THE PROCESS GROUP POST

VOL. 19 NO. 1 HELPING YOU IMPROVE YOUR ENGINEERING PROCESS http://www.processgroup.com/newsletter.html October 2012

SCRUM - LESSONS FROM THE TRENCHES

NEIL POTTER AND MARY SAKRY

Introduction

Agile and Scrum are popular development approaches used in software development and IT. Scrum is a pre-defined development lifecycle based on Agile principles. Agile methodologies promote a project-management process that encourages frequent inspection and adaptation, and a leadership philosophy using teamwork, self-organization and accountability. [See sidebar on page 2.]

Scrum, by design, does not come with prescriptive details on how to address some of the challenges project teams face, or the challenges introduced when using Scrum. As with all techniques and processes, Scrum and Agile come with benefits and risks.

The benefits of Scrum and Agile include:

- Work is chunked into small increments to manage progress and technical risk.
- Functional requirements and technical issues can be discovered early in the project.
- Frequent progress updates are available using burndown charts.
- Test case generation helps find defects in code early.

The challenges that can be introduced by using Scrum, if care is not taken, include:

- Overzealous use of the Agile Manifesto.
- Relying on story point estimates prematurely.
- Embracing change with inadequate thought as to the consequences.
- Avoiding end-user involvement.
- Managing dependencies.
- Managing surprises and risk.
- Assessing the risks of refactoring and code ownership.
- Having a good enough backlog.
- Assuming that all teams can self-organize.

All of these items can easily be addressed.

This article looks at these challenges while working with different Scrum teams around the world.

Overzealous use of the Agile Manifesto

The wisdom contained within the Agile Manifesto (listed below) can be used to strike a balance between rushed code and sloppiness, and too much planning and process. It can also be used to rationalize a chaotic approach to software engineering and project management.

Agile Manifesto

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

These statements are supported by the "Twelve Principles of Agile Software" (agilemanifesto.org/principles.html).

The manifesto does not mean that items on the right

should be dropped altogether and replaced by items on the left. This would be an overzealous use of the manifesto. Teams that do this and live in the moment run the risk of being burned.

Even if the items on the right have historically been done poorly in the past, it is not a sound reason to abandon them altogether. For example, "*Working software over comprehensive documentation*," in its most extreme interpretation, means have no documentation. Communication of information to colleagues and customers can be very error-prone and time-consuming with no documentation. On the opposite extreme, having lots of documentation and no working software is not desirable either.

For the item, "*Responding to change over following a plan,*" having no plan and just reacting to change is little better than having an overly-detailed and rigid plan and not responding to any change.

When the manifesto is interpreted with balance, it helps a team make progress. Implementing either the left or the right in the extreme can add significant risk to a project.

A straightforward approach for interpreting the manifesto is to base it on the needs of your project. For example, if your project is 30 people spread over four locations, some planning is obviously required. If they are distributed across several continents, then some written communication will reduce errors and wasted time. If in the past, planning just meant filling out a project plan template with little benefit to the team, then project planning needs to be fixed rather than abandoned.

If documentation has become academic or a non-value-added tax to the project, and you want to communicate to other teams and stakeholders (without repeating the information verbally), then address the documentation challenge rather than swing the pendulum too far and rely on code to be an appropriate level of documentation. The software code might be the most accurate description of what the system does, but it is not a useful or efficient way to communicate to managers, end users and testing personnel.

Relying on story point estimates prematurely

Story points are used to estimate the amount of work to complete in a sprint (between 1 and 4 weeks of work). It is a relative scale and *can* include the difficulty or complexity of the work to be done. For example, if one user story (user requirement) has a story point value of 5 and another of 20, then the latter has approximately four

Summary of Scrum

Scrum is a process that teams can adopt quickly to plan and manage their work. Each Scrum step has just enough detail to plan, design, build and test code, while tracking team progress.

Scrum has three primary roles: Product Owner, Scrum Master, and team member.

The Product Owner communicates the vision of the product to the development team. This includes representing the customer's interests through requirements and prioritization.

The Scrum Master acts as a liaison between the Product Owner and the team. The Scrum Master does not manage the team but instead works to help the team achieve its Sprint goals by removing obstacles. The Scrum Master verifies that the Scrum process is used.

The team members do the project work. The team typically consists of software engineers, architects, analysts and testers.

The intent of Scrum is to build working components in small iterations, each iteration lasting between one and four weeks. A *typical* Scrum cycle has the following steps:

- Write the user stories (and store in the Backlog)
- Plan the release (which could span more than one 2-4 week Sprint)
- Plan the Sprint
- Conduct the Sprint
  - Analysis
  - Design
  - Coding
  - Testing
  - Demonstration

  } Team-defined tasks
- Conduct Sprint retrospective

Daily stand-up meetings are held to track team progress and identify barriers.

[See www.scrumalliance.org]

times the effort or complexity to complete.

When implementing story points, consider the following lessons:

- Velocity (story points completed per sprint) is a risky number to use to predict the duration of future work until sufficient data have been collected to show a stable value.

  For example, achieving a velocity of 200 points in the first sprint, 100 points in the second sprint, and 150 points in the third sprint do not provide a good reference point to predict the next sprint. It can be used to estimate a possible range of outcomes, but not a single outcome.

  Similar to size estimation performed by teams before scrum was popular, historical data needs to be collected, analyzed and organized before it can become useful and reliable. If the range of sprint velocity numbers is around 20 percent or less, then it becomes a more reliable basis for future deadline and scope prediction.

- Velocity numbers can be difficult to interpret for historical reference.

  Unless the team members capture information such as the assumptions the estimates were based on, technical difficulty encountered on the project and the actual tasks performed in the sprint, then historical velocity data can be inaccurate for future work.

  For example, in one sprint the hardware used by the team might have been full of problems and caused testing to take twice as long. In the next sprint, hardware might have been problem-free and caused no delays. If this information is captured along with the historical velocity of the team, then it can be useful reference data.

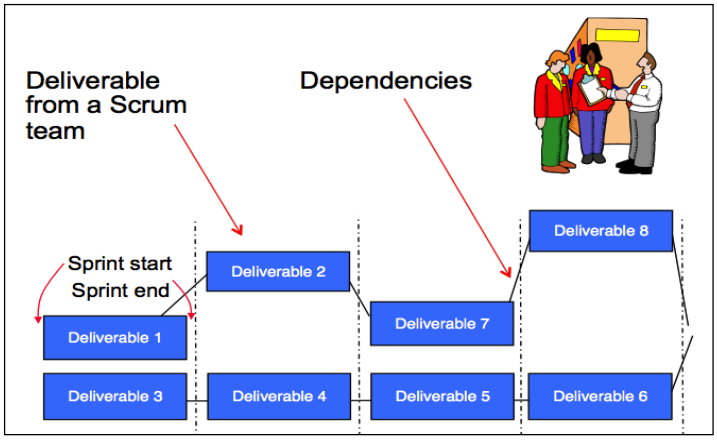


Figure 1

Embracing change with inadequate thought as to the consequences

One of the Agile themes is to, "Embrace change, even late in the project." Incorporating change based on new requirements, changes in customer needs, and errors in misunderstanding is important to make sure the product is successful. However, teams that just accept change because they feel "agile" face the risk of having nothing complete that can be usefully shipped. Accepting any change without adding it to the backlog and performing a backlog review is a sign of a sloppy Scrum implementation.

Not having end-user involvement

Agile has historically focused on close customer involvement and customer co-location. Where this is possible this is obviously desirable. However, many customers and end-users are many thousands of miles, countries and time zones away from the development group. This is addressed in Scrum by having a Product Owner be the intermediary between customers and the development team.

We have seen organizations take the easiest route to this and make the marketing or development manager the Product Owner and have no other end-user involvement. This avoids the original intent of obtaining real feedback from end users, which is the audience you really want to please.

Moving forward, create a group of real users (wherever they are located) with feedback and interaction coordinated by your existing Product Owner or marketing person.

Managing dependencies

When Scrum is used for self-contained projects that have few dependencies on other teams, then the nightly test builds can verify that technical dependencies (e.g., dependencies on code libraries and customer data) have not been broken.

When a Scrum team has dependencies with other teams, then some planning is needed to identify, communicate and coordinate the teams involved. The diagram (Fig. 1) shows a simple example of how dependencies can be captured over four sprints. Each box represents a sprint goal (deliverable) from each team, and each line represents a dependency.

Managing surprises and risk

Risk management consists of assessing, prioritizing, and mitigating potential problems (risks). Risks might be

related to customers, team skills, hardware, software or suppliers. The intent of risk management is to look downstream and determine what is worth paying attention to before it becomes a big problem. Where possible, the team takes action to make the risk less likely, or at least be prepared if the risk does happen.

In Scrum, risk management is mostly focused on technical risk by working on high-risk features in earlier sprints. This level of risk management is sound but can easily be expanded to cover more scope.

A little effort placed on assessing and mitigating risks can have a dramatic effect on the project. For example, assessing the risk of a supplier being late, or management not being committed to the release can lead to effective and early mitigation actions.

An example of a simple risk management process is listed at our article in Dr. Dobb's magazine (www.drdobbs.com/keep-your-project-on-track/184414727).

Assessing the risks of refactoring and code ownership

Refactoring is the practice of cleaning up code (or redesigning it) when it is believed the code could be cleaner or more efficient. The intent is to not modify the behavior or functionality of the code. While refactoring is encouraged, be wary that any modification to code (or any other document) can be the source of new unintended errors leading to wasted time. Change is not risk free.

Adopting the following actions can mitigate this risk:

- Code reviews by colleagues to check that errors have not been introduced.
- Nightly builds. These work well as long as there are test cases that cover functionality adequately. If your testing is not exhaustive (which it probably never will be), then risk should be assessed.
- Automatic comparisons of original and modified files that show what code actually changed.
- Version control to make it clear which version should be used, so that the team can go back to previous versions if necessary.

Having a good enough backlog

If Scrum and Agile are mistakenly used as reasons to rush into coding to "make progress," then there is a risk that the requirements backlog was rushed or skipped entirely. A phrase you might have heard is, "When we moved to Scrum, we threw out the requirements to speed things up." This would characterize a team ignoring the Scrum process.

One of the principles of Lean Thinking is to ensure that any source material (or information) is high-quality before it is fed into a workflow bottleneck. For example, feeding poorly defined, highly ambiguous, and conflicting requirements into a sprint can waste the time of the team in the sprint. Teams that start a development sprint with no backlog are likely to waste a lot of time (and money) very quickly.

The fix to this is not to revert back to a 6-month long requirements phase that you might have experienced in a previous life. Instead, allow time to review the backlog with the end user, product owner, and team members to remove any ambiguities they can, write test cases to identify ambiguity, and place high-risk requirements into earlier sprints to obtain early feedback. A well-groomed backlog can allow a team to be productive immediately in each sprint.

Assuming all teams can self-organize

Scrum teams are self-organizing, which means that the team is given a goal and left alone to determine how to achieve the work. Team members clarify the requirements, generate tasks, participate in sprint planning, volunteer for tasks to do, and decide with the product owner at the end of each sprint whether the work for that sprint is complete. When this works, it works well.

Not all teams possess self-organization skills and not all technical people want to perform the work of a scrum master (e.g., facilitate a team, track burn-down charts and manage team interference).

A good scrum master that is organized and has an outgoing personality will probably perform just fine. A scrum master that prefers to sit in a cube and develop code may try to avoid the scrum process since it involves planning, organization and team interaction. In this situation, other team members may naturally start to fill the vacuum left by the scrum master. If no one steps up to lead, the team will likely flounder and waste time.

Before you tell a team to self-organize or assign the role of scrum master, ensure that the scrum master has the skills and desire to perform the job. Train and mentor him or her, and after one or two weeks, look in and check that the team is actually self-organizing.

If you would like to learn more about Scrum, please see http://www.processgroup.com/services18asdcs.html

Practical Solutions for Your Current Challenges

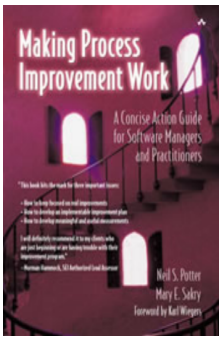
Webinar-style sessions to save on travel, or onsite coaching to save on time.

- ❒ **Run your software development projects faster and incrementally.**
  Two-day workshop, AGILE SOFTWARE DEVELOPMENT (SCRUM).
- ❒ **Understand how to save money, produce more and work faster.**
  Two-day workshop, DOING MORE FOR LESS.
- ❒ **Understand customer needs. Clarify product requirements early.**
  Two-day workshop, IN SEARCH OF EXCELLENT REQUIREMENTS.
- ❒ **Manage projects effectively. Meet project deadlines and reduce risks.**
  Three-day workshop, PROJECT PLANNING AND MANAGEMENT.
- ❒ **Meet project deadlines. Scope and estimate the project work.**
  One-day workshop, PROJECT ESTIMATION.
- ❒ **Avoid schedule delays caused by needless product rework. Find defects rapidly.**
  Two-day workshop, INSPECTION (PEER REVIEWS).
- ❒ **Hands-on SEI CMMI. Perform a CMMI gap-analysis.**
  The following workshops are available:
  - CMMI-DEV: Overview (1/2 day), LEVEL 2 (1 day), LEVEL 3 (2 days), Intro to CMMI-DEV (3 days).
  - Intro to CMMI-SVC (3 days), Supplement class (1 day), LEVEL 2 (1 day).
- ❒ **Identify critical changes to improve organizational results. Benchmark against the CMMI.**
  A PROCESS APPRAISAL examines your organization's current practices and generates a focused list of strengths and critical areas for improvement. Our SEI-certified Lead Appraisers conduct customized CMMI-based appraisals.
- ❒ **Clarify and refine business/project measures and analysis.**
  One-day workshop, MEASUREMENT AND ANALYSIS.
- ❒ **Systematically evaluate decision alternatives.**
  Half-day workshop, DECISION ANALYSIS AND RESOLUTION.
- ❒ **Goal/problem-based improvement for service and development organizations.**
  Two-day workshop, MAKING PROCESS IMPROVEMENT WORK.
- ❒ **Manage your suppliers.**
  One and one-half-day workshop, SUPPLIER MANAGEMENT.
- ❒ **Achieve more with your time. Make your staff more productive.**
  One-day workshop, TIME MANAGEMENT.
- ❒ **Tailored assistance. Dedicated phone/web-based assistance.**
  This service consists of customized education and coaching on your specific problems (e.g., meeting deadlines, quality and cultural change).

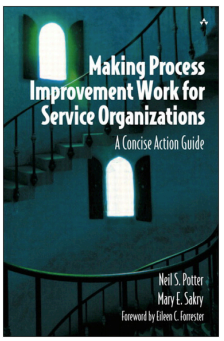
*Detailed information is available at **www.processgroup.com/services.html**. Contact us at **972-418-9541** or **help@processgroup.com** to discuss your needs.*

Read Our Books!

http://www.processgroup.com/book.html



Development audience



Services audience

Chapter 1. Developing a Plan
- Scope the Improvement
- Develop an Action Plan
- Determine Risks and Plan to Mitigate
- Chapter Summary

Chapter 2. Implementing the Plan
- Sell Solutions Based on Need
- Work with the Willing and Needy First
- Keep Focused on the Goals and Problems
- Align the Behaviors of Managers and Practitioners
- Chapter Summary

Chapter 3. Checking Progress
- Are We Making Progress on the Goals?
- Are We Making Progress on our Improvement Plan?
- Are We Making Progress on the Improvement Framework?
- What Lessons Have We Learned So Far?
- Chapter Summary

The Process Group

Telephone: 972-418-9541

Fax: 866-526-4645

E-mail: help@processgroup.com

Web: www.processgroup.com

POST back issues are on line