# Agile Cookbook A wiki guide to Agile Delivery in the 90 Day Cycle

## Introduction

This is a Cookbook designed as a guide to applying Agile practices within the **TimeboxedDeliveryCycle**. It is intended that this will become the primary source of practical and pragmatic advice for programmes taking on Agile delivery.

It's important to be aware that there are many approaches to Agile software delivery, and that all of these emphasise the adaptation of process to suit the specific problem at hand. You should take the time to read the **AgileManifesto**, and the supporting **AgilePrinciples** to understand the context within which this guide sits.

The focus of the Cookbook is on days 4 to 90 of the **TimeboxedDeliveryCycle**, effectively picking up directly after the end of the hothouse. The aim is to provide practical advice throughout the remainder of the delivery cycle. This advice is constructed to complement existing information about **HotHouse**, **AccelerateFramework** and **Tools**.

The Cookbook is not intended to be a general survey or guide to Agile practices or methodologies, for that we suggest you look at **FurtherReading**. Rather it is an approach to implementing the principles described in the **AgileManifesto** within the context of the **TimeboxedDeliveryCycle**. It is anticipated that the advice given here will be improved by feedback based on the practical application of this Cookbook.

The Cookbook will provide answers to the following questions:

- "What should I be doing?" :- Identifies how each **AgilePractice** should be undertaken at each stage - such as a focus on business value and a test-first approach. The minimum starting set of **AgilePractices** to be adopted when introducing Agile to your projects is described in **CorePractices**.

- "How do I do it?" :- Provides pragmatic step-by-step advice to **AgileRoles** and the way of working across the **TimeboxedDeliveryCycle** and recommend Agile activities/devices which should be followed to help achieve success.

- "Why should I/we do it?" :- Why make the change to agile and what are the business benefits?

- "What do I bring to the process?" :- The cookbook is structured around each of the **AgileRoles** in the development process, and the activities in which they will be involved.

- "How has it been done elsewhere?" :- Provides supporting information/collateral such as **CaseStudies** from peer organisations to help explain the reasons for making the change to Agile and to demonstrate how to put Agile into practice.

- "How do I get help?" :- See the **Help** or visit the google group where questions can be logged to be answered by peers who have experienced similar situations or by experienced moderators.

## Background

The material for the **AgileCookbook** has been seeded by BT and Thoughtworks.


# AgileOverview

There are a number of Agile development practices, for example XP, DSDM, Crystal, Scrum and others. They all share the same underlying principles expressed in the **AgileManifesto**, which emphasises working software, people, collaboration and responding to change.

Agile practices have been proved over the last five years in organisations to have considerable benefit in delivery. However there are still some **AgileMyths** which are aimed to be countered by the **AgileCookbook**.

## Major benefits:

- business and customers have control over priorities
- faster delivery of business benefits
- reduce risk through more frequent delivery
- higher quality of deliverable through continuous testing
- greater visibility and more accurate reporting

## Fundamental Principles

- **LoweringTheCostOfChange** during the delivery life cycle in order to unlock greater value.
- Focusing on delivering **WorkingSoftware** delivered incrementally in every **DevelopmentIteration**
- Expressing requirements in **UserStories** each of which are tied to **AcceptanceCriteria**
- Promoting **CollectiveOwnership** and **CommonVision** within the entire project team throughout all the **AgileRoles**

## Key Messages

- Change will always happen as you cannot predict the future. Clairvoyant Engineering does not apply so work to minimise the cost of the change instead of enforcing a change control system that is designed to stop or resist changes.
- When changes come, do the analysis and identify trade-offs to meet the revised scope in the current dates. Identify what can be dropped from the scope.

- Requirements must be related to "User Stories". A **UserStory** is written from the viewpoint of a user of the system and shows what they need to do their job. You must also add the business priority for each user story either in monetary terms or MOSCOW priorities. This priority is used to manage scope and change.

- Do not try to predict all the problems. Do enough to deliver the user story. This will reduce the work done on the projects and will reduce costs. Often too many extras are added "in case" and turn out to be wasted.

- Focus on the highest value for the next iteration to feed the ROI. When completed, show it to the business user. Expect that there will be changes in scope as a result.

- This is not "anti-planning" as you need to ensure that the team are working on things that can be measured:
  - What is delivered (working tested user stories)
  - How much did it cost to get here = predictions for future iterations. So use this to improve your estimations
  - Collect the statistics on what was done / tested to confirm status.

- Leads to a high degree of automated testing (metrics)

- This is a collaborative activity with Design+Analysis+Development+Test all together

- **ContinuousIntegration** (at least daily) within the system

## CorePractices

The **AgilePractices** listed here are what has been determined to be the core set of practices for the adoption of Agile. Programs which adopt all of these practices will be deemed to be practicing Agile.

### 1) Iterative Development

Agile projects base delivery of software and other project outcomes around fixed periods of time called Iterations. Specifically, software is delivered in a series of **DevelopmentIteration**s of equal duration. Other non-software deliverables can benefit from an iterative approach to delivery. The necessary conditions being that each Iteration should deliver a measurable, concrete result - preferably with a clear associated **BusinessValue**. In some cases this may simply mean a phased implementation, however it is preferable to attempt to incrementally deliver testable working product at the end of each iteration in order to benefit from the associated early feedback and testing. Iterative delivery is based on the principles of Lean Manufacturing, as described in this article by Mary Poppendieck - and as such is very applicable to non-software projects.

- What is Good: Each **DevelopmentIteration** will be planned and reviewed separately allowing for frequent adaptation to changing priorities. All code written in an iteration will have been designed, tested and accepted by the end so, theoretically, the code could be released at the end of the iteration if required. An **IterationPlanningMeeting**, **IterationKickOff** and **IterationRetrospective** should be held for each **DevelopmentIteration**.
- What is Excellent: In addition to the above, each **DevelopmentIteration** will deliver a "vertical slice" of the overall functionality. For example in an online banking scenario, iteration one may develop just the statements, iteration two may develop just the payments facility from front end to persistence. Each **DevelopmentIteration** should be no longer than two weeks in duration.

## 2) Automated Testing

Automated testing is emphasised in Agile projects because of the rapid pace of change and the necessity to be able to deliver **WorkingSoftware** frequently. Practices such as **TestDrivenDesign** and **TestDrivenDevelopment** as well as **ContinuousIntegration** all benefit from automated tests. The underlying principle is the efficient delivery of timely feedback. In the case of hardware, the term 'smoke-test' is often used to describe powering up the device to see if it emits smoke - which would indicate a defect. Look for opportunities to provide feedback in cost-effective ways in all areas of your project, not just software development. For example the **DailyStandUp** meeting provides timely feedback to the whole team on progress and issues, for the cost of 10-15 minutes of time per day.

- What is Good: **AutomatedUnitTests** are written alongside code. **AutomatedUnitTests** are included in a **DailyBuild** and can be run by developers on their workstation prior to checking in code to prevent regression. **AutomatedAcceptanceTests** for the **UserStories** delivered in a **DevelopmentIteration** are completed by the end of the following **DevelopmentIteration**, and can be run against the code in a dedicated test environment.
- What is Excellent: **AutomatedUnitTests** provide greater than 80% code coverage. **AutomatedUnitTests** and **AutomatedAcceptanceTests** are included in the **ContinuousIntegration** build and run every time code is checked in.

## 3) Continuous Integration

**ContinuousIntegration** refers to a fully automated build and test process that allows a team to build and test their software many times a day. The first step on the way to **ContinuousIntegration** is sometimes a **DailyBuild**. **ContinuousIntegration** underpins **LoweringTheCostOfChange** to allow late changes in requirements or structual improvements to be safely incorporated into the software. The underlying principle is that if the entire system is re-validated after every small change, then it is easy to identify the cause of any issue and resolve it. This principle can be applied to non-software development work wherever an opportunity to efficiently test after each change can be identified.

- What is Good: An automated, IDE independent **DailyBuild** incorporating (compile, test, build, deploy), running on a dedicated environment (not a developer workstation). The **DailyBuild** must be executed against the latest set of clean source code checked out from the **SourceControlRepository**. All source code must be stored in a **SourceControlRepository**, including build scripts, configuration files, test code, database setup scripts and anything else that would be required to build and install the software on a new environment. Any broken tests or other issues with the **DailyBuild** are addressed as a priority, ahead of implementing new functionality.
- What is Excellent: An automated, IDE independent **ContinuousIntegration** build incorporating (compile, test, build, deploy), which runs after each check-in to the **SourceControlRepository**. Any broken tests or other issues with the build are addressed as a priority, ahead of implementing new functionality. Developers run tests locally on their own workstations prior to checking in code to avoid breaking the build.

## 4) User Stories

The **UserStory** is the basic unit of scope in an Agile project. During the project each **UserStory** is progressively elaborated from being little more than a one-line description through to a set of **AcceptanceCriteria**. **UserStories** are a very effective mechanism for decomposing requirements into prioritised, testable, estimatable bite-sized pieces of work. As such they can be used in many non-software development projects, in fact - they were used to drive the delivery of this Cookbook.

- What is Good: A user story is written in the context of the value a feature would give an end user of the system. A standard way of writing them could beâ€¦ As [insert user] I need to [insert feature] so that [insert business value].

User stories are initially 1-2 lines long and contain enough detail to be testable, and provide an estimate and

priority. Additional detail is added prior to the **DevelopmentIteration** in which the **UserStory** will be delivered. Each **UserStory** should be of a size that would allow it to be completed within the period of one **DevelopmentIteration**.

- What is Excellent: In addition to the above, each **UserStory** should have predefined acceptance criteria which can be transformed into **AutomatedAcceptanceTests**. Each **UserStory** should be sized such that a **Developer** could complete two of them in a single **DevelopmentIteration**. Each **UserStory** should have a clear **BusinessValue** associated with it.

### 5) Customer Involvement

Agile methods consitently emphasise ongoing involvment of the **Customer** or **CustomerProxy** with the delivery/development team throughout the iterations, providing constant input and feedback. This is consistent with project managment practices that span all types of delivery, from software through to tailoring a bespoke suit or designing an office building. Regular provision of deliverables that allow the **Customer** and other important stakeholders to give informed feedback is critical to the success of any project, preventing the product from diverging from the **Customer**s vision, and ensuring effort is directed at those areas giving the greatest return on investment.
Having close collaboration with an empowered **Customer** improves productivity, since the time spent waiting for decisions to be made or recovering from incorrect assumptions is greatly reduced.

- What is Good: The **Customer** will play an active part in the **IterationPlanningMeeting**, **IterationKickOff** and **IterationRetrospective** sessions, providing feedback on current development and re-evaluating priorities. Rather than attempting to produce an exhaustive requirements spec up front, the customer will commit to elaborating on requirements as and when necessary in conjunction with the developers.
- What is Excellent: An empowered **Customer** (or **CustomerProxy**) co-located with team and involved day-to-day. The **Customer** works closely with the **BusinessAnalyst** team to elaborate **UserStories** and set **AcceptanceCriteria**.

## AgileRoles

- The **Customer** or **CustomerProxy** who is responsible for defining the requirements, priorities and accepts delivery of the completed **UserStories**.

- The **ProjectManager** who is responsible for delivering the completed system to the **Customer**.

- The **BusinessAnalyst** (who often acts as a **CustomerProxy**) is responsible for ensuring that the requirements are fully formed into proper **UserStories** with accompanying **AcceptanceCriteria**.

- The **Designer** is responsible for ensuring a coherent technical design with appropriate levels of quality, performance etc which satisfies the **Customer**'s requirements.

- The **Developer** is responsible for delivering software code which fulfils **UserStories** by meeting their **AcceptanceCriteria**.

- The **Tester** is responsible for ensuring that the **AcceptanceCriteria** tests are run and that they pass. In addition they are responsible for ensuring the overall quality of the system.

### Other roles that are useful to consider when adopting AgilePractices

- The **AgileCoach** is an experienced Agile practitioner who is responsible for helping the team adopt **AgilePractices**. The **AgileCoach** may also take on the **IterationManager** role.

- The **AgileEnablers** who work within the team in specific roles. Agile favours learning via interactions between individuals, teams benefit greatly if they are seeded with experienced **AgileEnablers** in areas such as the **Developer**, **BusinessAnalyst** and **Tester** teams.

- The **BuildMaster**, usually a **Developer** who has specific experience and skills in setting up and maintaining a **ContinuousIntegration** environment.

- The **IterationManager** who takes on the inward focusing parts of the **ProjectManager** role, specifically dealing with ensuring that the team is productive and that the iterative process is running smoothly.

- The **TechnicalLead**, usually a senior **Developer** working within the **Developer** team, who is responsible for ensuring that the technical **CommonVision** is maintained. The **TechnicalLead** works closely with the **Designer** in order to both communicate design goals to the team and to provide feedback to the **Designer** about implementation issues.

## TimeboxedDeliveryCycle

The approach described in this Cookbook is to break the **TimeboxedDeliveryCycle** down into a number of relatively short **DevelopmentIteration**s. Following an initial **IterationZero**, functionality is added incrementally as **WorkingSoftware** delivered at the end of each **DevelopmentIteration**.

This Cookbook provides role-specific walkthroughs, which detail how each of the **AgileRoles** is involved throughout the **TimeboxedDeliveryCycle**:

- **CustomerWalkthrough**
- **ProjectManagerWalkthrough**
- **DesignerWalkthrough**
- **DeveloperWalkthrough** (and **TechnicalLeadWalkthrough**)
- **TesterWalkthrough**
- **BusinessAnalystWalkthrough**

## FurtherReading

General online resources and books about Agile practices.

### Online Resources

The Agile Manifesto
The Agile Alliance
Martin Fowler's website
Agile Software Development on the C2 Wiki
Scrum
Extreme Programming

Ron Jeffries' website
Alistair Cockburn's Crystal Wiki
David J. Anderson's website
Scott W. Ambler's Agile Data website
Open Source Testing
Bret Pettichord's Agile Testing website

## Books

Refactoring: Improving the Design of Existing Code; Martin Fowler et al
Test Driven Development: By Example; Kent Beck
Agile Software Development, Principles, Patterns, and Practices; Robert C. Martin
Agile and Iterative Development: A Manager's Guide; Craig Larman
Agile Project Management : Creating Innovative Products (Agile Software Development Series); Jim Highsmith
User Stories Applied: For Agile Software Development (Addison-Wesley Signature Series); Mike Cohn
Agile Software Development with SCRUM; Ken Schwaber, Mike Beedle
Extreme Programming Explained: Embrace Change; Kent Beck
Test-Driven Development in Microsoft .NET (Microsoft Professional); James W. Newkirk, Alexei A. Vorontsov
Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results; David J. Anderson
Managing Agile Projects; Sanjiv Augustine
Agile Software Development; Alistair Cockburn
Agile Software Development Ecosystems; Jim Highsmith
Patterns of Enterprise Application Architecture; Martin Fowler et al
Domain - Driven Design: Tackling Complexity in the Heart of Software; Eric Evans
Lessons Learned in Software Testing; Cem Kaner, James Bach, Bret Pettichord