Goodbye, Scope Creep — Hello, Agile!

Michele Sliger, President, Sliger Consulting, Inc.

Abstract

"Scope creep" is a phrase used to describe the expansive change in requirements that begins some time after requirements are baselined at the beginning of the project and continues throughout the life of the project. The goal of most plan-driven projects is to lock in these requirements at the outset of the project and minimize change as the project team works to deliver what was originally defined. Therefore, scope creep is considered an unwelcome phenomenon that causes delivery delays and cost overages. Agile approaches offer an effective alternative to this problem; its value-driven approach allows for change and drives delivery by focusing on the most important features, first using rolling wave planning and progressive elaboration. Because agile frameworks are designed to welcome and manage change, scope does not "creep," because change is expected and accepted throughout the life of the project.

Scope Creep

If you've ever worked on a software development project, then you are already familiar with the phrase "scope creep." This is a term used to describe the changes to the scope of a project that occur after the project scope has been baselined and/or approved. Its typical manifestation is the addition of new product features without a corresponding increase in budget, time, or staffing. In other words, the project team is expected to do more work with the same amount of resources and deliver in the original timeframe. Other expressions of scope creep include the allowance of time extensions to complete the additional work and additional funding to pay for more staffing. However, all instances of scope creep are considered "bad form" because they stray from the original plan and, therefore, something to be strictly avoided.

Assuming Perfect Knowledge

Why straying from the original plan is considered bad form is another matter! The only way to prevent changes to an agreed-upon feature set is to spend a lot of time analyzing and defining that set and finalize the set *assuming perfect knowledge*. We must know not only what the product is expected to do, but also what is implied but not discussed, and we must psychically predetermine the actions and reactions of the users to the new product and the actions and reactions of its marketplace competition. And, let's not forget technological surprises in the software industry and geographical, weather, and material surprises in the construction industry. In other words, we need to know what we know, know what we don't know, and know what we don't know we don't know.

Acquiring perfect knowledge is impossible; yet, this is the way many project teams still work. Why? Because it is what they were taught to do! Today's teams were taught this planning method because it was what worked for their predecessors for decades—it worked for them then, but it isn't working for us now! So, today's project managers ask themselves what they and the team are doing wrong instead of considering the notion that it is the method itself that is no longer working.

The Speed of Change

Fifty years ago, planning out the scope of an entire project and locking it in worked because the pace of change was much slower. Teams had time to analyze, design, code, test, and deploy the product *before their customer could change their mind*. It wasn't until the late 1980s that all of the inventions that speed thoughts of change started being used by the general marketplace: the personal computer (PC), Internet, e-mail, and cell phones. The speed of communication increased, which increased the speed of knowledge, which drove the speed of change. Teams using the traditional plan-driven approach to product development could not keep pace with this speed of change.

There was also a corresponding increase in the ability of teams to deliver new products to the market as a result of technological changes. CAD/CAM systems, object-oriented programming languages, desktop compilers and debuggers, and reverse engineering applications allowed engineers to design and build products faster. One assumes this meant that the delivery of value was able to keep pace with the speed of change, but this has not typically been the case. The impediments to fast delivery of value to the marketplace were the old philosophy that change was bad and the brittleness of the traditional approach to scope definition and planning.

A new approach to product delivery would be required in order to take advantage of the technological advances and speed value to market. It would have to abolish the philosophy that change was bad and instead embrace it and it would have to grant teams flexibility in responding to change that would allow teams to deliver value efficiently and effectively.

An Agile Approach

In 2001, a group of like-minded individuals, each involved in software development projects, gathered at the Snowbird Ski Resort in Utah, USA. These people had been practicing new ways of developing software that allowed them to make changes quickly and easily. The industry media had coined the term "lightweight" software development to refer to their approaches because they were light on documentation and formal process. The individuals who were saddled with this term, however, were not happy with its connotations. The intent of the gathering was to come up with a new name that better described their philosophy and approach to product development. Over the two days they spent together, the group defined not only a new name—agile—but also a manifesto, set of guiding principles, and an alliance. The Agile Alliance included the representatives of Scrum, eXtreme Programming (XP), Crystal, Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), as well as several others. (*Manifesto for Agile Software Development*, 2001)

Flipping the Triangle

The Agile Alliance believes that its focus should be on delivering value early, frequently, and continuously. (Although the Agile Alliance was founded in the software development industry, there are broad agile approaches, like scrum, that extend the delivery of value definition to include non-software products as well.) To do this, agile advocates the ongoing daily involvement of the customer or customer representative with the delivery team, as well as frequent deliveries of product increments ("from a couple of weeks to a couple of months" (*Manifesto for Agile Software Development*, 2001)). These philosophical changes result in the flipping of the traditional triangle on its head, as shown in Exhibit 1.

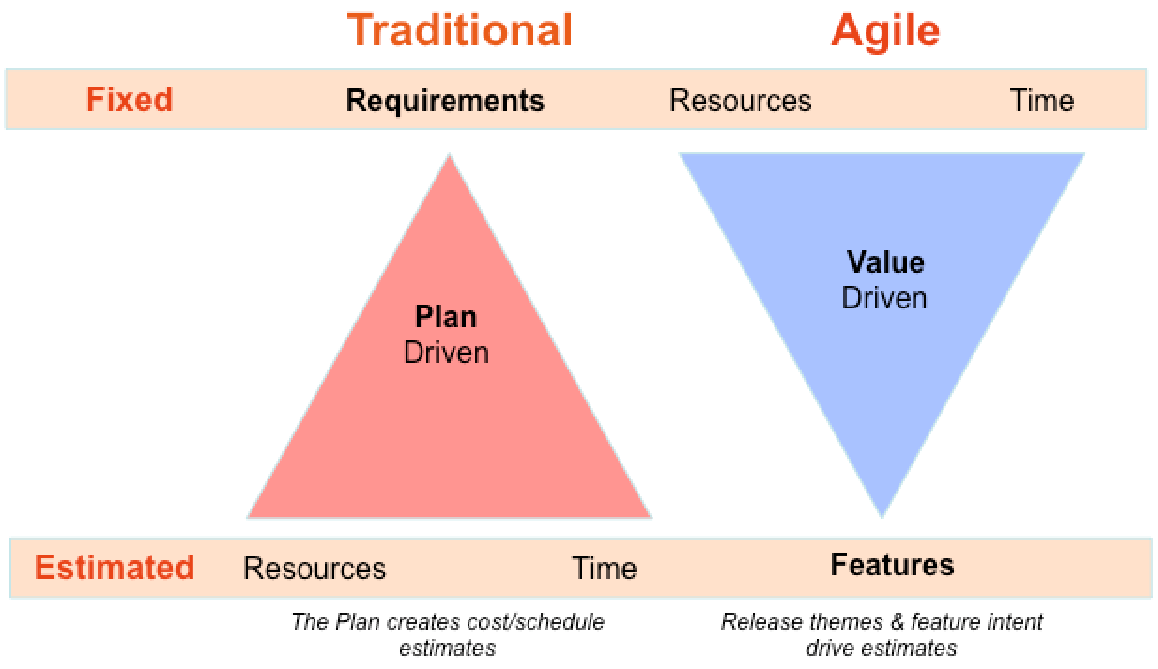

Exhibit 1 – Flipping the Triangle (DSDM Consortium).

In the traditional method of defining scope, a bottom-up approach was used that defined scope in a very granular detail, at the very beginning of the project, before any development work was started. The resulting definition of the detailed requirements was then set or “fixed,” and project managers estimated resources and time based on the collection of fixed requirements. This is considered a plan-driven approach, where sticking to the plan is considered the key to success.

Agile flipped the triangle on its head. Instead of fixing the requirements, they chose, instead, to fix things that were more well-known and less likely to change: resources and time. Features that would make it into production were then estimated based on these constraints. Because it is possible for either time or money to run out before all features are completed, agilists had to focus on completing the most important features first. Thus, rather than being plan-driven, agile is a value-driven approach to product development.

Using a Top-Down Approach

Agile uses a top-down approach to defining and estimating scope in order to eliminate the waste (time and money) of spending many weeks defining requirement details that may never be implemented or that may outdated by the time the team is ready to build the feature. Following just-in-time inventory ideals, agilists think of detailed requirements as inventory that must be stored and maintained until it is time to use them. In order to prevent the incurring costs associated with holding on to inventory that isn’t yet needed, the use of rolling wave planning and progressive elaboration contribute to the top-down method of scope definition and planning.

Agile projects begin with a clear vision and a backlog of high-level, course-grained features. The most important feature is then broken down into smaller pieces, sometimes called minimally marketable features (MMFs), which are sized so that each can be independently developed in a short timeframe. A group of MMFs can then be bundled into a minimally marketable release (MMR). Releases are planned using the course-grained features and gross-level estimations, then as the team prepares to work on a particular MMF or group of MMFs, planning in detail is done, complete with tasks estimated in hours and/or days. The detailed estimation and planning are only done for work the team is going to do in the next short timeframe. Remember our Agile Manifesto principle of frequent incremental product delivery “from a couple of weeks to a couple of months.” (*Manifesto for Agile Software Development*, 2001)

Agile and *PMBOK*$^{®}$*Guide*

Some concern has been raised that, in order to adopt agile development practices, one must throw out the knowledge outlined in the *PMBOK*$^{®}$ *Guide.* Unfortunately, misconceptions such as these still exist, particularly regarding the types of methodologies that can be used in conjunction with the guidance found in the *PMBOK*$^{®}$ *Guide*. The Project Management Institute (PMI$^{®}$) has no problem with the use of an agile approach; in fact, rolling wave planning and progressive elaboration are both acknowledged practices in the *PMBOK*$^{®}$ *Guide.*

In addition, the *PMBOK*$^{®}$ *Guide* also outlines project life cycle phases that correspond nicely to agile releases and/or iterations. An agile project can be made up of multiple releases or calendar quarters, which in turn can be made up of iterations. The initial phase in an agile project includes planning processes and usually a delivery of an increment of code; the final phase includes hardening and production readiness processes as well as a final project retrospective. All intermediate phases focus on the delivery of value in the forms of product increments.

Hopefully, concerns about the use of both agile approaches and the *PMBOK*$^{®}$ *Guide* will become a non-issue in the coming years as project managers learn more about both methodologies.

Being Agile

The agile concepts presented so far are broad and represent the highest level of interpretation of the Agile Manifesto. What follows are three specific agile methodologies, or frameworks, and how each embraces and manages changing scope throughout the life of the project.

Scrum

Scrum was created by Jeff Sutherland, Ken Schwaber, and Mike Beedle and they based their approach on a paper written in 1986 by Hirotaka Takeuchi and Ikujiro Nonaka, titled *The New New Product Development Game*. Scrum is a rugby term and refers to the team circle that occurs when there is a need to get the ball back into play again. Seeing teams plagued by analysis paralysis, Schwaber felt the word “scrum” was an appropriate term to indicate the need to get them unstuck and moving forward again.

Scrum manages scope change through the use of a product backlog of ranked features (by importance to the customer) and timeboxed deliveries called “sprints.” The product backlog is owned by the customer or their representative and they have the freedom to reprioritize the features as they see fit. The customer provides the highest ranked features to the team at the beginning of each sprint, along with any details needed to implement the features. The team estimates the work, commits to only what it believes it can get done in the sprint, and then begins the work. During the sprint, no changes are allowed. However, while the team is hard at work preparing the features for delivery, the customer can pick out the next set of features and prep each with enough detail for the team to begin working on them in the next sprint. Thus, change is allowed and discussed at the beginning of every sprint.

eXtreme Programming (XP)

XP was created by Kent Beck, whose first book on the subject, *eXtreme Programming eXplained*, included in its title the phrase that established the philosophy of the agile movement: e*mbrace change*. XP is a set of technical practices for software developers that turn the dial up to 11 (that’s the “extreme” part) on practices that are generally considered to be good things to do. For example, if peer reviews of code are good, then XP turns the dial up to 11 by doing paired programming (i.e., having peer review done as one is coding0. This becomes a collaborative exercise, one in which both programmers can learn from each other, while they assist one another in the creation of high-quality efficient codes.

XP’s method for managing change is similar to that of scrum. Customers provide the team with “stories,” which is a specific technique for writing a requirement from the user’s point of view, during planning game meetings, which are held at the beginning of each iteration and release. Customers are then expected to stay with the team during the iteration, sitting with the preferably co-located group, so they can answer any questions that might arise and assist with further detailed requirement definitions during story implementation in the iteration.

Kanban

The Kanban method was created by David Anderson and is a mechanism of the Toyota Production System. Instead of delivering in iterations, Kanban provides for a flow of frequent deliveries as part of a pull system. Pull systems allow workers to pull work into the system based on their capacity to handle it, rather than pushing work based on demand. A Kanban board is used to help the delivery team visualize the flow, limit work in process, and prompt problem solving when bottlenecks occur.

Kanban manages scope change by providing customers with an input queue where they can place their feature requests. As with all queues in Kanban, there is a limit on the number of items that are allowed in the queue. Customers are asked to simply replace empty slots in the queue and not required to keep or maintain a prioritized backlog. They must simply ask themselves which two new items they would like next. And, if service level agreements, average lead times, and due-date performance numbers are known, the question might more correctly become "which two new items would you like delivered thirty days from now?" (Anderson, 2010, p 105)

Conclusion

The only reason to use a traditional bottom-up approach to defining scope for a project is if it is known at the outset that the requirements will not change. If scope is truly fixed, then there is no need to use the change management models that make up the agile suite of methodologies.

However, if change is expected, or if there is a history of change on similar projects, then say goodbye to scope creep and say hello to agile! Take advantage of an agile framework and allow it to assist you and your team in the early, frequent, and continuous deliveries of valued product increments.

PMI does not advocate any particular methodology; it only supplies a standard of good project management practices, and whether individuals choose to follow a traditional or an agile approach, the *PMBOK® Guide* will support them both. You don't have to cast aside your PMP designation to be agile: All you need to do is change what you think of change!

References

Anderson, D. J. (2010). *Kanban: successful evolutionary change for your technology business*. Sequim, WA: Blue Hole Press.

Beck, K. (2000). *eXtreme programming eXplained: embrace change*. Reading, Massachusetts, Addison-Wesley

DSDM *Atern: The Agile Project Delivery Framework*. (2009). Retrieved February 19, 2008 from http://www.dsdm.com/.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler M. et al. (2001). *Manifesto for Agile Software Development* Retrieved July 18, 2010 from http://www.agilemanifesto.org/.

Project Management Institute (PMI) (2008*). A guide to the project management body of knowledge (PMBOK*® *Guide)*—Fourth Edition. Newtown Square, PA: Project Management Institute.

Schwaber, K., & Beedle, M. (2001). *Agile software development using scrum*. Upper Saddle River, NJ: Prentice Hall.

Takeuchi, H,. & Nonaka, I. (1986, January-February). The New New Product Development Game. *Harvard Business Review*, [Reprint 86116].