"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individual and Interactions over Processes and Tools
Working Product over Comprehensive Documentation
Customer over Contract Negotiation
Responding to change over Following a plan

That is, while there is value in the items on the right, we value the items on the left more."
Some Agile Software Development Models

- Scrum
- Extreme programming (XP)
- Feature-driven development (FDD)
- Kanban
- Adaptive software development (ASD)
- Lean software development
- Rapid application development (RAD)
Agile development

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code
Plan-driven and agile development

Plan-based development

- Requirements engineering
- Requirements specification
- Design and implementation

Requirements change requests

Agile development

- Requirements engineering
- Design and implementation
Plan-driven and agile development

Plan-driven development:

- A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- Not necessarily waterfall model – plan-driven, incremental development is possible
- Iteration occurs within activities.

Agile development:

- Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.
Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:

- Focus on the code rather than the design
- Are based on an iterative approach to software development
- Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.
The principles of agile methods

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer involvement</td>
<td>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.</td>
</tr>
<tr>
<td>Incremental delivery</td>
<td>The software is developed in increments with the customer specifying the requirements to be included in each increment.</td>
</tr>
<tr>
<td>People not process</td>
<td>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.</td>
</tr>
<tr>
<td>Embrace change</td>
<td>Expect the system requirements to change and so design the system to accommodate these changes.</td>
</tr>
<tr>
<td>Maintain simplicity</td>
<td>Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.</td>
</tr>
</tbody>
</table>
Agile method applicability

- Product development where a software company is developing a small or medium-sized product for sale.
- 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 few external rules and regulations that affect the software.
Agile development techniques
A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques. Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.

- New versions may be built several times per day;
- Increments are delivered to customers every 2 weeks;
- All tests must be run for every build and the build is only accepted if tests run successfully.
The extreme programming release cycle

1. Select user stories for this release
2. Break down stories to tasks
3. Plan release
4. Develop/integrate/test software
5. Release software
6. Evaluate system
Agile development techniques

**XP practices**

- Small releases
- Simple design
- Test-first development
- Pair programming
- Continuous integration
- On-site customer
Incremental development is supported through small, frequent system releases.
Customer involvement means full-time customer engagement with the team.
People not process through pair programming, collective ownership and a process that avoids long working hours.
Change supported through regular system releases.
Maintaining simplicity through constant refactoring of code.
Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations. Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.

Key practices
- User stories for specification
- Refactoring
- Test-first development
- Pair programming
Test-first development

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.
Test-driven development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written so that they can be executed automatically
  - Usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.
Test automation means that tests are written as executable components before the task is implemented.

- These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.

As testing is automated, there is always a set of tests that can be quickly and easily executed.

- Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.
Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.

Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the ‘display logic’ and workflow between screens.

It is difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.
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. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.

Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods.
Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.

There are three phases in Scrum.

- The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
- This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
- The 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.
## Scrum terminology

<table>
<thead>
<tr>
<th>Scrum term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development team</td>
<td>A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.</td>
</tr>
<tr>
<td>Potentially shippable product increment</td>
<td>The software increment that is delivered from a sprint. The idea is that this should be ‘potentially shippable’ which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.</td>
</tr>
<tr>
<td>Product backlog</td>
<td>This is a list of ‘to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.</td>
</tr>
<tr>
<td>Product owner</td>
<td>An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.</td>
</tr>
</tbody>
</table>
### Scrum terminology

<table>
<thead>
<tr>
<th>Scrum term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrum</td>
<td>A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.</td>
</tr>
<tr>
<td>ScrumMaster</td>
<td>The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.</td>
</tr>
<tr>
<td>Sprint</td>
<td>A development iteration. Sprints are usually 2-4 weeks long.</td>
</tr>
<tr>
<td>Velocity</td>
<td>An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team’s velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.</td>
</tr>
</tbody>
</table>
Agile project management

Scrum sprint cycle

- Review work to be done
- Select items
- Plan sprint
- Sprint
- Review sprint

- Product backlog
- Sprint backlog
- Potentially shippable software

Scrum
The Scrum sprint cycle

- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.
Once these are agreed, the team organize themselves to develop the software.

During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.

The role of the Scrum master is to protect the development team from external distractions.

At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.
The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.

The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.

This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.
Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.
Scaling agile methods
Scaling agile methods

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.
‘Scaling up’ is concerned with using agile methods for developing large software systems that cannot be developed by a small team.

‘Scaling out’ is concerned with how agile methods can be introduced across a large organization with many years of software development experience.

When scaling agile methods it is important to maintain agile fundamentals:

- Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.
The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.

Agile methods are most appropriate for new software development rather than software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems.

Agile methods are designed for small co-located teams yet much software development now involves worldwide distributed teams.
Contractual issues

- Most software contracts for custom systems are based around a specification, which sets out what has to be implemented by the system developer for the system customer.
- However, this precludes interleaving specification and development as is the norm in agile development.
- A contract that pays for developer time rather than functionality is required.
  - However, this is seen as a high risk by many legal departments because what has to be delivered cannot be guaranteed.
Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.

Two key issues:

- Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
- Can agile methods be used effectively for evolving a system in response to customer change requests?

Problems may arise if original development team cannot be maintained.
Key problems are:
- Lack of product documentation
- Keeping customers involved in the development process
- Maintaining the continuity of the development team

Agile development relies on the development team knowing and understanding what has to be done.

For long-lifetime systems, this is a real problem as the original developers will not always work on the system.
Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:

- Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
- Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
- How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.
System issues

- How large is the system being developed?
  - Agile methods are most effective a relatively small co-located team who can communicate informally.

- What type of system is being developed?
  - Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this analysis.

- What is the expected system lifetime?
  - Long-lifetime systems require documentation to communicate the intentions of the system developers to the support team.

- Is the system subject to external regulation?
  - If a system is regulated you will probably be required to produce detailed documentation as part of the system safety case.
People and teams

- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code.

- How is the development team organized?
  - Design documents may be required if the team is distributed.

- What support technologies are available?
  - IDE support for visualisation and program analysis is essential if design documentation is not available.
Organizational issues

- Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- Is it standard organizational practice to develop a detailed system specification?
- Will customer representatives be available to provide feedback of system increments?
- Can informal agile development fit into the organizational culture of detailed documentation?
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.

Large systems are ‘brownfield systems’, 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.

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.
Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.

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.

Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.
A completely incremental approach to requirements engineering is impossible.

There cannot be a single product owner or customer representative.

For large systems development, it is not possible to focus only on the code of the system.

Cross-team communication mechanisms have to be designed and used.

Continuous integration is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system.
Multi-team Scrum

- **Role replication**
  - Each team has a Product Owner for their work component and ScrumMaster.

- **Product architects**
  - Each team chooses a product architect and these architects collaborate to design and evolve the overall system architecture.

- **Release alignment**
  - The dates of product releases from each team are aligned so that a demonstrable and complete system is produced.

- **Scrum of Scrums**
  - There is a daily Scrum of Scrums where representatives from each team meet to discuss progress and plan work to be done.
Scaling agile methods

Agile methods across organizations

- Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach.
- 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.
- 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.
- There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.
Scaling agile methods

Key points

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code.
- Agile development practices include:
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.
Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.

Many practical development methods are a mixture of plan-based and agile development.

Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.