description of the key practices. More in-depth descriptions are available in the second edition of Extreme Programming Explained by Kent Beck. Some practices may not be viable within every agency but not all of the practices must be adopted in order to gain value and mitigate risk.

Name	Description
Test-First Programming	Practice of writing a failing test, writing just enough code to make it pass.
Pair Programming	Practice of two programmers working at the same workstation. One writes code while the other reviews each line of code as it is entered.
Continuous Integration	Every commit to the code repository triggers an integrated build of the code and runs all tests against the build.
Code and Tests	Production code and test code are treated as first class artifacts. Test code is maintained with the same importance as production code.
Ten-Minute Build	Practice of keeping the automated build of the code below ten minutes. This includes running all tests.
Shared Code	All the team owns all the code. There is no section of the code that is the responsibility of an individual.
Whole Team	The team is not just developers. It includes a cross-functional mix of members who are required to complete the work.
Sit Together	Team is co-located in the same space with all barriers to collaboration removed.
Shrinking Teams	Practice of reducing team size by making one member of the team as idle as possible until he/she can be removed.
Real Customer	Practice of having the actual customer who will use the software be a part of the process.

As detailed below, Agile Engineering Practices enhance the empirical process control (Inspection, Adaptation, and Transparency) of Scrum.

Inspection

Inspection is a critical principle of Scrum. It is a response to the admission that all the requirements and the resultant plans cannot be fully known at the beginning of a complex software project. Thus we need to inspect our work and our process to be able to adapt as we go. Feedback is one of the primary methods of inspection.

Gain the earliest feedback

In Scrum we use short iterations of batched work called sprints so that we can receive feedback on that work immediately after it is completed versus waiting for a large User Acceptance Test (UAT) effort at the end of the project. In sequential waterfall development, each phase provides feedback to the previous phases. For example, the development phase may enlighten the team about a problem coming from the design phase. This may occur many weeks or months after the design phase is completed. The rework that occurs at this point can be very expensive as the cost curve increases over time. Thus it is highly desirable to receive feedback about work as early as possible. This is why all phases of software development (analysis, design, development, testing) happen during the sprint in Scrum. Several of the XP practices help move feedback up even earlier in the process. For example, the practice of **Continuous Integration** provides feedback on the quality of the code within 10 minutes or less. **Test-First Programming** provides feedback on the code within minutes. **Pair Programming** provides design feedback on the code within seconds. Having such early feedback reduces the cost to adapt even within the sprint and increases the chance of successful delivery of functionality within the sprint.

Increase feedback

Scrum has many feedback loops built into the different processes that the team uses. Sprint reviews and retrospectives are examples. In XP, there are engineering feedback loops that constantly allow the team to inspect and adapt as they work.

The **Continuous Integration** practice allows the **Whole Team** to see if the work they have integrated is fulfilling the requirements as reflected in the test results and checks run by the Continuous Integration tool.

The **Test-First Programming** practice provides feedback on the teams understanding of the requirements. They won't be able to write the tests if they don't fully understand the requirements. The automated tests then provide immediate feedback to the team demonstrating whether the code meets the requirements under test.

The **Pair Programming** practice provides instant feedback to an individual team member about their work. It is a simultaneous, rigorous code review session. Design ideas are critiqued along with the implementation of those designs.

Improving Adaptation

The feedback that comes from inspection is utilized to adapt the process, requirements, and code to better meet the goals for the software. In order to confidently adapt, the Scrum team has to have the confidence to make the necessary changes. Several of the XP practices help reduce the risk of these changes and bolster the confidence of the Scrum team.

Deliver high quality software not just on-time, on-budget

There are numerous agencies where Agile projects are delivering working software on-time and on-budget. In addition, thanks to Scrum, the highest priority functionality is being built first and the teams are improving their ability to adapt to change. However, at most agencies there is room for improvement in both the quality of the delivered software as well as the agility of the teams.

By using the **Test-First Programming** practice in a rigorous manner, teams can drastically reduce the number of defects found in the code. Test-First Programming is also known as test-driven development (TDD).Teams can also achieve much better software design via TDD. There is a greater adherence to design by contract with TDD and the coded functionality reflects the functionality specified in the user stories. Refactoring is an important part of TDD. TDD gives the Scrum team the confidence that improving the code via a refactoring is not breaking the functionality. (Just-in-time adaptation)

Improve estimates

XP practices such as Pair Programming, Shared Code, Whole Team, and Sit Together can improve the Scrum team's ability to make better estimates.

With **Pair Programming** the team will see many different methods of problem solving and technical approaches such that when they have a new story to estimate they will draw upon this knowledge to produce their numbers. It is important to correctly manage Pair Programming in order to maximize the value.

The **Shared Code practice** means that everyone on the Scrum team has knowledge of the complete code base. There are no silos in place that limit who can work on what part of the code. Therefore all user stories can be estimated by all members of the team. This improves the accuracy of the estimates as more input from different viewpoints helps pinpoint the LOE for the work.

The **Whole Team** practice in XP means that everyone who is needed to get the work done (based on the definition of done) will be involved during the estimation process. This avoids the situation in which a developer may be making an estimate on a story in which they have no real experience or domain knowledge of the content. By having the whole team involved, the accuracy of the estimates can be improved.

The **Sit Together** practice benefits estimation because the team is constantly hearing others talk and engaging them in discussions that don't just involve the stories that they are working on at the time. This increases the knowledge of the domain and technical system such that their estimation improves.

Increase collaboration with co-location

One of the key values of the Agile Manifesto[5] is the value of customer collaboration over contract negotiation. In Scrum we try to reduce all forms of uncertainty simultaneously. Uncertainty can be around the features of the final product, the design and technologies used to implement the product, or even who the customer or market is for the product. In Scrum we address that uncertainty via collaboration. Via the XP practice of **Sit Together** all of the team is co-located so that they collaborate in the most effective way to eliminate the uncertainties as soon as they need to (last responsible moment decision). Nothing enhances collaboration as well as communicating with all our senses.

Co-location is an advanced practice and can be impractical for some agencies where they have their staff spread amongst various locations geographically or even within the same building. The downside is that the communication process is much more difficult and effective collaboration will be limited. Another advanced practice of XP derived from Lean thinking is that of **Shrinking Teams practice**. With Shrinking Teams the idea is to make one person as idle a possible rather than easing the load on everybody. The ultimate outcome is to eventually reduce the team size. By using the Shrinking Teams practice, one can attempt to move toward co-location by reducing the work of those that are not co-located until you eventually shrink the team to just those members that are co-located.

Minimize risk

Scrum attempts to minimize risk by constantly adapting and inspecting as we build the software. XP is extremely risk adverse and addresses issues of quality in the code via such practices as Pair Programming, Test-First

Programming, **Real Customer Involvement**, Whole Team and others. Together they make a perfect pairing as Scrum allows the Product Owner to adjust priorities based on feedback during the project and XP allows the Scrum team to make the adaptations required to satisfy the Product Owner without undue technical risk to software quality.

Increasing Transparency

The key to inspection is transparency. All of the information needed to produce software must be available to the people involved in creating the software. By letting everyone concerned observe and understand what is happening, transparency enables more communication and creates trust in the process and amongst the team. It also allows decision makers to make project adjustments earlier on in the project, i.e., before the budget is exceeded and the deadlines are already missed.

Project transparency is greatly improved via Scrum. The Daily Scrum gives a view of progress on user stories. The backlog and burndown charts give a view into the progress on the project. Velocity charts show the team's speed. But individual team member's level of work and quality of work can be hard to ascertain. (Knowledge hording)

The **Pair Programming practice** brings tremendous transparency to the skill and contribution levels of the individual team members. Those that are highly skilled and giving high levels of effort will be quickly exposed. Those that are not skilled and/or not working hard will also quickly be exposed. The Scrum concept of succeeding or failing as a team doesn't mean that Pair Programming should be used as a way to expose incompetence and discipline those that don't meet the standards. Instead it can be a way to bring out the need for further training or coaching.

The **Continuous Integration practice** allows everyone to know where the quality of the product stands. The successful builds tell the whole team if the product is deliverable and meeting the definition of done. Failed builds let the team know if the product is going off course and needs to be corrected immediately. This transparent measure of quality is not achievable with any traditional method. In order to achieve Continuous Integration it is important to aim to implement the **Ten-Minute Build practice** as well.

The **Code and Tests practice** emphasizes the code and associated tests as primary artifacts of the project which will greatly increase transparency

because unlike all other artifacts, the tests are guaranteed to reflect what is actually developed. They are always up to date and accurate.

Summary

The Agile Engineering Practices enable a Scrum team to take their performance to the next level. Each of the practices mentioned in this paper will increase the value delivered to the customer via Scrum by increasing the team's agility and reducing the risk of rapid change. This is because the Agile Engineering Practices enhance the empirical process control (Inspection, Adaptation, and Transparency) of Scrum. We anticipate that the government will soon enhance their Scrum projects by combining them with XP as is common in industry.

While using all of the practices is the most effective way of achieving ultimate team performance, many Scrum teams are not ready to make the leap. The GAO makes this point when discussing an agency's strategic planning:

"Allow for a gradual migration to Agile appropriate to your readiness"[4]

Therefore it is advisable to undergo an assessment of the team's Agile maturity and determine which of the practices can be instituted immediately and which ones the team can aspire to achieving down the road. Again the GAO reiterates this point:

"*Combine Agile frameworks such as Scrum and XP if appropriate."*[4]

Future papers in the Agile in the Federal Government series will look at how agencies can scale Agile and how they can employ Agile for operations and maintenance. In both these endeavors, XP plays a critical role.

[1]http://www.ccpace.com/asset_files/Agile_Govt.pdf
[2]25 POINT IMPLEMENTATION PLAN TO REFORM FEDERAL INFORMATION TECHNOLOGY MANAGEMENT - VIVEK KUNDRA U.S. CHIEF INFORMATION OFFICER DECEMBER 9, 2010
[3]CONTRACTING GUIDANCE TO SUPPORT MODULAR DEVELOPMENT – OFFICE OF MANAGEMENT AND BUDGET, EXECUTIVE OFFICE OF THE PRESIDENT JUNE 14, 2012
[4]SOFTWARE DEVELOPMENT, EFFECTIVE PRACTICES AND FEDERAL CHALLENGES IN APPLYING AGILE METHODS – UNITED STATES GENERAL ACCOUNTABILITY OFFICE (GAO-12-681) JUNE 14, 2012
[5]http://agilemanifesto.org/