


MCSE 652



Software Quality Assurance

Chapter 3: Agile Software Process

Dr. Mehedi Hasan
Stamford University Bangladesh

Outline



- ▶ Agile Methodology
 - ▶ Concepts and ideology
 - ▶ Origins
 - ▶ Applicable domains
 - ▶ Problems with Agile process
- ▶ Plan-driven vs. agile development
- ▶ Agile Process models
 - ▶ Extreme programming
 - ▶ Feature-Driven Development

What is Agile Software Development?



- ▶ Easily moved, quickness, lightness, adaptable, flexible, active software processes
- ▶ Fitting the process to the project
- ▶ Avoidance of things that waste time

- ▶ “Agile” –ease of movement
An agile mind –mentally quick and alert

Why Agility?



- ▶ **Rapid development and delivery** is now often the most important requirement for software systems
 - ▶ Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
 - ▶ Software has to evolve quickly to reflect changing business needs.
- ▶ **Rapid software development**
 - ▶ Specification, design and implementation are inter-leaved
 - ▶ System is developed as a series of versions with stakeholders involved in version evaluation
 - ▶ User interfaces are often developed using an IDE and graphical toolset.

Agile manifesto



- ▶ *We are uncovering better ways of developing ^[SEP]software by doing it and helping others do it. ^[SEP]Through this work we have come to value:*
 - ▶ ***Individuals and interactions** over processes and tools*
 - Working software** over comprehensive documentation*
 - Customer collaboration** 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.*

Agile methods



- ▶ 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

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.

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 not a lot of external rules and regulations that affect the software.
- ▶ Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

Problems with agile methods



- ▶ It can be difficult to keep the interest of customers who are involved in the process.
- ▶ Team members may be unsuited to the intense involvement that characterizes agile methods.
- ▶ Prioritizing changes can be difficult where there are multiple stakeholders.
- ▶ Maintaining simplicity requires extra work.
- ▶ Contracts may be a problem as with other approaches to iterative development.

Agile

vs.

Plan Driven

1. Small products and teams; scalability limited
2. Untested on safety-critical products
3. Good for dynamic, but expensive for stable environments.
4. Require experienced Agile personnel throughout
5. Personnel thrive on freedom and chaos

1. Large products and teams; hard to scale down
2. Handles highly critical products; hard to scale down
3. Good for stable, but expensive for dynamic environments
4. Require experienced personnel only at start if stable environment
5. Personnel thrive on structure and order

Plan-driven and agile development



- ▶ Plan-driven development

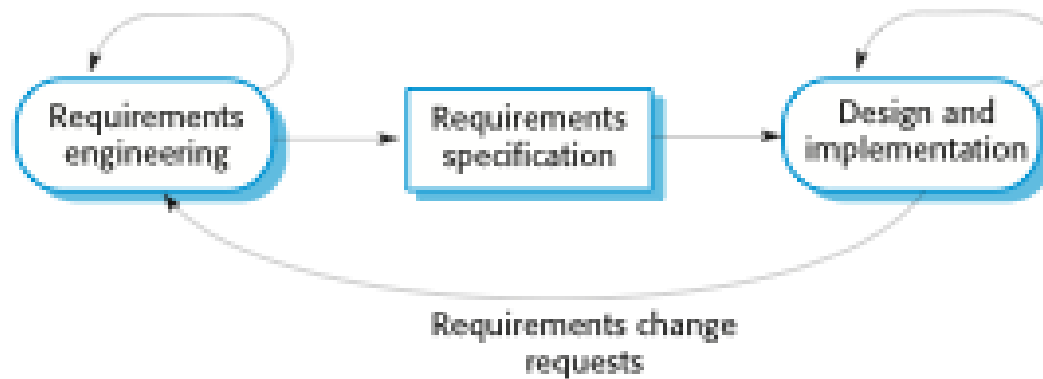
- ▶ A plan-driven approach to software engineering is based around **separate development stages with the outputs** to be produced at each of these **stages planned in advance**.
- ▶ Not necessarily waterfall model – plan-driven, incremental development is possible
- ▶ Iteration occurs within activities.

- ▶ Agile development

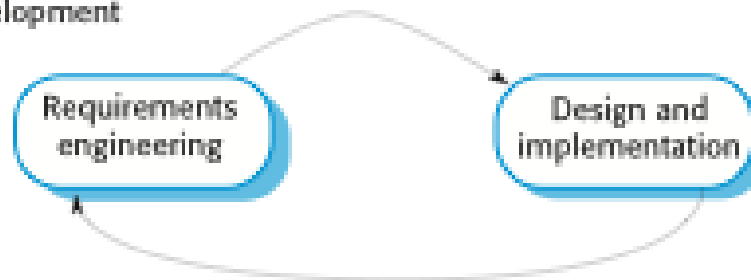
- ▶ **Specification, design, implementation and testing** are interleaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Plan-driven and agile specification

Plan-based development



Agile development



Technical, human, organizational issues



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.

Technical, human, organizational issues



- ▶ **What type of system is being developed?**

- Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).

- ▶ **What is the expected system lifetime?**

- Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.

- ▶ **What technologies are available to support system development?**

- Agile methods rely on good tools to keep track of an evolving design

- ▶ **How is the development team organized?**

- If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

Technical, human, organizational issues



- ▶ **Are there cultural or organizational issues that may affect the system development?**
 - Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- ▶ **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
- ▶ **Is the system subject to external regulation?**
 - If a system has to be approved by an external regulator (e.g. A software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

Agile Process models



- 1. Extreme Programming (XP)**
- 2. Feature Driven Development (FDD)**
- 3. Others: Scrum, Crystal, Adaptive Software Development**

Extreme programming (XP)



- ▶ Perhaps the best-known and most widely used agile method.

"Extreme Programming turns the conventional software process sideways. Rather than planning, analyzing, and designing for the far-flung future, XP programmers do all of these activities a little at a time throughout development."

-- *IEEE Computer* , October 1999

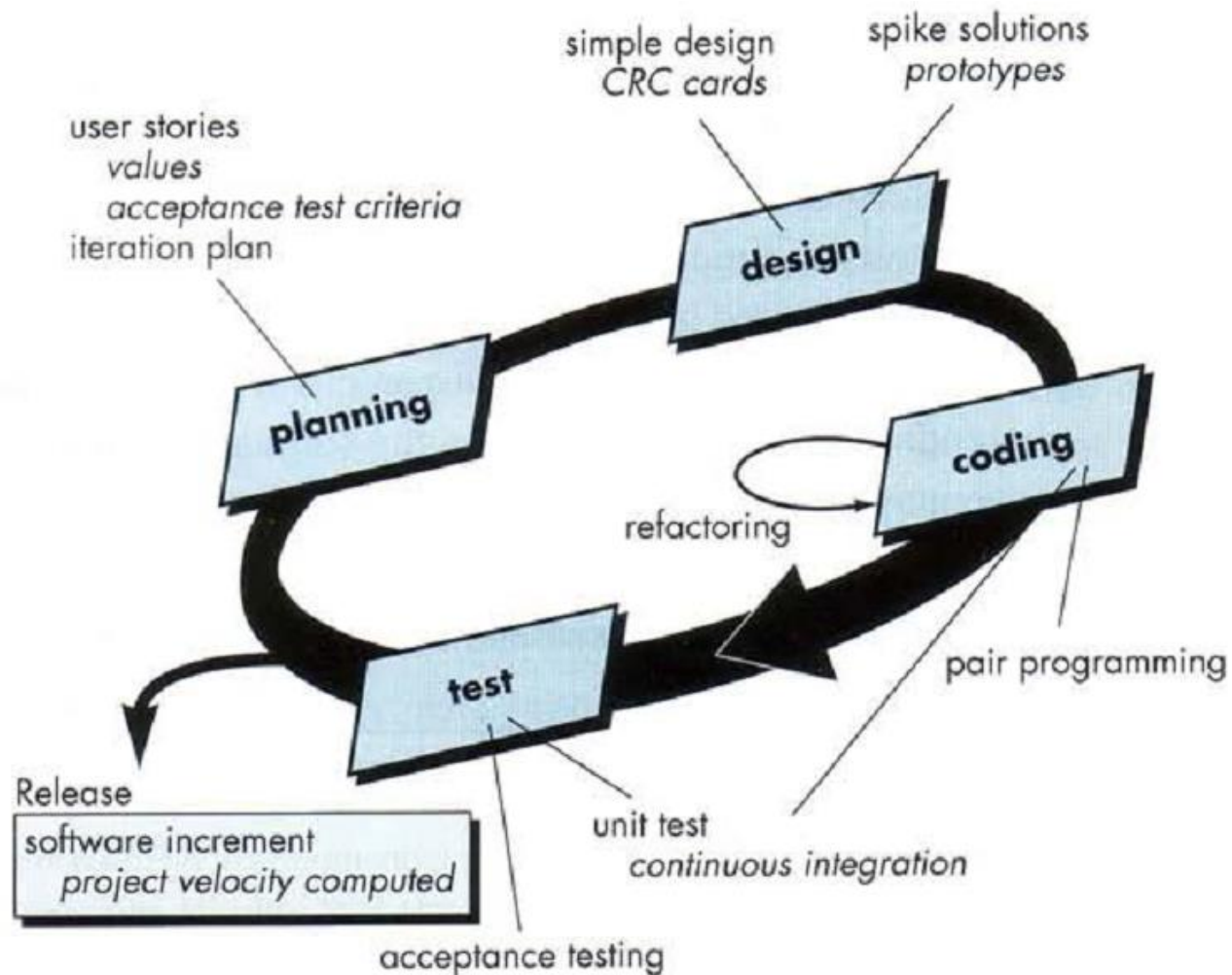
- ▶ Extreme Programming (XP) takes an 'extreme' approach to iterative development.
 - ▶ New versions may be built several times per day;
 - ▶ Increments are delivered to customers every 2 weeks;
 - ▶ All tests must be run for every build and the build is only accepted if tests run successfully.

XP and agile principles

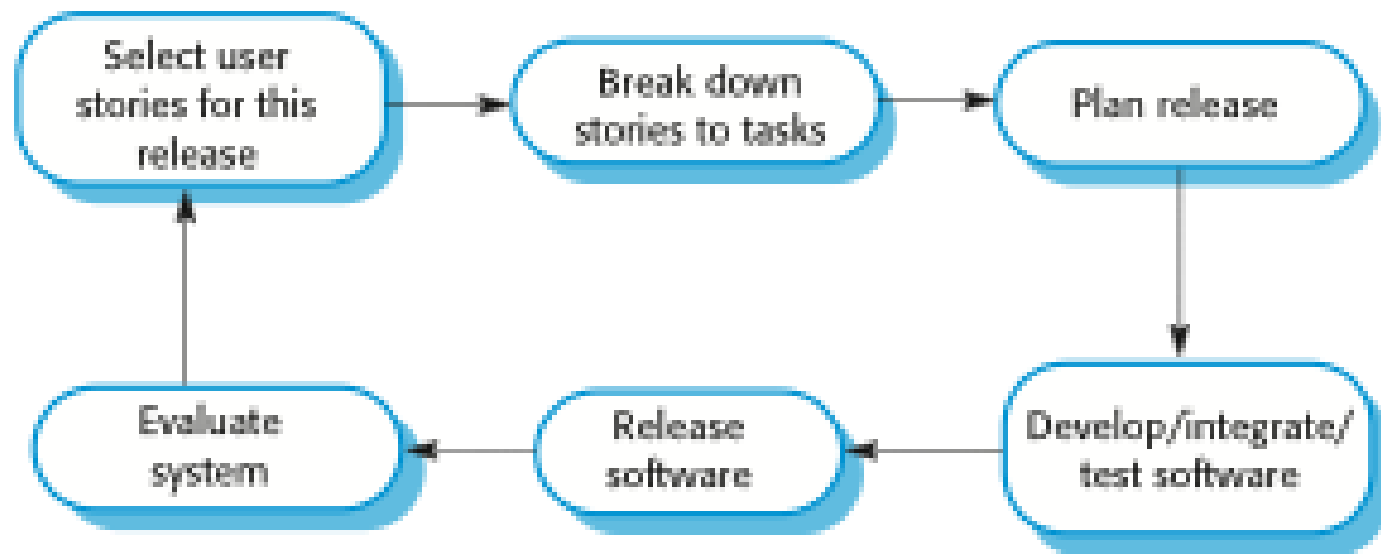


- ▶ Incremental development is **supported through small, frequent system releases.**
- ▶ Customer involvement means **full-time customer engagement** with the team.
- ▶ People-oriented through **pair programming and collective ownership.**
- ▶ Change supported through **regular system releases.**
- ▶ Maintaining simplicity through **constant refactoring of code.**

The extreme programming process



The extreme programming release cycle



Extreme programming practices (a)

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'.
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.

Extreme programming practices (b)

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.

Requirements scenarios



- ▶ In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- ▶ User requirements are expressed as scenarios or user stories.
- ▶ These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- ▶ The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

Pair programming



- ▶ In pair programming, programmers sit together at the same workstation to develop the software.
- ▶ Pairs are created dynamically so that all team members work with each other during the development process.
- ▶ The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- ▶ Pair programming, through research, is found to increase productivity.

Refactoring



- ▶ Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- ▶ This improves the understandability of the software and so reduces the need for documentation.
- ▶ Changes are easier to make because the code is well-structured and clear.
- ▶ However, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring



- ▶ Re-organization of a class hierarchy to remove duplicate code.
- ▶ Tidying up and renaming attributes and methods to make them easier to understand.
- ▶ The replacement of inline code with calls to methods that have been included in a program library.
- ▶ Breaking up a long method into smaller methods

Testing



▶ Principles

- ▶ All code must have tests, with the test created first
- ▶ Unit tests are executed daily during the automated build
- ▶ When a bug is found, new tests are created
- ▶ All unit tests must pass before code can be released

▶ Benefits

- ▶ Provides constant visibility into areas of the system at risk
- ▶ The result is rapid progress and a system that always works better than it did the day before

Test-first development




- ▶ Writing tests before code clarifies the requirements to be implemented.
- ▶ Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
 - ▶ 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.

Customer involvement in Testing



- ▶ The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- ▶ The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- ▶ However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

Why XP is “extreme”?



Commonsense practices taken to extreme levels

- ▶ *Code reviews are done all the time (pair programming)*
- ▶ *Testing is done all the time (unit testing and acceptance testing)*
- ▶ *KIS (Keep it simple) design supports its current functionality.
(simplest thing that works)*
- ▶ *If design is refined consistently to include new changes daily
(refactoring)*
- ▶ *Integration testing is done and tested several times a day
(continuous integration)*
- ▶ *Short iterations of work accomplished is practiced.
(hours or days rather than weeks)*

2. Feature Driven Development



- ▶ Feature Driven Development focuses on regular delivery of client-valued features
- ▶ More structured than XP (has more formal steps and more precise tracking of progress)
- ▶ List of features presents a value to customers as it reflects the functionality that will be available in a software application

Core values of FDD



- ▶ A system for building systems is necessary
- ▶ Simple is better
- ▶ Process steps should be obviously valuable to each team member
- ▶ Discovering list of features to implement and follow feature-by-feature implementation.

Six Roles



- ▶ Project Manager
- ▶ Chief Architect
- ▶ Development Manager
- ▶ Chief Programmers
- ▶ Class Owners (aka Developers)
- ▶ Domain Experts

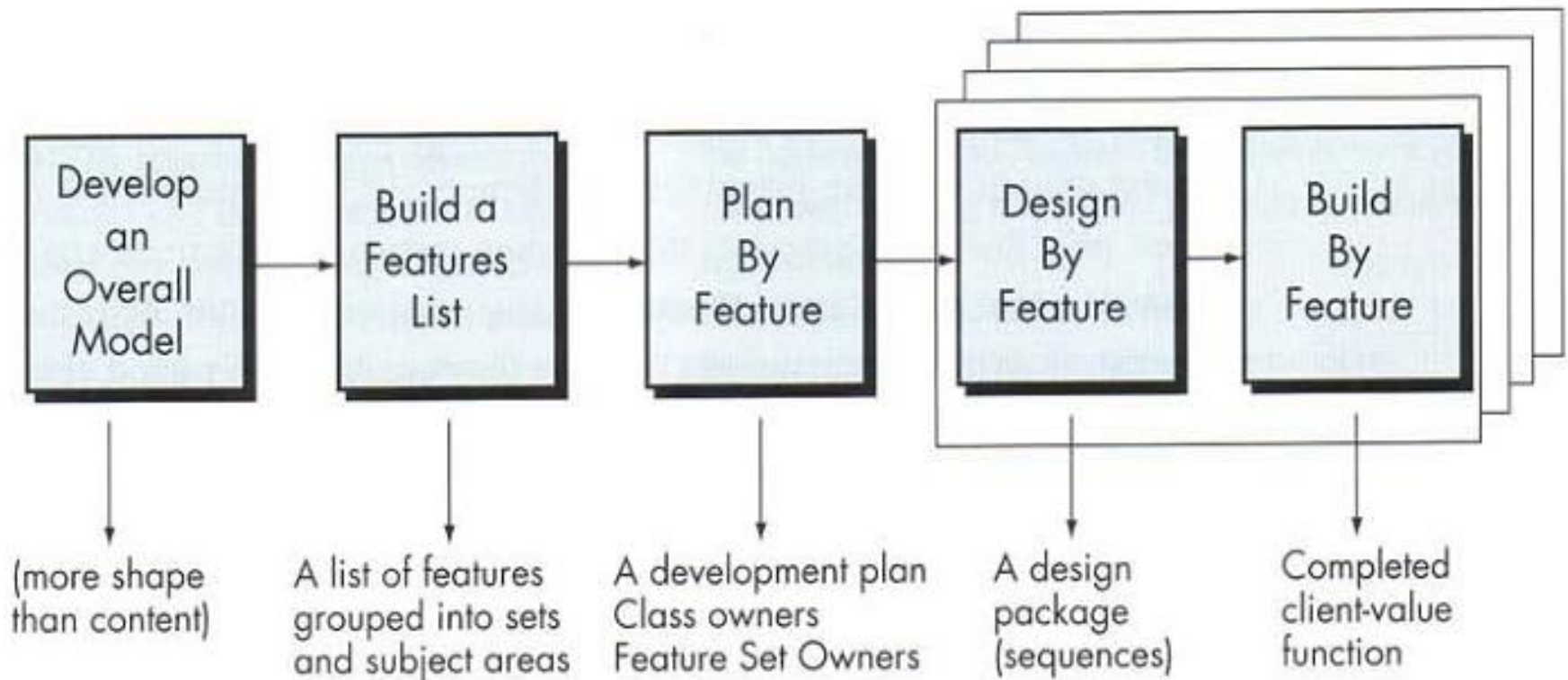
2. Feature Driven Design (FDD)



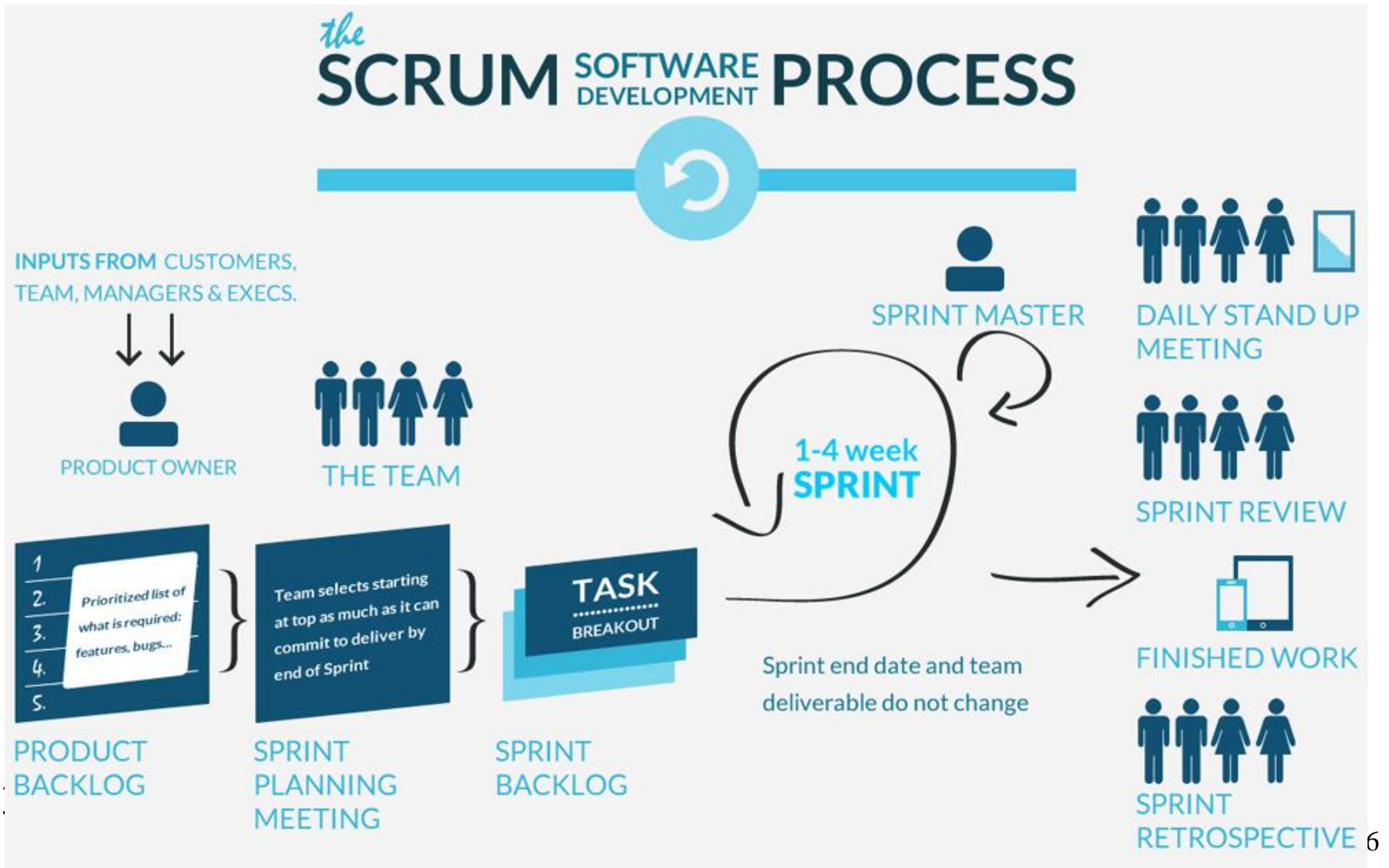
Five FDD process activities

1. Develop an overall model – Produce class and sequence diagrams from chief architect meeting with domain experts and developers.
2. Build a features list – Identify all the features that support requirements.
Features are functions that can be developed in two weeks and expressed in client terms
 1. i.e. Calculate the total of a sale
3. Plan by feature -- the development staff plans the development sequence of features
4. Design by feature -- the team produces sequence diagrams for the selected features
5. Build by feature – the team writes and tests the code

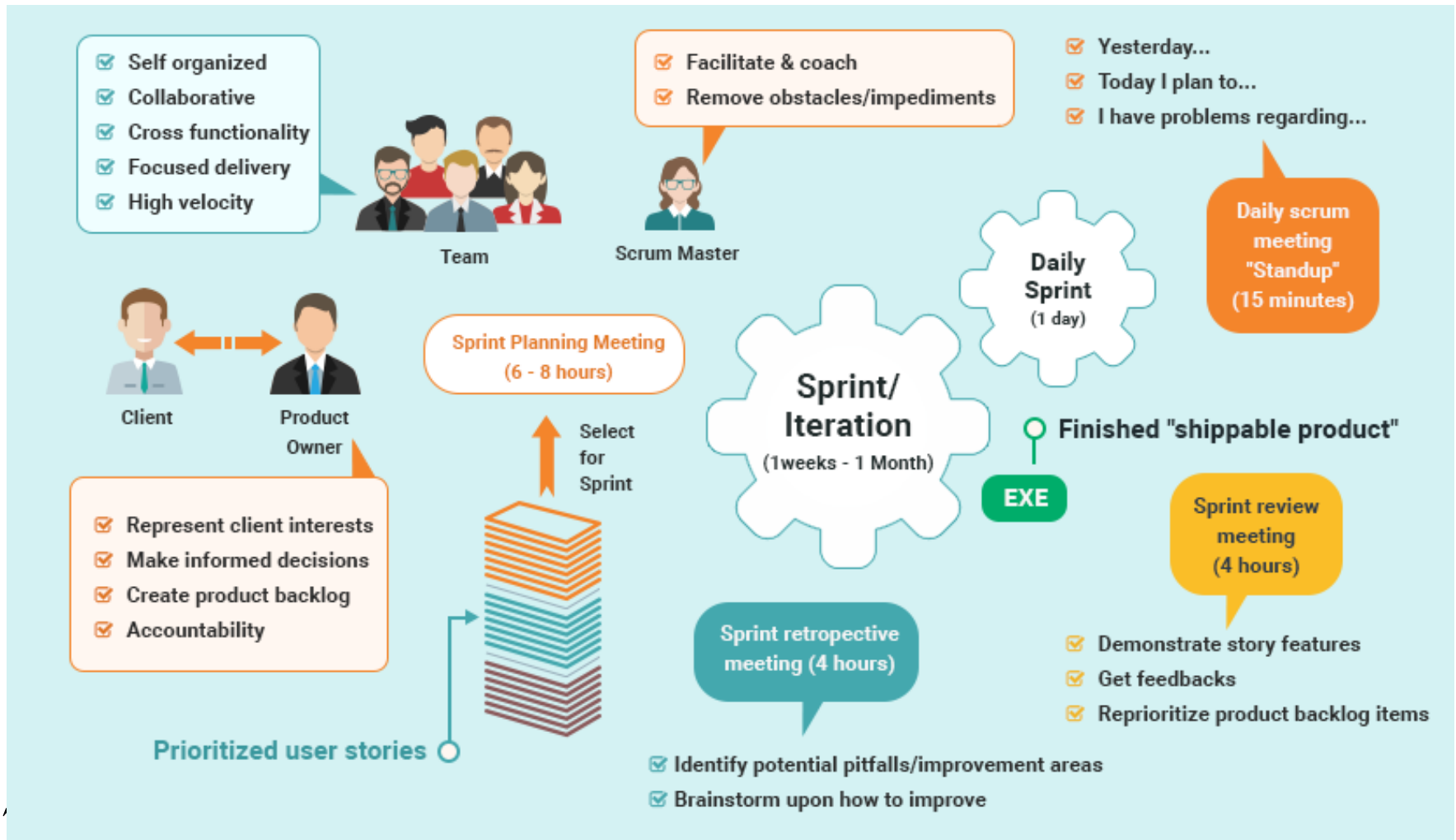
2. Feature Driven Development (FDD)



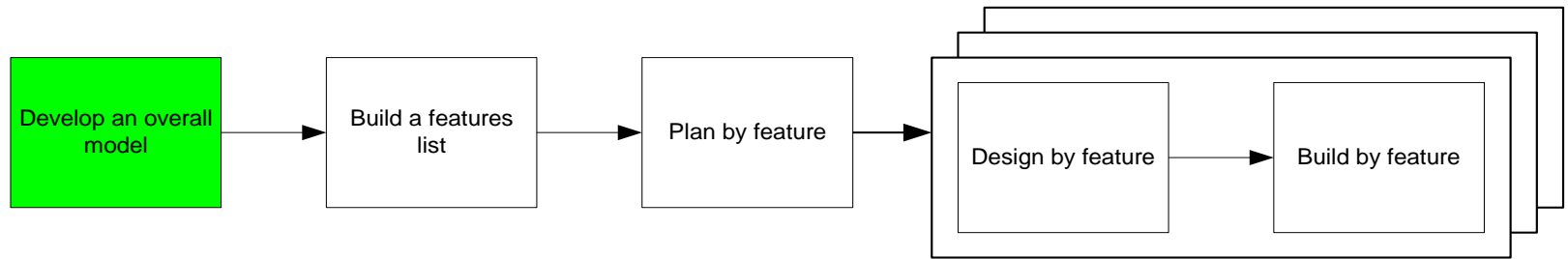
Modern Approach



SCRUM



1. Develop an overall model



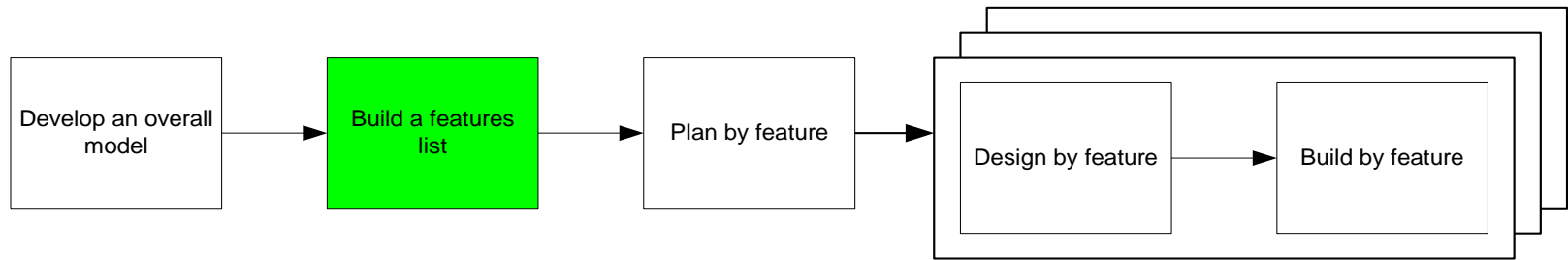
Who?

domain experts, chief architect, chief programmers

Activity:

Establishes the shape of the system, creates base object model (UML), includes review and model notes

2. Build a features list



Who? Feature List Team:
domain experts, chief programmers, chief architect

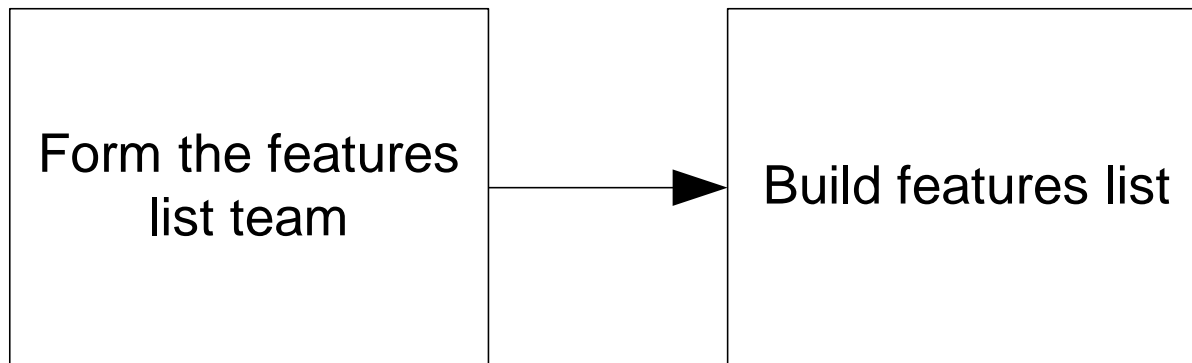
Activity:
Functional decomposition of model from step 1
Build feature list, feature is a business activity step; customer centric than technology centric

2. Build a features list

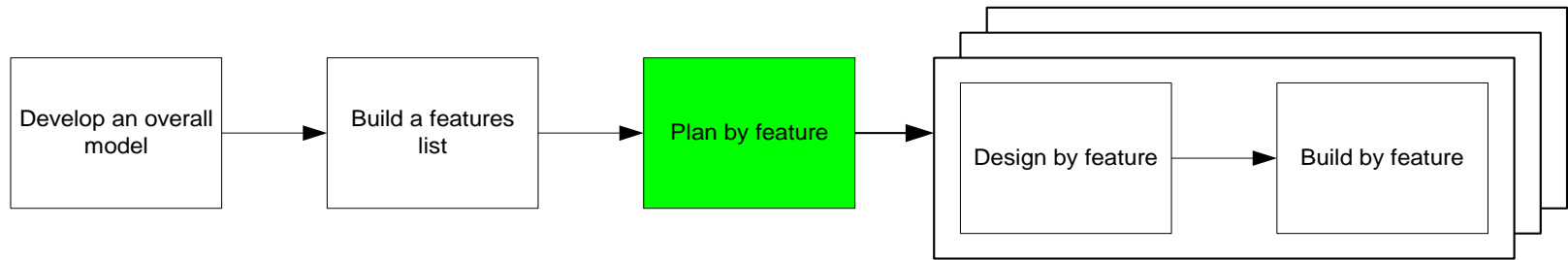


- ▶ Functional decomposition of model developed in step 1
- ▶ Subject area to business activity to business activity step
- ▶ Feature is a business activity step, customer centric not technology centric
- ▶ Nomenclature: <action> <result> <object>
- ▶ “Generate an account number for the new customer”

2. Build a features list



3. Plan By Feature



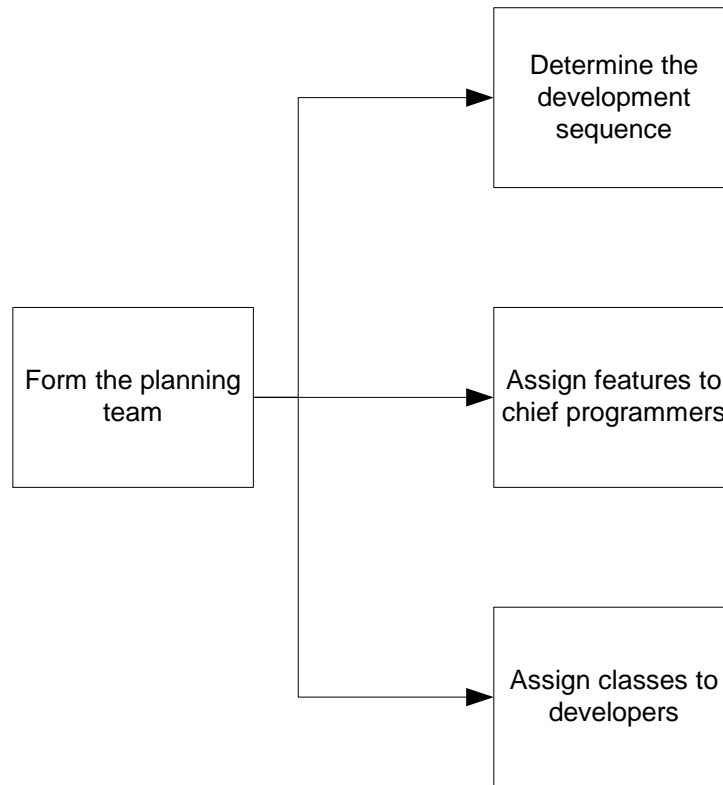
Who? The Planning Team:

the project manager, the development manager, and chief programmers.

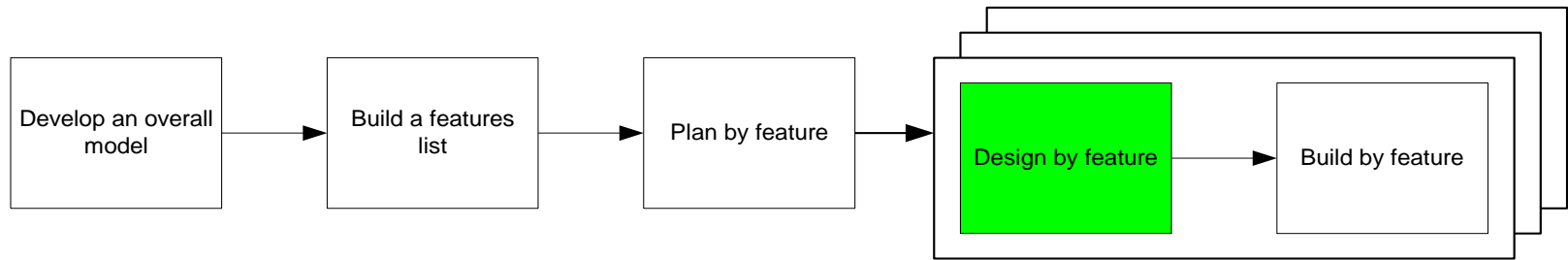
Activity:

- ▶ Group features into feature sets (one or more business activities)
- ▶ Prioritize based on customer need
- ▶ Establish completion dates (MM/YYYY)

3. Plan By Feature



4. Design by feature



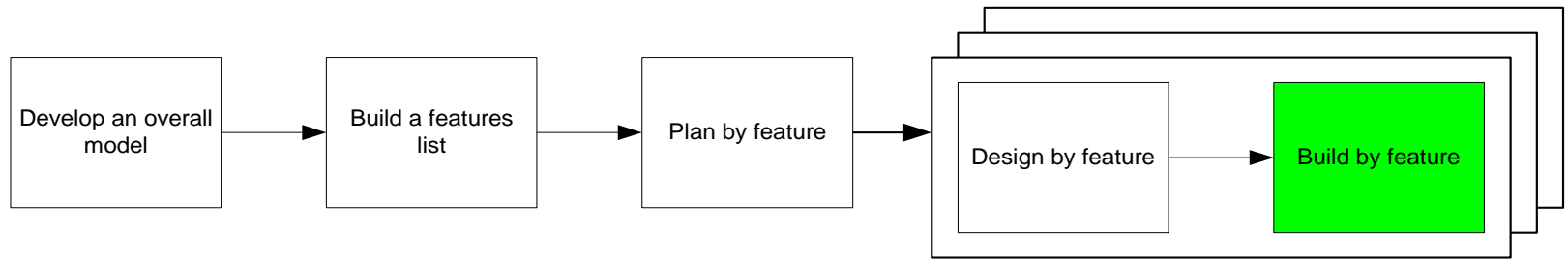
Who?

The Feature Team: chief programmer, class owners

Activity:

- ▶ Work package level—now based on the technical architecture
- ▶ Two weeks or less of work
- ▶ Fleshes out class and object design, create sequence diagrams as necessary
- ▶ Updates object model created in process #1.

5. Develop by feature



Who?

Class owners, chief programmers

Activity:

- ▶ Implement
- ▶ Code inspection
- ▶ Unit test
- ▶ Promote to build

Market Position: FDD v XP



FDD

- ▶ More hierarchical
- ▶ Class owners
- ▶ Success with above average developers
- ▶ Client works on 1,2,4
- ▶ Process 1
- ▶ “Live the life”!
- ▶ Medium-sized

XP

- ▶ Peer to peer
- ▶ Collective ownership
- ▶ Success with average developers
- ▶ Client on the team
- ▶ Constant refactoring
- ▶ 40 hour weeks
- ▶ Small-sized

Key points



- ▶ Agile methods embrace change and are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- ▶ The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- ▶ Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

Key points



- ▶ A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- ▶ FDD development consists of the two main stages: Discovering list of features to implement; Feature-by-feature implementation; Discovering list of features is a critical process as the quality of this step largely defines how precise the project will be tracked, how maintainable and extensible the code will be. This process requires full-time participation of customers.
- ▶ Scaling agile methods for large systems is difficult. Large systems need up-front design and documentation.