Agile Methods

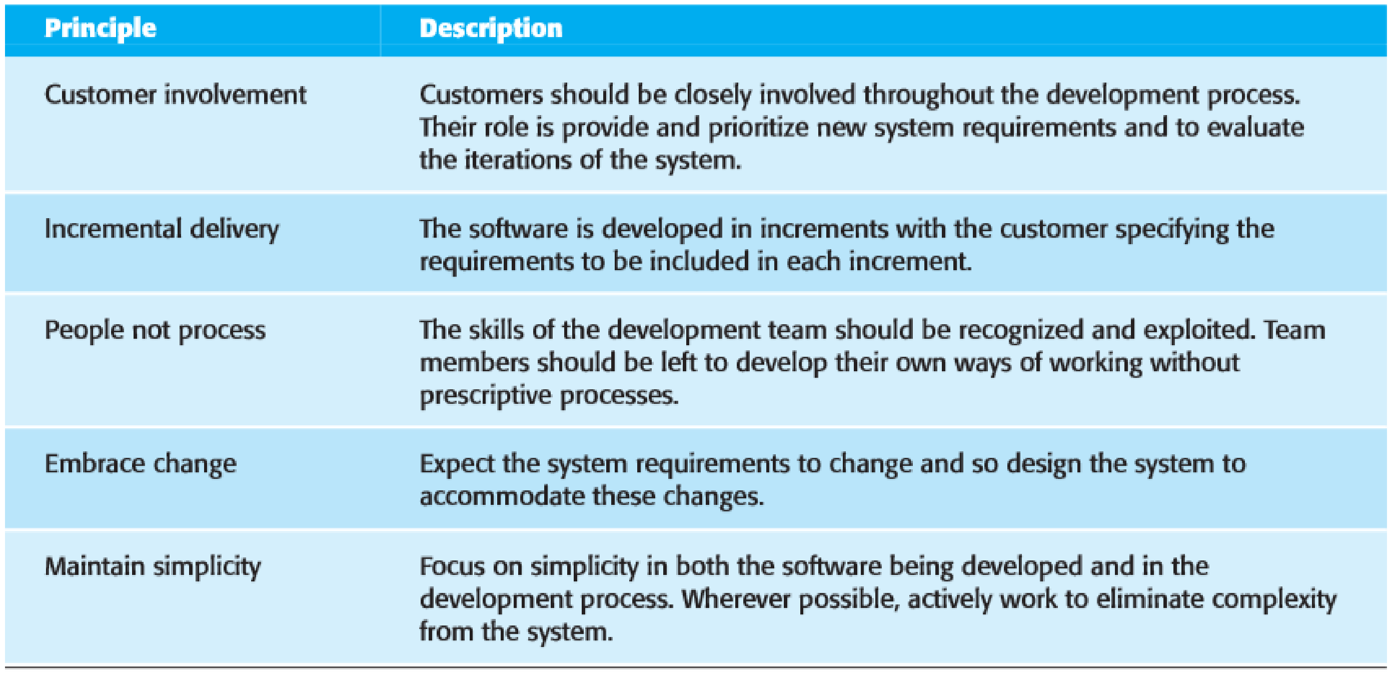
Introductions

- The heavyweight, plan-driven development approach is applied to small and medium-sized business systems, the overhead involved is so large that it dominates the software development process.
- More time is spent on how the system should be developed than on program development and testing.
- As the system requirements change, rework is essential and, in principle at least, the specification and design has to change with the program.
- Dissatisfaction with these heavyweight approaches to software engineering led a number of software developers in the 1990s to propose new ‘agile methods’.
- These allowed the development team to focus on the software itself rather than on its design and documentation.
- Agile methods universally rely on an incremental approach to software specification, development, and delivery.
- They are best suited to application development where the system requirements usually change rapidly during the development process.
- They are intended to deliver working software quickly to customers, who can then propose new and changed requirements to be included in later iterations of the system.
- They aim to cut down on process bureaucracy by avoiding work that has dubious long-term value and eliminating documentation that will probably never be used.
- Probably the best-known agile method is extreme programming.

Agile methods have been very successful for some types of system development.

1. Product development where a software company is developing a small or medium-sized product for sale.
2. Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.

The principles of agile methods

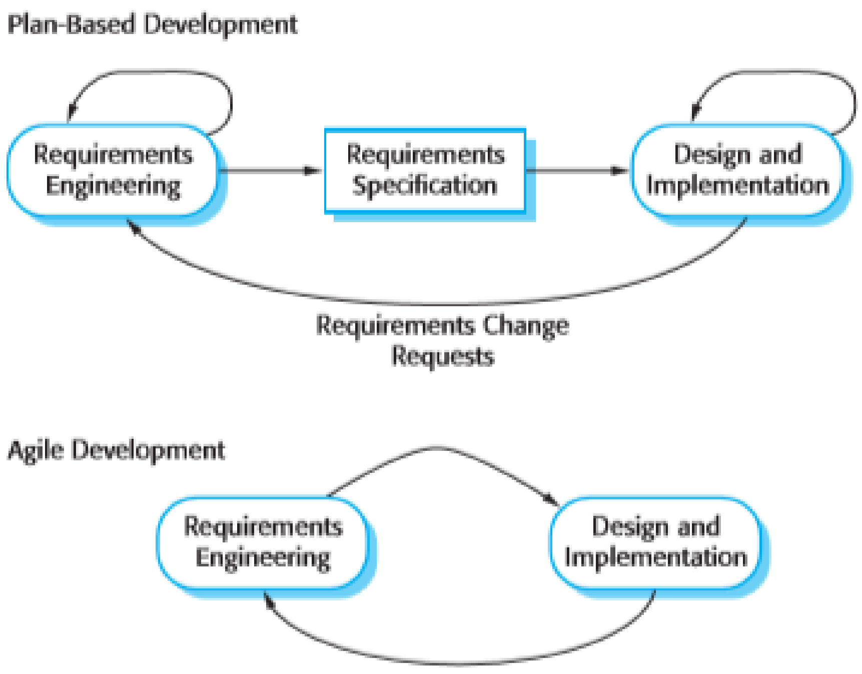


Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Plan-driven and agile development

Introduction

- Agile approaches to software development consider design and implementation to be the central activities in the software process.
- They incorporate other activities, such as requirements elicitation and testing, into design and implementation.
- By contrast, a plan-driven approach to software engineering identifies separate stages in the software process with outputs associated with each stage.


Plan-Based Development
Requirements Engineering
Requirements Specification
Design and Implementation
Requirements Change Requests
Agile Development
Requirements Engineering
Design and Implementation

A plan-driven software process can support incremental development and delivery. It is perfectly feasible to allocate requirements and plan the design and development phase as a series of increments.
An agile process is not inevitably code-focused and it may produce some design documentation.
The agile development team may decide to include a documentation 'spike', where, instead of producing a new version of a system, the team produce system documentation.

Extreme programming

Extreme programming (XP) is perhaps the best known and most widely used of the agile methods.

In XP, several new versions of a system may be developed by different programmers, integrated and tested in a day.

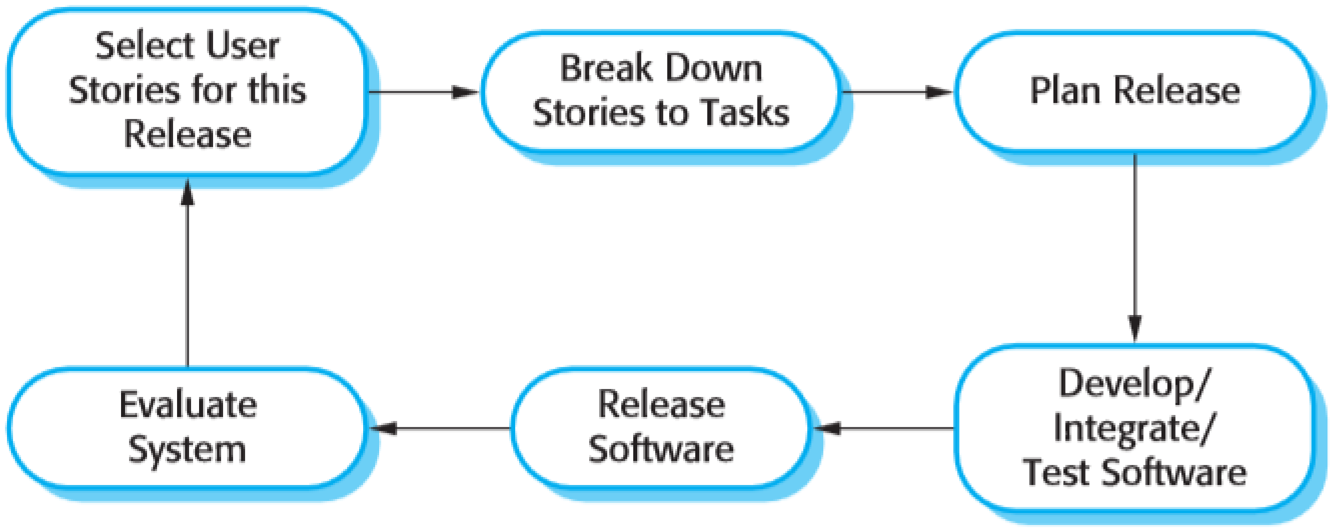

Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Figure illustrates the XP process to produce an increment of the system that is being developed.

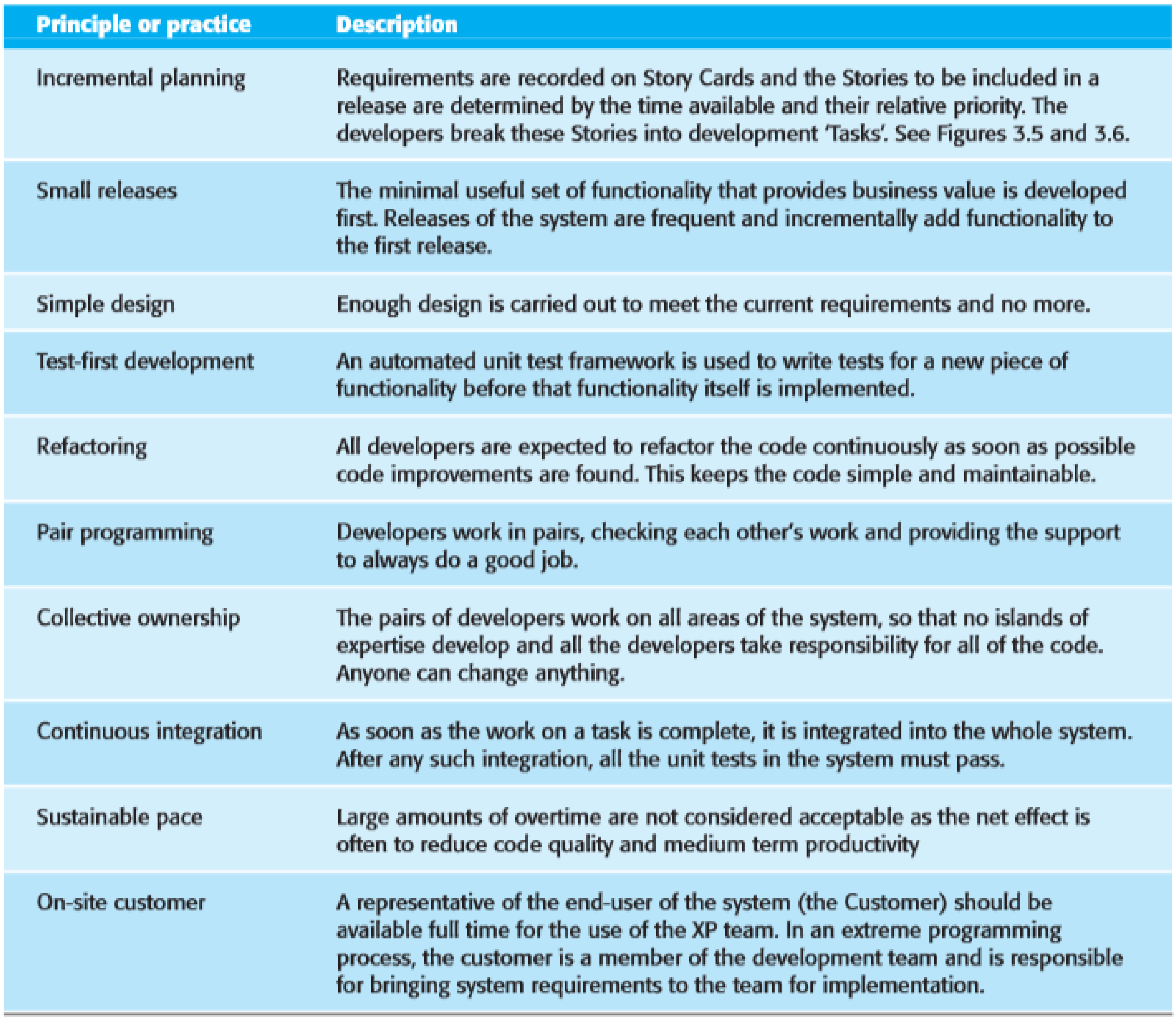
In extreme programming, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks.

Programmers work in pairs and develop tests for each task before writing the code.

All tests must be successfully executed when new code is integrated into the system.

There is a short time gap between releases of the system.

Practices/Principles of XP



Principle or practice	Description
Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Testing in XP

One of the important differences between incremental development and plan-driven development is in the way that the system is tested. With incremental development, there is no system specification that can be used by an external testing team to develop system tests. As a consequence, some approaches to incremental development have a very informal testing process, in comparison with plan-driven testing. To avoid some of the problems of testing and system validation, XP emphasizes the importance of program testing. XP includes an approach to testing that reduces the chances of introducing undiscovered errors into the current version of the system.

The key features of testing in XP are:

1. Test-first development

Instead of writing some code and then writing tests for that code, you write the tests before you write the code.

2. Incremental test development from scenarios

3. User involvement in the test development and validation, and

4. The use of automated testing frameworks.

Pair Programming

- Innovative practice that has been introduced in XP is that programmers work in pairs to develop the software
- They actually sit together at the same workstation to develop the software.
- The use of pair programming has a number of advantages:
    - I. It supports the idea of collective ownership and responsibility for the system.
    - II. It acts as an informal review process because each line of code is looked at by at least two people.
    - III. It helps support refactoring, which is a process of software improvement.

Agile project management

The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.

The standard approach to project management is plan-driven.

A plan-based approach really requires a manager to have a stable view of everything that has to be developed and the development processes.

However, it does not work well with agile methods where the requirements are developed incrementally; where the software is delivered in short, rapid increments; and where changes to the requirements and the software are the norm.

Scrum Approach for Agile Project Management

A general agile method but its focus is on managing iterative development rather than specific technical approaches to agile software engineering.

Scrum does not prescribe the use of programming practices such as pair programming and test-first development.

It can therefore be used with XP, to provide a management framework for the project.

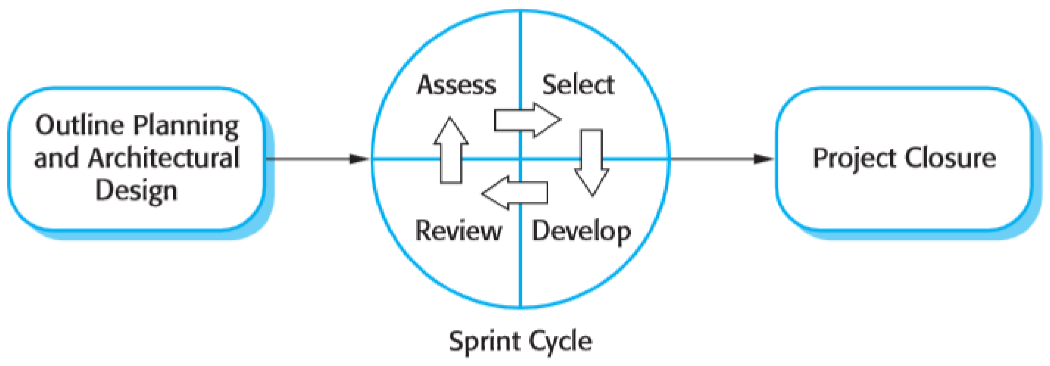


Figure 3.8 The Scrum process

Three phases in Scrum

I. **Outline planning phase →** where you establish the general objectives for the project and design the software architecture.
II. **Series of sprint cycles→** where each cycle develops an increment of the system.
III. **Project closure phase→** wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessons learned from the project.

Key characteristics of Scrum process are as follows:

I. Sprints are fixed length, normally 2–4 weeks. (Development of a release of the system in XP).
II. The starting point for planning is the product backlog, which is the list of work to be done on the project.
III. The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.
IV. Once these are agreed, the team organizes themselves to develop the software.
V. At the end of the sprint, the work done is reviewed and presented to stakeholders.
VI. The next sprint cycle then begins.

Advantages of scrum process in telecommunication software development environment:

1. The product is broken down into a set of manageable and understandable chunks.

2. Unstable requirements do not hold up progress.

3. The whole team has visibility of everything and consequently team communication is improved.

4. Customers see on-time delivery of increments and gain feedback on how the product works.

5. Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Scaling Agile Methods

Agile Methods:

→ Mostly used for the development of small and medium-sized systems.

Scaling Agile Methods:

→ Developed by large organizations to cope with larger systems

Large software system development is different from small system development in a number of ways:

1. Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones. It is practically impossible for each team to have a view of the whole system. Consequently, their priorities are usually to complete their part of the system without regard for wider systems issues.

2. Large systems are 'brownfield systems' (Hopkins and Jenkins, 2008); that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development. Political issues can also be significant here—often the easiest solution to a problem is to change an existing system. However, this requires negotiation with the managers of that system to convince them that the changes can be implemented without risk to the system's operation.

3. Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development. This is not necessarily compatible with incremental development and frequent system integration.

4. Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed, that require certain types of system documentation to be produced, etc.

5. Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.

6. Large systems usually have a diverse set of stakeholders. For example, nurses and administrators may be the end-users of a medical system but senior medical staff, hospital managers, etc. are also stakeholders in the system. It is practically impossible to involve all of these different stakeholders in the development process.

Two perspectives on the scaling of agile methods:

1. A 'scaling up' perspective, which is concerned with using these methods for developing large software systems that cannot be developed by a small team.
2. A 'scaling out' perspective, which is concerned with how agile methods can be introduced across a large organization with many years of software development experience.

Why is it difficult to introduce agile methods into large companies?

Fundamentals of agile methods—flexible planning, frequent system releases, continuous integration, test driven development, and good team communications.

It is difficult to introduce agile methods into large companies. Give Reasons:

1. Project managers who do not have experience of agile methods may be reluctant (unwillingness) to accept the risk of a new approach, as they do not know how this will affect their particular projects.

2. Large organizations often have quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods. Sometimes, these are supported by software tools (e.g., requirements management tools) and the use of these tools is mandated for all projects.

3. Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities, and people with lower skill levels may not be effective team members in agile processes.

4. There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.