Agile Development Methodology

Introduction:

The Agile development movement began in earnest in the 1990s as a rejection of the establishment with its rather staid and seemingly sluggish development methods known generally by names such as the waterfall model or V-model. These older methods had gained a reputation for **missing deadlines, going over budget or failing completely** and Agile offered a means to make this a thing of the past. With most revolutions the ideas are rarely new and often promise more than they can actually deliver, and so it has proven with Agile methods. While there are considerable benefits to Agile methods, they are by no means a panacea for past ills and should only be adopted after serious consideration and careful planning. It is critical that the strengths and weaknesses of any method are understood before proceeding, and risks reduced through awareness and preparedness for potential pitfalls. This article aims to provide a quick starting point for understanding the most prominent methods, their individual characteristics and unique offerings.

All Agile methodologies **share a set of core ideas** which are prescribed from method to method in subtly different ways; iterative and incremental delivery of working code, frequent collaboration with stakeholders, closely working, self-organizing teams, and the ability to embrace change late in the project. Agile methods are shamelessly incestuous, borrowing from each other and using existing ideas in slightly different ways so it becomes very difficult to tell whether a project is following any one method as even the slightest adaptation of any aspect of a process can make it seem more like another.

Fortunately, the majority of the concepts are compatible so that it becomes very easy to use Agile methods like a smorgasbord, picking and choosing parts at will.

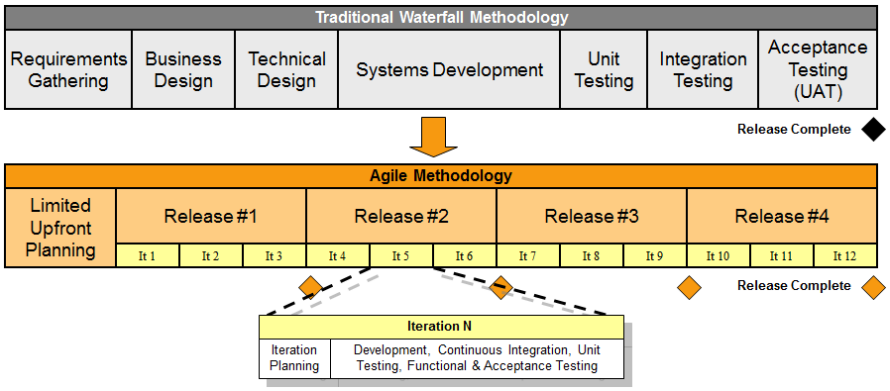


Figure 1: The iterative approach over the traditional approach

Some Agile proponents are so enthusiastic that they fail to recognize that Agile methods have drawbacks. They are not particularly adaptable to larger, enterprise or distributed developments where teams cannot all meet face-to-face and they are less well suited to fixed-price, contractual projects in which functionality is non-negotiable. They are also difficult to apply to embedded systems. Some Agile methods have little up-front design effort which can lead to considerable rework or integration problems. User involvement is also a double edged sword when frequently changing ideas lead to more requirements churn than even Agile processes are prepared for.

The Agile Manifesto (www.agilemanifesto.org) provides a set of guidelines for those wishing to become more agile but it is not a set of rules and is does not in of itself define process. This leaves plenty of room for debate as to which methods are Agile and which are not. In this article we shall address those that are generally accepted as Agile followed by those for which argument continues. It should be stressed that while some methods describes themselves as Agile, or include 'Agile' in their name, that does not necessarily make them so.

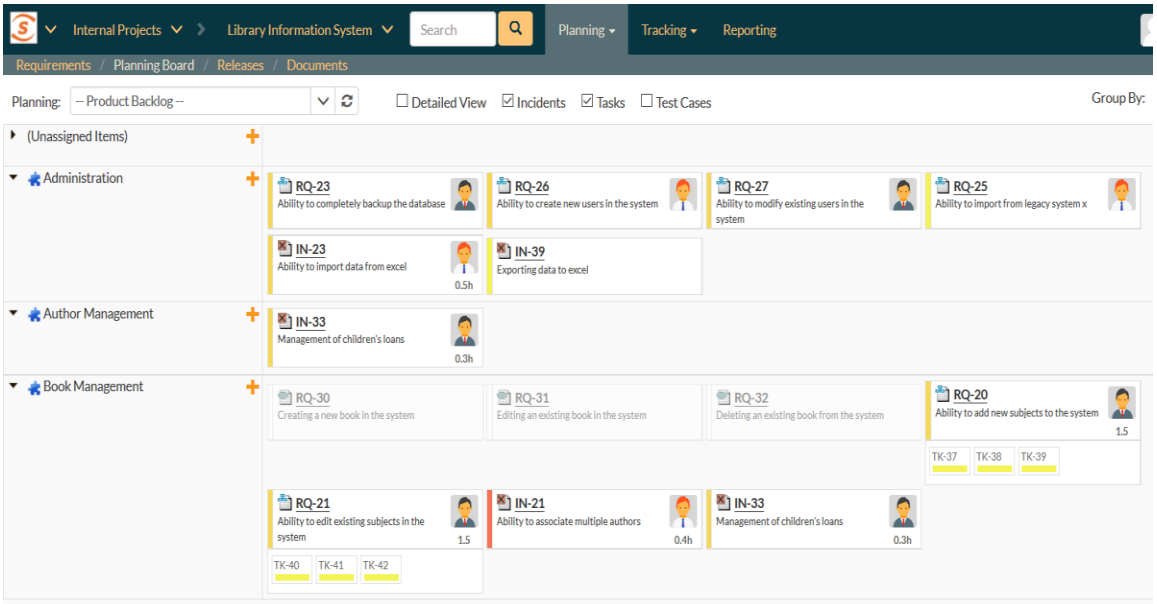
Section 1. Yes, it is Agile!

Scrum:

With prominent web sites such as www.scrum.org and www.scrumalliance.org to promote and support the Scrum method and those adopting it, Scrum has become one of the 'go-to' techniques for those striving to become Agile. Along with XP, it is perhaps the most well-known of the Agile methods, and also the most precisely defined which means that there is a lot of documentation and pre-built process for teams that are willing to adopt the methodology completely.

However, Scrum's greatest strength is also its greatest weakness. With specifically described roles, short iterations, tight schedules, daily meetings and an insistence on a release-quality product after each iteration (or sprint in Scrum terms), Scrum can seem so very foreign to inexperienced teams that it can be difficult to achieve early success unless you have some in-house knowledge.

For those who are old-hands at the method however, it can prove quite effective. As with Agile methods generally, requirements begin as a simple prioritized list of needs with little detail, known as the backlog. Only enough of these needs are pulled from the backlog to fill a single sprint and to begin their short journey from refinement to tested software. A typical product backlog is shown below:



On their way, these needs will be elaborated by stakeholders, discussed in daily meetings (scrums), in which they are subjected to design proposals, used to create test cases, implemented, tested and delivered to stakeholders to review and provide input for subsequent sprints - and all this in just 2 to 4 weeks. Progress throughout the sprint is measured by the number of story points left to complete in that sprint and displayed using a 'burndown chart'. A story point is an arbitrary measure of the complexity or effort required to implement a user story. The only requirement is that story points are consistently applied. The rate at which story points are completed is called the velocity. After each sprint there is a review and weaknesses in the process or team performance are identified and changes made. In this way the project should improve from one sprint to the next.

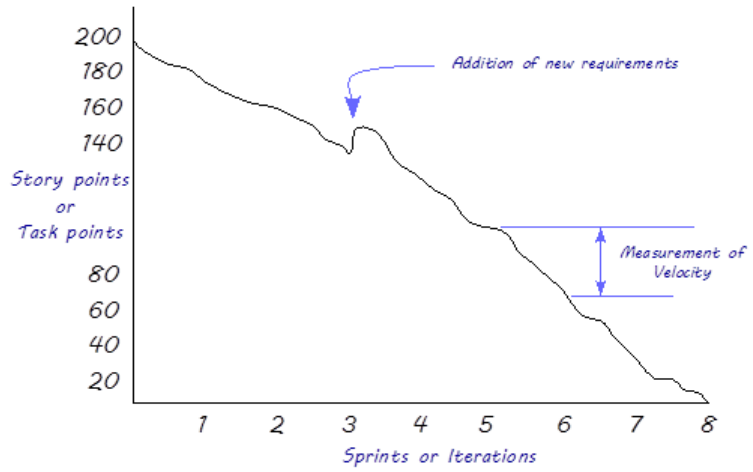
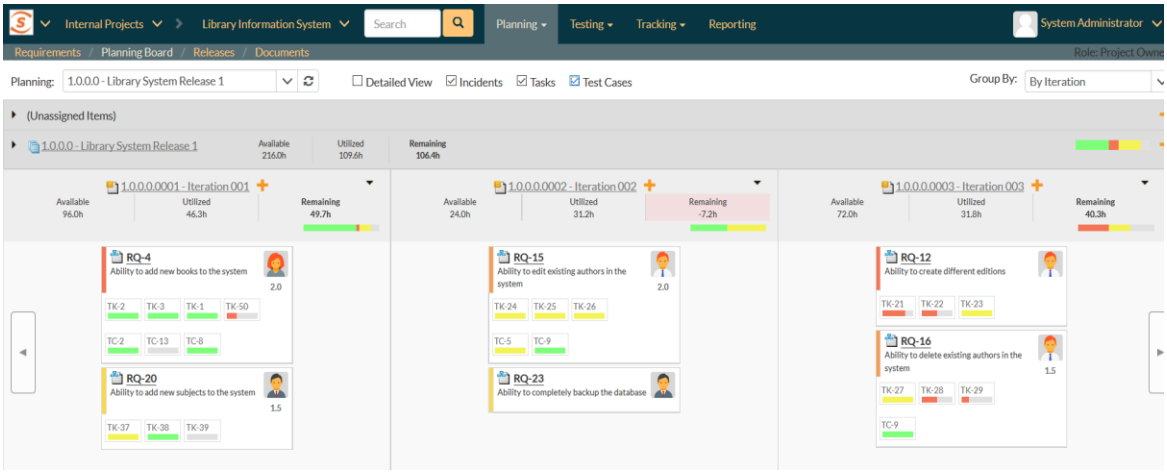


Figure 2: Typical burndown chart

The three most prominent roles in Scrum are the ScrumMaster, the Product Owner and the team member. The ScrumMaster acts as a super team leader, (but not a team member) keeping the team focused on the goals of the sprint, ensuring Agile principles continue to be followed, working with the Product Owner to keep the backlog up to date, and resolving any issues which might put the schedule at risk.

The Product Owner decides how to prioritize the requirements, making changes as necessary based on feedback from stakeholders. Note that the Product Owner does not choose which requirements go into a sprint exactly; that is dictated by the amount of work that can be done in the sprint and the backlog order. The Product Owner can only prioritize, not say how many items from the top of the list will be addressed next. A typical Sprint backlog is shown below:



Team members are those that design, code, test and produce the product. The team should consist of cross-functional individuals able turn their hands to whatever is required to meet the sprint goals, as well as at least one representative of the stakeholders. Completing the sprint on time and with all goals met is the responsibility of the team as a whole, it is not the job of the ScrumMaster!

While Scrum does not describe the whole process, what it does define it does so very specifically. To fill the gaps, most projects pull in ideas from other processes resulting in Agile projects that use a hybrid of more than one concept.

Extreme Programming (XP)

The idea developed by Kent Beck was to use best programming practices but take them to the extreme – hence the name. As such, none of the individual concepts of XP are really new, but their application and combination is what makes XP different, or at least did in the 2000s, which is when it initially became popular. XP advocates adaptability; recognizing that each project faces different problems, applying those methods that will work and discarding those that will not. Partial XP therefore becomes a true possibility, often combined with Scrum, which has gaps in its definition particularly suited to adoption of XP practices.

XP embraces standard Agile principles such as short iterations; requirements being loosely defined with user stories until they are needed; almost immediate and regular testing; close, on-site stakeholder involvement with a rapid feedback loop; and team responsibility for success or failure.

As is necessary for Agile projects, the XP team expects and accepts changes in requirements based on feedback from stakeholders and the obligatory high level of communication within the team. Intense communication is necessary because everyone is required to understand the system as a whole so that anyone can work on and change any of the code. This fact alone makes XP difficult (but not impossible) to apply to large projects. Quickly written code is often used as a means to explain ideas and try out designs. With short iterations comes regular layering of new functionality, thus refactoring is encouraged in order to consolidate separately written parts into a more efficient whole.

Testing taken to the extreme means using as many testing techniques as necessary, as often as possible. Methods include unit testing, acceptance testing, integration testing and test-first programming, (although this is debatably a coding technique not a testing technique.) Initially, daily integration testing was advocated, however this has sometimes proven impractical and the frequency practiced is often longer, perhaps weekly; a good example of taking from XP only that which works for a particular project.

Pair programming is part of XP, the idea being that two heads are better than one. Using only one workstation, one person writes code while the other observes, injecting ideas, pointing out potential problems and thinking about the ‘big picture’ as the code is written. Roles are reversed frequently and the pair work as if joined at the hip. If a pair get too comfortable with one another they may begin to think alike and two heads thinking as one is no better than one head, defeating the purpose. Pairs should be split up and reassigned when this happens. While pair programming does increase staffing needs, it is an excellent technique for building in quality which is an Agile ideal.

Finally: simplicity. The simplest solutions are encouraged, addressing the immediate problems, not problems that might arise tomorrow. This helps everyone to remain focused and meet deadlines but can result in considerable redesign and refactoring over time.

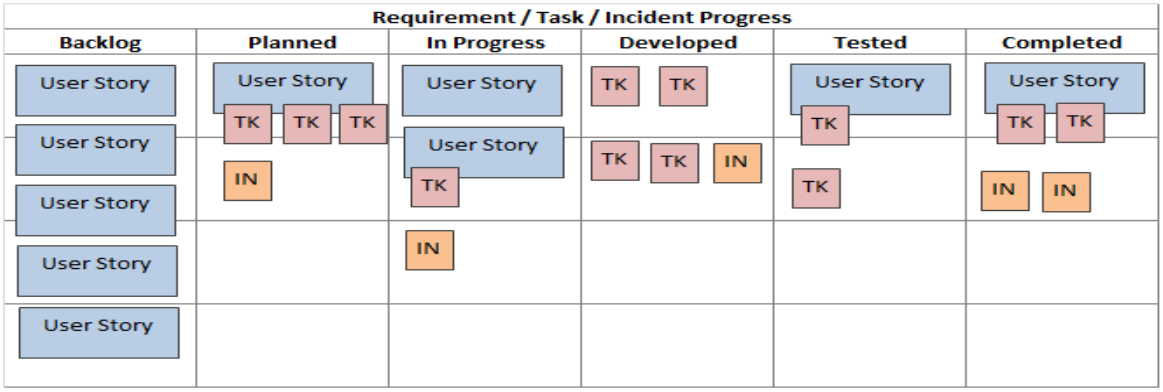
One of the reasons XP is so popular is its flexible nature. It is easy to combine features of XP with other ideas; indeed, XP is more about technique than process and so dovetails well with process-centric approaches such as that of Scrum.

Agile Development Methodology

Kanban

Kanban has the dubious distinction of being Agile without being iterative. Processes like Scrum have short iterations which mimic a project lifecycle on a small scale, having a distinct beginning and end for each iteration. Kanban requires that the software be developed in one large development cycle. Considering this, it is surprising to learn that Kanban is considered Agile at all, and yet it fulfils all twelve of the principles behind the Agile manifesto, because while it is not iterative, it is incremental.

The principle behind Kanban that allows it to be incremental and Agile, is limited throughput. With no iterations a Kanban project has no defined start or end points for individual work items; each can start and end independently from one another, and work items have no pre-determined duration for that matter. Instead, each phase of the lifecycle is recognized as having a limited capacity for work at any one time. A small work item is created from the prioritized and unstartedrequirements list and then begins the development process, usually with some requirements elaboration. A work item is not allowed to move on to the next phase until some capacity opens up ahead. By controlling the number of tasks active at any one time, developers still approach the overall project incrementally which gives the opportunity for Agile principles to be applied. A typical Kanban board is illustrated below:



Kanban projects have Work In Progress (WIP) limits which are the measure of capacity that keeps the development team focused on only a small amount of work at one time. It is only as tasks are completed that new tasks are pulled into the cycle. WIP limits should be fine-tuned based on comparisons of expected versus actual effort for tasks that complete.

Kanban does not impose any role definition as say, Scrum does and along with the absence of formal iterations, role flexibility makes Kanban attractive to those who have been using waterfall-style development models and want to change but are afraid of the initial upheaval something like Scrum can cause while being adopted by a development team.

Summary

While Agile methods have more similarities than differences, it is possible to pick out one or two distinguishing factors for each.

Scrum	**Requires a ScrumMasterBurndown chartsLimited to code production**
XP	Pair programmingTest-first programmingSelectable ideas
Kanban	Incremental but not iterativeWork In Progress (WIP) limits
DSDM Atern	MoSCoW prioritizationHeavy in detailAddresses full lifecycle
DAD	Relatively newAddresses full lifecycleBorrows from elsewhere
AUP	Grew from RUPHeavy on pre-iterative phases
Feature Driven Development	Implementation by functional area
Lean Development	Ideas to overlay on other process

Agile Development Methodology

Bibliography:

- www.dsdm.org
- www.Agilemanifesto.org
- http://www.inflectra.com/Agile-Software-Development.aspx
- www.scrum.org
- www.scrumalliance.com
- http://disciplinedagiledelivery.com/
- A Practitioner's Guide to Agile Software Delivery in the Enterprise by Scott W. Ambler and Mark Lines, 2012; http://www.ambysoft.com/books/dad.html
- Going Beyond Scrum - Disciplined Agile Delivery by Scott Ambler, 2013; http://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf
- http://www.ambysoft.com/unifiedprocess/agileUP.html
- Software Development Methodologies by Raman Ramsin; http://sharif.edu/~ramsin/index_files/sdmlecture15.pdf
- www.featuredrivendevelopment.com
- *Lean Software Development* by David J. Anderson, 2012; http://msdn.microsoft.com/en-us/library/hh533841.aspx#BKMK_LeanAgile
- *Lean Software Development Principles* by John P Vajda, PMP,CSM, 2010; http://www.slideshare.net/jpvajda/lean-software-development-principles
- *The Principles of Lean Software Development* by Scott Ambler, 2010; https://www.ibm.com/developerworks/community/blogs/ambler/entry/principles_lean_software_development?lang=en
- *Feature Driven Development Overview*, Nebulon Pty. Ltd, 2005; http://www.nebulon.com/articles/fdd/download/fddoverview.pdf