CSC 540 2.0
Software Engineering

M. K. A. Ariyaratne, PhD.

Department of Computer Science
Faculty of Applied Sciences
University of Sri Jayewardenepura
Sri Lanka

UNIT 4
Agile Software Development

Outline

1. Introduction to Agile Software Development
2. Agile Principles and Manifesto
3. Agile Methodologies (Scrum, XP, etc.)
4. Agile Practices
5. Challenges and Benefits of Agile
6. Hybrid Models

Agile Software Development - Introduction

- Rapid development and delivery is the most critical requirement for software systems.
- It is often practically impossible to derive a complete set of stable software requirements
- Software development processes that plan on completely specifying the requirements and then designing, building, and testing the system are not geared to rapid software development.
- Therefore conventional methods like waterfall method are not suitable for most cases.

Agile Software Development - Introduction

- The need for rapid system development and processes that can handle changing requirements has been recognized for some time.
- Rapid software development processes are designed to produce useful software quickly.
- Although there are many approaches to rapid software development, they share some fundamental characteristics:
  1. Specification, design and implementation are inter-leaved
  2. System is developed as a series of versions with stakeholders involved in version evaluation
  3. User interfaces are often developed using an IDE and graphical tool set.

Agile Software Development - Introduction

- Agile methods are incremental development methods in which the increments are small and, typically, new releases of the system are created and made available to customers every two or three weeks.
- Customers are involved in the development process to get rapid feedback on changing requirements.
- Minimize documentation by using informal communications rather than formal meetings with written documents.

Agile 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.

Agile manifesto

- The philosophy behind agile methods is reflected in the agile manifesto that was agreed on by many of the leading developers of these methods.

  This manifesto states:

- We are uncovering better ways of developing. software by doing it and helping others do it. Through this work we have come to value:
  1. Individuals and interactions over processes and tools.
  2. Working software over comprehensive documentation.
  3. Customer collaboration over contract negotiation.
  4. 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.

12 Principles of Agile

- Customer satisfaction through early and continuous delivery.
- Welcome changing requirements, even late in development.
- Deliver working software frequently.
- Close collaboration between developers and business people.
- Build projects around motivated individuals.
- Promote sustainable development.
- Continuous attention to technical excellence.
- Simplicity is essential.
- Self-organizing teams produce the best designs.
- Regular reflection for team improvement.

Popular Agile Methodologies

- Scrum: Iterative development with fixed-length sprints.
- Extreme Programming (XP): Focus on technical practices like pair programming and test-driven development.
- Kanban: Visual workflow management.
- Lean Development: Reducing waste and increasing value.
- Feature-Driven Development (FDD): Feature-based delivery.

Agile Methods - Appropriate for:

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

- Product development where a software company is developing a small or medium-sized product for sale.
- 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.

Agile Methods - Problems

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the speedy involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Some companies find it difficult to move to a working model where processes are informal and defined by development teams.
- The use of agile methods is likely to make subsequent system maintenance more difficult and expensive.

Agile Methods - Problems

- Agile methods rely on team members understanding aspects of the system with very few documentation. If an agile development team is broken up, then this implicit knowledge is lost and it is difficult for new team members to build up the same understanding of the system and its components.

Best way is a hybrid approach where agile methods incorporate some techniques from plan-driven development.

Scrum Framework

- Roles: Product Owner, Scrum Master, Development Team.
- Artifacts: Product Backlog, Sprint Backlog, Increment.
- Ceremonies:
  - Sprint Planning
  - Daily Standup
  - Sprint Review
  - Sprint Retrospective

What is a Sprint?

Definition: A time-boxed period in Agile development during which specific work is completed and made ready for review.

- **Duration:** Typically 1–4 weeks.
- **Purpose:** Deliver a potentially shippable product increment.
- **Key Characteristics:**
  - Fixed duration.
  - Defined sprint goal.
  - Incremental progress toward the overall product goal.
- **Activities During a Sprint:**
  - Sprint Planning.
  - Daily Standups.
  - Development and testing of features.
  - Sprint Review.
  - Sprint Retrospective.

Sprint Planning

Objective: Define the work to be completed during the sprint.

- **Participants:** Product Owner, Scrum Master, Development Team.
- **Key Activities:**
  - Discuss the sprint goal.
  - Select user stories from the backlog.
  - Break down stories into tasks and estimate effort.
- **Outcome:** A sprint backlog and a clear sprint goal.

Daily Standup

Objective: Synchronize the team's work and address any impediments.

- Held daily, usually 15 minutes.
- Each team member answers:
  - What did I accomplish yesterday?
  - What will I work on today?
  - Are there any blockers?
- **Outcome:** Improved team coordination and early identification of issues.

Sprint Review

Objective: Demonstrate and inspect the completed work.

- **Participants:** Scrum Team, Stakeholders.
- **Key Activities:**
    - Team showcases work completed during the sprint.
    - Stakeholders provide feedback.
- **Outcome:** Validation of the product increment and adjustments to the product backlog.

Sprint Retrospective

Objective: Reflect on the sprint and identify ways to improve.

- **Participants:** Scrum Team.
- **Key Activities:**
  - Discuss what went well, what didn't, and what can be improved.
  - Identify actionable improvements for the next sprint.
- **Outcome:** Continuous improvement in team processes and collaboration.

Extreme Programming (XP)

- Extreme programming (XP) is perhaps the best known and most widely used of the agile methods.
- The approach was developed by pushing recognized good practice, such as iterative development, to 'extreme' levels.
- For example, in XP, several new versions of a system may be developed by different programmers, integrated and tested in a day.
- Here, the requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks.

Extreme Programming

- 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.

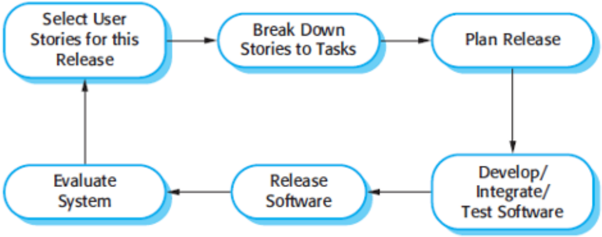
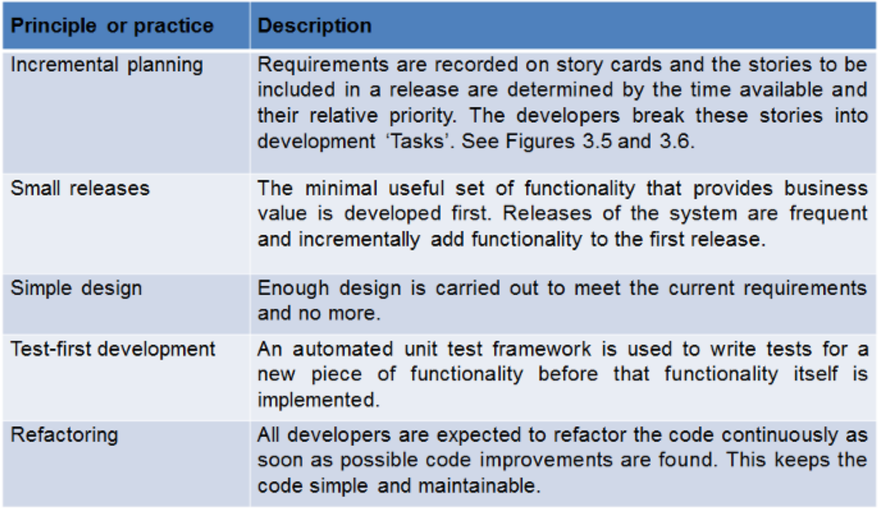

Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Figure: Extreme Programming - Release cycle

Extreme Programming and Agile principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People work 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 practices



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.

Extreme Programming practices (2)

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.

Extreme Programming requirement 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.

Extreme Programming and Change

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be consistent rather probable.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented

Extreme Programming and 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

Extreme Programming and Refactoring - Examples

- 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 in-line code with calls to methods that have been included in a program library.

Extreme Programming and Pair Programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

Extreme Programming and Pair Programming (2)

- 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 is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

Pair Programming - Advantages

- It supports the idea of collective ownership and responsibility for the system.
    - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
    - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

Core Agile Practices

- **User Stories:** Short descriptions of features from the user's perspective.
- **Backlog Grooming:** Refining and prioritizing tasks.
- **Daily Standups:** Short meetings for status updates.
- **Sprint Reviews:** Demonstrating completed work.
- **Retrospectives:** Reflecting on process improvements.

Benefits of Agile

- Faster time-to-market.
- Better collaboration and communication.
- Improved product quality.
- Increased flexibility to handle changes.

Challenges of Agile

- Requires cultural change.
- Difficulties in scaling for large teams.
- Customer involvement may wane.
- Maintaining simplicity requires discipline.

Agile and Plan-Driven Hybrid Approaches

- Combining Agile's flexibility with Plan-Driven's structure.
- Example: Using Agile for development but Plan-Driven for regulatory documentation.
- Suitable for complex, large-scale projects.

Agile Software Development - Key points

- Agile methods 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.