# Thoughts on Agile

## What is Agile:

Agile is a way of developing software and other 'soft products' focused on flexibility and adapting to changing user or customer requirements to maximise value. In many circumstances the end user / customer either doesn't have the knowledge to specify its requirements fully before development starts, or circumstances change, or new insights during the course of the project suggest better ways of achieving value. Agile can adapt to maximise the value created in these circumstances.

Traditional project management focuses on up-front planning and estimating to define the project and then efficiently delivering the defined requirements 'to plan'[1]. The traditional approach is optimal in a wide range of circumstances including:

- When the outcomes or deliverables are well understood; eg, a typical construction project.
- When knowing the precise form of the outcome or deliverable is critical to other aspects of the overall program of works; eg, the software that is integrated with an aircraft's avionic control systems.
- When there are long lead time items to manufacture and then fit into other aspects of the project's overall work; eg, typical engineering and infrastructure projects.

These types of project are known as 'closed' or 'semi-closed' projects: the objective is clear[2].

For 'semi-open' and 'open' projects the challenge is altogether different.


he final objective is not clear and the ways of achieving the objective may or may not be known. is typical for most business software projects and business change projects. There may be a clear

[1] A criticism of the traditional 'Waterfall lifecycle' is that it follows a ridged single pass approach from planning to implementation. However, the original paper by Royce suggests arranging the project so the end result is the second pass through the lifecycle since it would be impossible to remove all the uncertainty on a single pass - yet many people equate the waterfall lifecycle to a single pass approach (with all the associated woes).

[2] For more on project typology, see: **Projects aren't projects – Typology**: https://mosaicprojects.wordpress.com/2009/04/09/projects-arent-projects2

aspiration or strategy, discovering how to get there is a journey. This is the space where Agile methodologies offer significant advantages over more traditional software development methods[3] provided the performing organisation can properly govern[4] and manage the agile development environment.

Agile is not a synonym for anarchy[5]. The Agile Manifesto outlines the philosophy for the 'movement'; see: http://agilemanifesto.org/. From this starting point a range of methodologies have developed including Scrum and XP.

## Understanding Agile:

The first key point of understanding is that Agile is not of itself a project management methodology. Agile is a soft product development methodology primarily used in software development but with significant potential in a wide range of application areas. The methodology can be used for routine operational maintenance of software within an organisation as effectively as for the development of a new software system within a project[6].

The difference between operations and projects can be summarised as follows:

- Projects are temporary, unique endeavours undertaken to achieve a defined objective; once the objective is achieved (or found to be unachievable or undesirable) the project is closed. The objective may be defined in traditional terms of time cost and scope, or in more outcome focused language such as a defined problem satisfactorily resolved but if there is not a defined objective to be achieved before closure, the work being undertaken is not a project.

  Project management is the process of defining scope, deciding on methodologies, creating teams, and all of the other project management processes defined in the *PMBOK® Guide*. When Agile is chosen as the product development methodology for a project it will certainly influence the way the project is planned, resourced and controlled but of itself, Agile is not 'project management'. Projects are delivered by temporary teams assembled to work on the unique project deliverable (as described in the Project Charter) and then reassigned to other work as the project closes down.

---

[3] For more on **selecting the right projects for Agile** see: https://mosaicprojects.wordpress.com/2010/06/13/selecting-the-right-projects-for-agile/

[4] For more on **governing agile** see: https://mosaicprojects.com.au/PDF_Papers/P177_Governing_Agile.pdf

[5] PMI define three types of lifecycle that use progressive development either as a stand-alone approach or as pert of a hybrid approach combined with elements of a traditional predictive project delivery methodology:

- **Iterative** (Spiral and other similar methodologies) – the end product (exit criteria) is developed progressively, the first iteration builds a 'rough framework' for the overall product, then each phase improves or refines the product through a series or repeated cycles (eg, to improve the efficiency of a process). Each iteration is usually quite small, and learning from earlier iterations are used to improve later iterations. The scope does not change without authorisation.
- **Incremental** - New features or functions are added during each cycle, phase 1 delivers part of the product in a usable form, later phases add the additional features or functions required to achieve the final end product (exit criteria). The scope does not change without authorisation.
- **Adaptive** = ***Agile*** – the project scope adapts to the evolving needs of the client. The focus is on customer satisfaction rather then delivering a pre-defined scope of works for a set price. The scope is expected to adapt to meet changing client requirements as everyone learns what is 'best' through the work of developing earlier iterations.

  Frequently iterative and incremental are combined with a simple prototype being developed first and then this initial 'working model' is progressively improved and enhanced during each project phase (iteration) until the full scope and functionality is achieved. The difference between 'Agile' and, 'iterative and incremental' lifecycles is the way scope changes – Agile expects the scope to be adapted to meet emerging client requirements, the other two options are focused on risk minimisation by avoiding a 'big bang' implementation.

[6] For more on this see: **Agile is NOT a Project Management Methodology**: https://mosaicprojects.wordpress.com/2009/03/05/agile-is-not-pm

- Operational work in IT tends to be characterised by stable teams working on dozens of minor objectives selected on the basis of an organisation wide prioritisation. The work still needs to be planned, managed, budgeted and resourced but so does all operational work. Unquestionably, Agile can be a very effective methodology for the management of IT maintenance work[7].

The major difference between operations and projects is permanent teams -v- temporary teams and the overall objective of the activity.

- Projects are about implementing a changed state for the organisation or community; eg, a new building. Creating a new capability.
- Operations are about maintaining and improving the current status quo; eg, typical plant and software maintenance. Incrementally creating an improved or enhanced capability[8].

Scrum and XP are Agile product development methodologies that can be chosen for many IT applications. They can usefully be deployed in both operations and projects (unlike 'waterfall' which is almost exclusively a project based methodology). Agile would probably also be extremely useful in other situations such as developing training materials and many business change projects where most of the deliverables are relatively intangible and subject to change based on new learning as the project progresses. However these advantages cannot turn them into an IT project management methodology any more than deciding to use a particular construction technique such as precast concrete can make 'pre-casting' a construction project management methodology.

## Agile Project Management:

There are two aspects to consider. One is applying Agile principles to the management of any project. The other is the need to adapt traditional project management processes to facilitate the effective management of an IT project using Agile as the product delivery mechanism.

The adaptations to traditional project management processes needed to allow Agile to work best are briefly discussed below, see also ***Managing Agile Projects***[9].

In the opposite direction, the Agile community has some good ideas to pass on to conventional project managers, including:

### Customer Engagement

While it may not be possible to iterate the building of a piece of machinery, engaging and explaining to the customer in their language -no jargon- what's happening will highlight issues early. If the customer doesn't like something, the sooner you know the better.

One of the key tenets of Agile is to engage effectively with your customer and end-users, understand their needs and problems, and then deliver an effective solution. This requires regular and effective communication, openness and accountability, and a good measure of trust to support robust relationships between the project team and their key stakeholders.

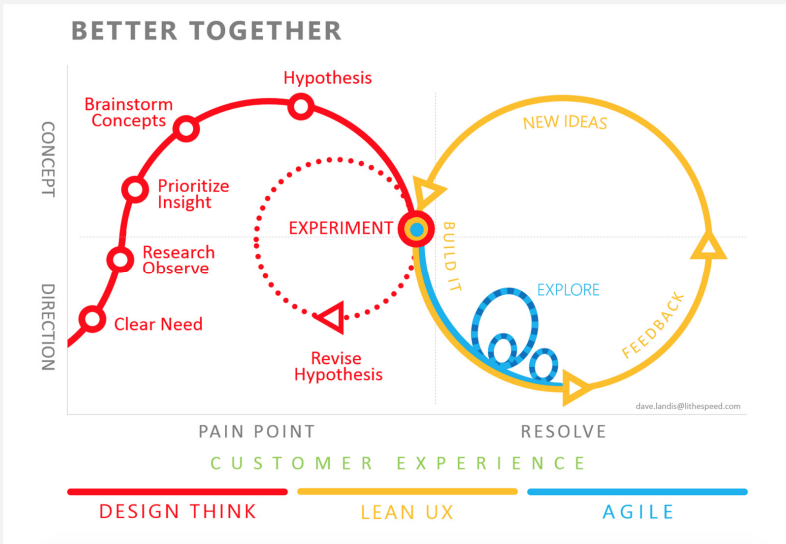
---

7 For more on this see: **De-Projectising IT Maintenance**: https://mosaicprojects.wordpress.com/2009/03/06/de-projectising-it-maintenance

8 For more on the definition of a project see: **Developing a concise definition of a project**: https://mosaicprojects.com.au/PDF_Papers/P007_Project_Fact.pdf

9 See **Managing Agile Projects**: https://mosaicprojects.wordpress.com/2009/03/07/managing-agile-projects

**Going Light and Lean**




Those are hardly new ideas, but they've been embraced by the Agile philosophy for a good reason, they work!

- Lean[10] was developed by Toyota as a manufacturing philosophy and has been adapted to many other areas. Some of its key principles, such as minimising unnecessary movement, simplifying process and continuous improvement, have huge potential in project management. The principles of 'Lean' are:
    - Specify value from the perspective of the end user or customer
    - Review all of the steps in the value stream for each product – eliminate those steps that do not create value
    - Make the value creating steps occur in a tight sequence so the product flows smoothly towards the customer
    - As flow is introduced let customers pull value from the preceding step (upstream activity)
    - Once the full system has been introduced, continually improve the process to eliminate all waste.
    - These principles are supported by 'lean enablers', practices that improve workflows:
        - Base human relations on respect for people
        - Define value from the perspective of the stakeholders
        - Plan the work as a 'value-add' stream (or streams)
        - Organise the value stream as an uninterrupted flow of tasks
        - Pull the work in progress forward as needed
        - Make all imperfections visible and pursue perfection.
- Light is focused on the minimising unnecessary overhead. Complex plans and processes should be simplified, but only to remove excess complication, not to remove core requirements.

Slimming down the project management overhead to its optimal level is probably the easiest way to free up the resources needed to engage your stakeholders more effectively and is definitely supported by A Guide to the Project Management Body of Knowledge (*PMBOK® Guide*).

[10] For more on ***Lean & Light*** see: https://www.mosaicprojects.com.au/WhitePapers/WP1046_Process_Improvement.pdf or http://en.wikipedia.org/wiki/Lean_manufacturing

## Project Managing Agile:

Traditional project management has grown up focused on typical engineering projects where the final product to be delivered is scoped, designed, built, tested and implemented – in that order. This is OK if the client knows what it needs precisely and the number of changes is relatively small. This paradigm is not so effective if the project is a ***quest*** to achieve an objective[11] and everything changes routinely.

Agile is a methodology ideally suited for developing projects where an iterative approach is needed to refine understanding and deliver value early, typical in many software and other 'soft' projects, but if the work is to be managed as a project how should the *PMBOK® Guide* processes be applied?.

The *PMBOK® Guide* has 9 technical knowledge areas[12]:

### Project Scope Management

Traditional project management expects scope management to define the output. In an Agile project the final outputs should be defined in terms of achieved capabilities, how the capability will be achieved will be discovered along the journey.

This makes 'Verifying the Scope' interesting. There needs to be clearly defined way to assess if the capability has been delivered. How do you measure a 'user friendly interface'? It's not impossible to do but how it's done needs to be clearly defined. Change control is also more challenging, as is configuration management.

### Project Schedule Management

Ideally time should not be an issue if the objective is to achieve a required capability. In reality there are usually deadlines.

In an Agile project, scheduling and workflow become closely aligned. The key requirement is an overall system architecture that defines the sequence modules need to be built in to allow progressive testing and implementation of capability. The software architecture defines the build sequence that defines the schedule.

Scheduling is at a much higher level though. A 'sprint' is likely to be a single activity of 1 to 4 weeks duration[13]. The sequencing of the 'sprints' and the number of sprints that can operate in parallel define the resource requirements and the project duration.

### Project Cost Management

Agile projects have to be based on a cost reimbursable system. One tool designed to include a degree of competition with the ability to properly compensate the contractor for its work was southernSCOPE the methodology requires tenders to bid on a project at a $ per function point rate based on a project description and the estimated number of function points. At the end of the project the same independent person who prepared the initial estimate, re-counts the function points and the price is determined.

This innate variability in Agile creates two issues for management the first is deciding if the project objective is to achieve as much as possible within a pre-defined budget (scope is the variable) or if the objective is to achieve a defined functionality (eg, a complete system) as cost effectively as possible. In either situation formal performance management techniques (Earned Value[14]) can still be applied,

---

[11] For more on the different type of project see ***Project Size and Categorisation***: https://www.mosaicprojects.com.au/WhitePapers/WP1072_Project_Size.pdf

[12] The *PMBOK® Guide* also includes 'Integration Management', however, the principles defined in Integration Management do not change based on the type of project being managed.

[13] Sprints are typically 'time boxed' and the amount of work included in each sprint designed to allow completion within the agreed time. For more on ***time boxing*** see: https://www.mosaicprojects.com.au/WhitePapers/WP1020_Time_Boxing.pdf

[14] For more on Earned Value see: https://www.mosaicprojects.com.au/WhitePapers/WP1081_Earned_Value.pdf

however, the performance metric is more likely to be either 'function points' or the 'story points' used to plan sprints and assess the backlog.

**Project Quality Management**

This is probably easier under Agile. The basic definition of 'quality' is 'fit for its intended use'. Quality is continually assessed by the involvement of the client and the iterative release of modules to production.

**Project Human Resource Management**

Basically remains unchanged but the skills of the people needed for an Agile project are likely to be different.

**Project Communications Management**

The level of trust needed to run an Agile project is much higher than a traditional project. Effective 'real' communications in all directions are essential. This is different to producing project reports!

**Project Risk Management**

Recognize you are on a journey focused on delivering value. Significant time and cost contingencies are needed and should be used to optimize the value of the final product.

**Project Procurement Management**

This should not change significantly BUT the procurement process needs to be aligned to what it is being bought. Agile works in a collaborative partnering space. In the engineering world these are call Alliance Contracts. Traditional contracts will not support Agile delivery methods.

**Project Stakeholder Management**

Effective engagement with core stakeholders, particularly the client is central to the Agile Manifesto. This aspect of project management is likely to have a much higher priority in an Agile project than a traditional project.

In conclusion:

- Tailor the processes in the *PMBOK® Guide* appropriately to work effectively with an Agile project delivery method and the overarching PM process will enhance the probability of success.
- Treat an Agile project in the same way as a traditional project and the PM processes will guarantee failure (or at least fail to contribute much)!

## The Three basic Agile Methodologies:

These are probably the three most popular agile methodologies:

**1. Scrum**

Scrum has found its way into a variety of projectized organisations, including law firms and universities. The non-profit Scrum Alliance[15] defines the key elements of Scrum as:

- A prioritised wish list called a product backlog is created.
- During the planning phase, the team selects a small chunk from the top of that wish list, called a sprint backlog, and decides how to implement those pieces.
- The team is given a certain amount of time, called a sprint, to complete its work and meets each day to assess its progress.

[15] For more on Scrum see: https://www.scrumalliance.org/

- At the end of the sprint—usually two to four weeks—the work should be ready to hand to a customer.
- The sprint ends with a sprint review and a retrospective.
- The next sprint then begins.

For Scrum to cross over into other industries, you need to be able to break down the requirements into a discrete set of items that could be worked across a set of iterations with usable (or defined) deliverables at the end of each sprint.

**2. Kanban**

Kanban originated in the motor vehicle industry and is adaptable to non-software development projects, including human resources and legal because its principles are not associated with any specific industry. Its key principles are:

- **Visualize the workflow**. This can be done by using a card wall, with the columns on the wall representing the states or steps in the workflow and the cards representing the work items.
- **Limit the works in progress**. Select the most important and valuable work items and keep the number small to ensure the team is making good progress.
- **Manage flow.** The flow of work through each state or step should be actively monitored, measured and reported in order to evaluate positive or negative effects of incremental and evolutionary changes.
- **Make process policies explicit**. Ensure an explicit understanding of the mechanism of a process to achieve a rational, objective discussion of issues—and facilitate consensus around improvement suggestions.
- **Improve collaboratively**. To truly leverage Kanban, teams must collaborate. As with any other agile method, the team should meet as a team to plan, meet daily for a stand-up, and can choose to do retrospectives to inspect and adapt their process.

To adapt to a human resources project, for example, visualize the hiring process through a Kanban board. Categories on the board would include a column for the candidates who submitted résumés, a column for candidates who are qualified for the position and a column for candidates who have moved past the phone interview process. Support that workflow with a document that outlines who is responsible for these different roles and kickoff the process with a short meeting attended by all stakeholders. (For more on Kanban see separate heading below on page 12).

**3. Extreme Programming (XP)**

XP focuses on test-driven development, small releases and a team structure that includes the customer. Many of the rules for this agile methodology are designed specifically to address coding, designing and testing. XP recommends planning the release at a high level, then planning each iteration at its start (or every two weeks).

## Key Agile Processes:

Some of the key differences between an 'Agile' approach to software development and the more traditional 'waterfall' approach include:

**Agile Requirements Gathering with User Stories**

Knowing the goals of your end users and project stakeholders is the necessary first step to any successful project, so just as in traditional projects, Agile projects start with basic requirements gathering. However, instead of trying to nail down all of the details up front, Agile seeks to capture just enough information to have a detailed conversation with the customer at a later date. This information is gathered as 'user stories'; short descriptions of the functionality in customer terms[16].

[16] For more on ***traditional requirements gathering*** see: https://www.mosaicprojects.com.au/WhitePapers/WP1071_Requirements.pdf

A user story is a placeholder and a promise to have a future conversation about the needed functionality. A useful structure for brainstorming user stories (although certainly not the only one) is as follows: *As a [Type of User], I want to [Function to Perform] so that [Business Value].* This format focuses the story descriptions so that they stay customer and business value oriented instead of technical in nature.

For example, in a sales management system, some typical user stories might be:

- As a sales person, I want to add a new contact so that I can follow up later with prospects.
- As a sales manager, I want to view new contacts added by salesperson, so I can track leads
- As a system administrator, I want to add a new sales person so they can access the system

Delaying the detailed conversation until shortly before development avoids some of the waste that is typical of projects where detailed requirements are gathered early, but become invalid before the work begins. Additionally, some requested features may no longer be needed by the customer after a few months, saving the cost and effort of a detailed analysis and design.

To develop the most comprehensive list of requirements, it is important to define all of the key stakeholders first, with a particular focus on the business users of the system, using a tool such as the ***Stakeholder*** *Circle®*[17]. Once the appropriate stakeholders are identified, many options exist for capturing the requirements including focused interviews and group sessions using techniques such as brain storming and 'Six Thinking Hats[18]'.

A key strength of Agile development approaches is the ability to incorporate new requests for functionality as they are discovered. If some of the user stories aren't uncovered in the first planning sessions they can easily be added later; particularly if the business climate changes or if the stakeholders find they have forgotten something important.

**Define the architecture:**

Incremental improvements and maintenance work/projects work on an existing software and hardware infrastructure and can largely be managed in an agile environment through prioritisation (discussed below) and 'burn down' approaches to selecting work to incorporate in a sprint, with the application of common sense. However agile can also be applied to the development of new systems; in this situation a critical element is defining the ***technical architecture*** needed for the application to work. This refers to defining and designing the hardware, software, databases, connectivity (internally and to other systems), and other 'things', required to allow the new system to function, and to a lesser extent how the different components in the new system will interact. There is absolutely no point in designing a set of screens to fulfil a user requirement if the supporting database and hardware are not available (the Agile objective for each sprint or iteration is working code). This has to be developed very early in the project lifecycle - for instance in the setup sprint. Factors to consider in the design of the architecture include:

- **Hardware**. Identify the hardware your solution will run on and any other hardware that will be needed. Plus any hardware the system will interact with such as cell phones, personal digital assistants (PDAs), printers, scanners, bar code readers, etc.
- **Software**. Identify any software and tool requirements. This would include things like the client and server operating systems, browser type, third party software packages, etc.
- **Interfaces**. The major interfaces should be identified including data transfers to/from various major components in the system being developed plus other applications, vendors, clients, etc.
- **Network**. The network that is needed to support the solution should be diagrammed. This includes modems, lines, routers, hubs, etc.

---

[17] For more on the ***Stakeholder*** *Circle®* see: https://mosaicprojects.com.au/PMKI-TPI-075.php

[18] For more on **Six Thinking Hats** see: https://mosaicprojects.com.au/Mag_Articles/P038_6_Thinking_Hats.pdf

- **Firewall/security**. If your solution needs to run outside of your internal network, you will probably need to incorporate security features such as a firewall. In fact, you may need two firewalls (or more) to protect company data from unauthorised outside access.
- **Datastores**. Identify the major datastores and the specific package/vendor involved. For instance, if you utilize a database, identify the specific software (Oracle, SQL Server, etc.). Do the same for data marts, data warehouses, major files, cloud applications, etc.
- **Tiers**. Many solutions are created using a two tier (client-server) or three tier approach. Web solutions, for instance, are typically designed in three tiers.

Detailed documentation is not needed; the architecture could be defined on a whiteboard or flipchart. However, it is important that the information be shared with the team for additional ideas and concerns. In general, the more complex your project architecture is, the more potential problems you will encounter over time. Every piece of hardware and software, and every programming connection and data exchange is subject to failure and bugs. The best solutions for long-term stability are the simple designs that achieve the *minimum functionality* required using as few components as possible.

It is important that the *project technical architecture* is created by experienced staff because the architecture will create far-reaching implications based on a limited amount of information. The architecture does not have to be perfect the first time. However, it is important that the architecture be as close as practical to the final outcome and it is also important that it be flexible.

**Prioritising the Work:**

After the initial requirements gathering, the user stories are prioritised with your customers:

- Prioritise the full list based on what's important to your customer.
- Select a subset of these for your first release.
- Then choose an even smaller subset for your first iteration or sprint
  (a fixed length time period during which development will take place, usually 1 to 4 weeks).

It is during the iteration planning process that the more detailed conversations with the customers occur to define the details of the 'user story' and clarify any issues. This close contact with the customer continues during development to enable the asking questions as needed.

**Iteration / Release Planning:**

Planning a release for an Agile software development is a collaboration between the development team and customer team. Once a list of candidate user stories have been identified for the release, the development team estimates the relative effort for each of the software features, as an input for prioritising within the release.

One technique for this high level estimate is to use 'points', so that the estimates are based on the relative size of each function (this one is twice as big as that one).

For example, for five stories around an online book store, the estimates might look like this:

- As a customer I want to browse books by category 4 points
- As a customer I want to search books by title 2 points
- As a customer I want to search books by author 2 points
- As a customer I want to buy a book with a credit card 8 points
- As an administrator, I want to add books to the store 4 points

This point allocation suggests the team believes that the search-related stories are about the same size (2 points), the 'browse' story is twice as difficult as those (4 points), the 'buy' story is significantly harder (8 points), and the administrative story is about as hard as the browse story (4 points).

To plan the iteration you start with the expected capacity of the team. If a similar team completed 10 points during the last release, then that could be the starting point for planning this one. However, for

the initial iteration, it's best to be conservative - you can always add more functionality if the first few iterations show that you can get more done than you thought.

Based on the prioritised list of user stories and their estimated size a balanced set of work is determined for the first iteration. You prioritise based on the business value of the story, as well as the developer estimate. In more complex developments, the software architecture may require development in a particular sequence; eg, until some data tables are developed, a screen input module may not be possible to develop and test.

Different combinations of stories should be considered to fill the release capacity in the most sensible way. For example, it might make more sense to schedule 2 or 3 smaller stories of less importance instead of a single larger, slightly more important story. Tools such as ExtremePlanner[19] let's you try out these scenarios until you've reached a stable point.

In the scenario above, the first iteration may build the administrative and browse functions with one search. This would allow the book store to be populated with books and the basic system road tested whilst the purchasing function is built together with the second search as a second iteration. Once both iterations are 100% complete, the first release of the bookstore to the buying public can occur.

To make this work, and fulfil customer needs, at the beginning of every iteration a meeting is held between the product owner and the project team to determine the workload for the new iteration. During the meeting the product owner evaluates the requirements backlog and pulls off the next set of user stories that are of the highest priority (the level of effort for each user story should have been assigned when it was added to the backlog) and the project team agree to take on as many story points as they can complete within the iteration to optimise 'velocity' and maintain 'rhythm'. User stories that are selected for a iteration need to be completed in that iteration. Which brings into consideration another key Agile tenet: that before each iteration can be completed; the functionality must be fully tested and signed off by the client.

Optimising 'velocity' is important, the workload to stay relatively even from iteration to iteration. If the project team found it was not able to complete a set of user stories in the time allowed for a prior iteration, the team can agree to take on less work in the next. Likewise if the team realises that they could have done more work in an iteration they should take on more work in the next. The pace at which the team can complete story points from the backlog is known as the team's 'velocity'.

Maintaining 'rhythm' is important. In an Agile project it is important to stay on a steady iteration cycle If a story is not ready when the iteration is ready to move to production, the affected code needs to be pulled out so that the remaining code from the iteration can be released on time. There should be no delays to the iteration completion date. The focus is on hitting the end-date over and over again. This steady pace for each iteration is called the team "rhythm".

**Test Driven Development (TDD)**

This term refers to a process of understanding the requirements within a user story and then immediately defining the tests that are used to validate that the requirement exists and is correct. As each test is created, code is written to validate that the test works as planned. The logic for this technique is that when code has been written to validate each possible test of a requirement, the code to implement the requirement is completed. In addition to the code being completed, the testing process should be simple since the appropriate tests were already defined and the code was built to successfully pass the tests.

This TDD technique does not always catch logic that is required between components, or where a cascade of events across different components results in an error. So, the TDD approach to development must also be combined with more of a big-picture view of how the components interrelate so that the full solution minimizes defects.

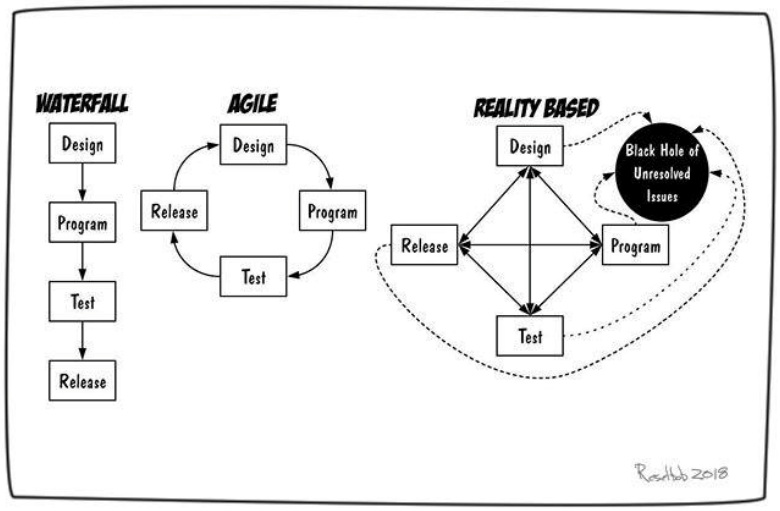
[19] For more on **ExtremePlanner** see: http://www.extremeplanner.com

Each test case should describe how you will validate one feature, function or requirement. A typical test case contains the following information:

- **Test case ID**. This is the ID of the test case to be used for tracking throughout the testing process. If you are tracking requirements, you can use this test case ID to cross-reference with the requirement ID.
- **Feature** / **function** / **requirement to test**. Describe the feature, function or requirement you are testing. If the requirements are numbered, this can be cross-referenced here as well, rather than repeating the requirement again.
- **Data or activities required to test**. Describe the test data and the input that is required. If there are multiple values required to test certain conditions, they can be listed on each line. For instance, if you are testing the acceptance of a credit card number, you might want to include test data for each of the following:
  - A valid credit card number (probably a dummy number given by the bank)
  - An invalid credit card number
  - A number with too few digits
  - A number with too many digits
  - A number with alphas
  - A number with blanks separating each set of four numbers
  - A number in an invalid format (blanks or dashes separating the sets of four numbers)
- **Expected results**. Describe the expected output for each specific condition.

These test cases are used to build test data for the initial tests. Additional test cases can be defined during the testing process.

**Day-to-Day Agile Management**




One of the key attributes of an Agile approach to software development is the emphasis on close communication within the team as well as with the customer. Team communication usually takes the form of a brief daily meeting where the team members can share progress and identify obstacles. In the Scrum method, this meeting is called a *Daily Scrum*, while the XP method refers to this as a *Daily Stand-up Meeting*.

The purpose of the daily meeting is for each team member to communicate three critical items:

- What they've accomplished since the last meeting

- What they are planning to work on next
- Obstacles or challenges they have encountered

It's important that the team is safe to honestly communicate status and issues in the daily meeting because the overall project controls are likely to be at the iteration level, not the detailed activity level inside each iteration [see: Managing Agile Projects[20]].

Project managers should take care not to stifle the flow of information by challenging honest reporting or trying to control the meeting. The successful Agile PM focuses on supporting the team and removing obstacles to their success [see: Servant Leader[21]].

The use of a visual aid in the meeting room to show the stories and tasks that they are being worked in the iteration on is valuable. This can take the form of a taskboard with index cards that can be moved around, or a computer and a projector that can show an electronic representation of the iteration. The taskboard should be updated either during or before the daily meeting. This usually involves identifying any completed tasks from the previous day, updating in-progress tasks with new estimates (if necessary) and team members selecting new tasks to work on today.

JIRA, a tool developed by Australian Company Atlassian, can be used for bug tracking, issue tracking, and project management. The name 'JIRA' is actually inherited from the Japanese word "Gojira" which means "Godzilla". The basic use of this tool is to track issues, and bugs in your project and developed code.

**Managing the Iteration and Release Date**

Agile methods use iterations, or short, time-boxed development cycles[22]. During an iteration, developers work on the highest priority features. They plan to complete whatever they committed to by the end of the iteration. However, due to the unpredictability of software development, it is likely the team will have too much to do (and less frequently, not enough to do) if they are estimating honestly.

When there is too much to do, the best option is to cut scope, since changing the iteration date undermines the primary advantage of time-boxed iterations - achieving predictable and quick, technical and business feedback (see discussion on rhythm and velocity above). Adding more resources part way through an iteration rarely works.

There are three effective ways to cut scope while preserving high quality, each of which has different tradeoffs:

- **Simplify Over-engineered Designs:** Make sure that you are doing the simplest thing that meets the requirements for today's functionality, testability, and ease of maintenance – don't over-engineer for the possible future.
- **Simplify Features:** Most specific feature requests can be solved by alternative means that still address the business problem. An automatic notification whenever any order is placed may be solved by a complex real-time change to the on-line order system, or by an hourly email with an order summary report. The latter might be a matter of minutes to implement with a simple database report, while the former might mean intrusive, risky changes to the order processing logic in the system.
- **Cut Low Priority Features:** As a last resort, you may need to eliminate the lowest priority features to make a deadline.

---

[20] **Managing Agile Projects**, see: http://mosaicprojects.wordpress.com/2009/03/07/managing-agile-projects

[21] **Servant Leader**, see: https://mosaicprojects.com.au/WhitePapers/WP1014_Leadership.pdf

[22] For more on **timeboxing** see: https://www.mosaicprojects.com.au/WhitePapers/WP1020_Time_Boxing.pdf

You build trust and confidence by involving the stakeholders in tough decisions while there's still time for them to react; so communicate with your customer about the proposed cuts as soon as possible to allow them to re-negotiate priority if necessary and maintain value.

## Kanban development:

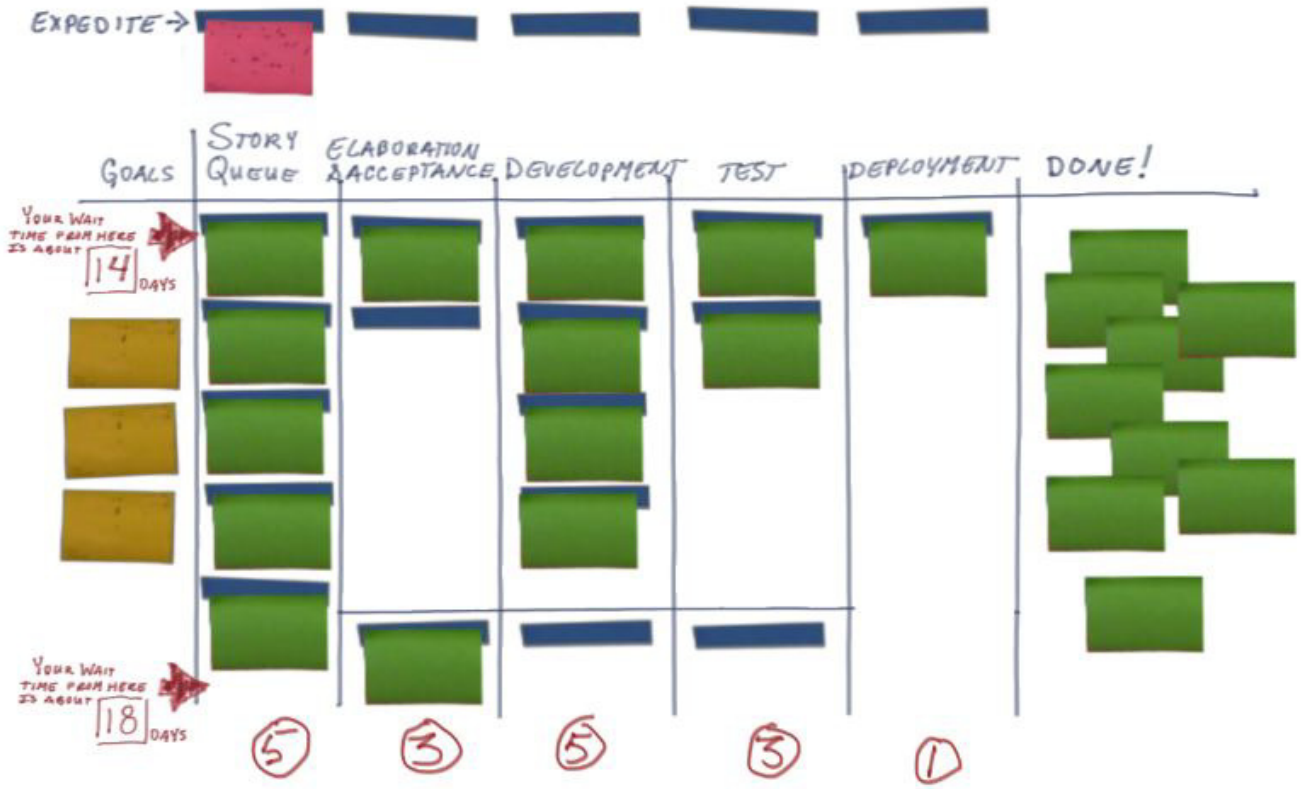
Kanban is a lean manufacturing process designed to eliminate unnecessary work in progress[23]. Kanban thinking applied to Agile development results in sweeping changes including:

- time-boxed development is out
- stories are larger and fewer
- estimation is optional or out completely
- velocity is replaced by cycle time

Kanban development revolves around a visual board used for managing work in progress.

The basic idea is stories start on the left side of the board and move quickly through the phases of development necessary for them to be considered 'done'. Finished stories, ready to release into production pile up at the end.

Each process step column is divided into two parts: The top is used for stories currently in progress in that phase. The bottom is the buffer. When work for that phase of the story is completed, it moves from 'in progress' in the current phase to the 'buffer' where it will wait to be pulled into the next phase.




Because this is a Kanban board the amount of work in progress is limited, as is the number of stories allowed on the board. The numbers written on the bottom limit the number of stories allowed at each station. The stations themselves are not fixed; they are optimised for any particular software

[23] For more on Kaban in manufacturing see: http://en.wikipedia.org/wiki/Kanban

environment. The number of stories allowed overall and at each station are a factor of the people capable of doing the work. The team needs to be balanced and flexible to minimise throughput time and avoid bottlenecks at individual stations.

Kanban stories are larger then the minimal stories used in other Agile techniques. Each story represents a 'minimal marketable feature' (MMF). To be marketable the feature needs to be large enough to be useful. A MMF may take weeks to build. But the important thing isn't how long it takes to build, but that it will be understandable and valuable to those who'll receive it.

## Conclusion:

Agile is an effective tool for use in delivering products where the outcome is uncertain. Whilst created as a software development methodology, many of the ideas can translate to mainstream project management.

---

First published 8th October 2010, augmented and update.