Software Engineering

Agile Software Engineering: Agile methods, Scrum, and Extreme Programming

Min-Yuh Day, Ph.D, Associate Professor

Institute of Information Management, National Taipei University

https://web.ntpu.edu.tw/~myday

1102SE03
MBA, IM, NTPU (M5010) (Spring 2022)
Wed 2, 3, 4 (9:10-12:00) (B8F40)
<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Subject/Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2022/02/23</td>
<td>Introduction to Software Engineering</td>
</tr>
<tr>
<td>2</td>
<td>2022/03/02</td>
<td>Software Products and Project Management: Software product management and prototyping</td>
</tr>
<tr>
<td>3</td>
<td>2022/03/09</td>
<td>Agile Software Engineering: Agile methods, Scrum, and Extreme Programming</td>
</tr>
<tr>
<td>4</td>
<td>2022/03/16</td>
<td>Features, Scenarios, and Stories</td>
</tr>
<tr>
<td>5</td>
<td>2022/03/23</td>
<td>Case Study on Software Engineering I</td>
</tr>
<tr>
<td>6</td>
<td>2022/03/30</td>
<td>Software Architecture: Architectural design, System decomposition, and Distribution architecture</td>
</tr>
<tr>
<td>Week</td>
<td>Date</td>
<td>Subject/Topics</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>2022/04/06</td>
<td>Make-up holiday (No Classes)</td>
</tr>
<tr>
<td>8</td>
<td>2022/04/13</td>
<td>Midterm Project Report</td>
</tr>
<tr>
<td>9</td>
<td>2022/04/20</td>
<td>Cloud-Based Software: Virtualization and containers, Everything as a service, Software as a service</td>
</tr>
<tr>
<td>10</td>
<td>2022/04/27</td>
<td>Cloud Computing and Cloud Software Architecture</td>
</tr>
<tr>
<td>11</td>
<td>2022/05/04</td>
<td>Microservices Architecture, RESTful services, Service deployment</td>
</tr>
<tr>
<td>12</td>
<td>2022/05/11</td>
<td>Industry Practices of Software Engineering</td>
</tr>
</tbody>
</table>
## Syllabus

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Subject/Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>2022/05/18</td>
<td>Case Study on Software Engineering II</td>
</tr>
<tr>
<td>14</td>
<td>2022/05/25</td>
<td>Security and Privacy; Reliable Programming; Testing: Test-driven development, and Code reviews; DevOps and Code Management: DevOps automation</td>
</tr>
<tr>
<td>15</td>
<td>2022/06/01</td>
<td>Final Project Report I</td>
</tr>
<tr>
<td>16</td>
<td>2022/06/08</td>
<td>Final Project Report II</td>
</tr>
<tr>
<td>17</td>
<td>2022/06/15</td>
<td>Self-learning</td>
</tr>
<tr>
<td>18</td>
<td>2022/06/22</td>
<td>Self-learning</td>
</tr>
</tbody>
</table>
Agile Software Engineering
Software Engineering and Project Management

- **Analyze**: Requirements definition
- **Design**: System and Software design
- **Build**: Implementation and unit testing
- **Test**: Integration and system testing
- **Deliver**: Operation and maintenance

Project Management
Information Management (MIS)
Information Systems

**Project-based software engineering**

CUSTOMER generates requirements which helps-with the implementation by DEVELOPER.

Product software engineering

1. Opportunity

- inspires
- realizes

Product features implemented-by Software

Software execution models

Stand-alone execution
- User’s computer
  - User interface
  - Product functionality
  - User data
- Product updates
- Vendor’s servers

Hybrid execution
- User’s computer
  - User interface
  - Partial functionality
  - User data
- Additional functionality
  - User data backups
  - Product updates
- Vendor’s servers

Software as a service
- User’s computer
  - User interface (browser or app)
  - Product functionality
  - User data
- Vendor’s servers

Product management concerns

Business needs

Product manager

Technology constraints

Customer experience

Software Development Life Cycle (SDLC)

The waterfall model

- Requirements definition
- System and Software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

Plan-based and Agile development

Plan-based development

- Requirements engineering → Requirements specification → Design and implementation

Requirements change requests

Agile development

- Requirements engineering → Design and implementation
The Continuum of Life Cycles

Predictive Life Cycle

Analyze → Design → Build → Test → Deliver

Iterative Life Cycle

A Life Cycle of Varying-Sized Increments

## Iteration-Based and Flow-Based Agile Life Cycles

### Iteration-Based Agile

<table>
<thead>
<tr>
<th>Requirements Analysis</th>
<th>Design</th>
<th>Build</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Repeat as needed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
</tbody>
</table>

### Flow-Based Agile

<table>
<thead>
<tr>
<th>Requirements Analysis</th>
<th>Design</th>
<th>Build</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Repeat as needed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Design</td>
<td>Build</td>
<td>Test</td>
</tr>
</tbody>
</table>

From personas to features

1. Personas
   A way of representing users

2. Scenarios
   Natural language descriptions of a user interacting with a software product

3. Stories
   Natural language descriptions of something that is needed or wanted by users

4. Features
   Fragments of product functionality

Multi-tier client-server architecture

Service-oriented Architecture

Client 1
Client 2
Client 3
Client ...

Web Server

Service gateway

S1
S2
S3
S4
S5
S6

Services

VM

Virtual web server
Server software
Guest OS
Hypervisor
Host OS
Server Hardware

Virtual mail server
Server software
Guest OS

Container

User 1
Container 1
Application software
Server software

User 2
Container 2
Application software
Server software

Everything as a service

- **Infrastructure as a service (IaaS)**
  - Cloud data center
  - Storage
  - Network
  - Virtualization

- **Platform as a service (PaaS)**
  - Cloud management
  - Monitoring
  - Software development

- **Software as a service (SaaS)**
  - Photo editing
  - Logistics management
  - Database

Software as a service

Microservices architecture – key design questions

- What are the microservices that make up the system?
- How should data be distributed and shared?
- How should the microservices in the system be coordinated?
- How should microservices communicate with each other?
- How should service failure be detected, reported and managed?
Types of security threat

**Availability threats**
An attacker attempts to deny access to the system for legitimate users

Distributed denial of service (DDoS) attack

**Integrity threats**
An attacker attempts to damage the system or its data

Virus
Ransomware

**Confidentiality threats**
An attacker tries to gain access to private information held by the system

Data theft

Software product quality attributes
A refactoring process

1. Identify code ‘smell’
2. Identify refactoring strategy
3. Make small improvement until strategy completed
4. Run automated code tests
Test-driven development (TDD)

1. Identify new functionality
2. Identify partial implementation of functionality
3. Write code stub that will fail test
4. Run all automated test
5. Implement code that should cause failing test to pass
6. Run all automated test
7. Refactor code if required

Functionality incomplete
Functionality complete

All tests pass

DevOps

Multi-skilled DevOps team

Code management and DevOps

DevOps automation
- Continuous integration
- Continuous deployment
- Continuous delivery
- Infrastructure as code

Code management system
- Branching and merging
- Code repository
- Transfer code to/from developer’s filestore
- Recover version information
- Save and retrieve versions

DevOps measurement
- Data collection
- Data analysis
- Report generation

Agile
Agile software engineering

• Software products must be brought to market quickly so rapid software development and delivery is essential.

• Virtually all software products are now developed using an agile approach.

• Agile software engineering focuses on delivering functionality quickly, responding to changing product specifications and minimizing development overheads.

Agile Manifesto Values, Principles, and Common Practices

Agile is a Blanket Term for Many Approaches

Lean

Agile

Kanban
ScrumBan
Scrum
Crystal
XP
FDD
AUP
DSDM

Agile Manifesto and Mindset

Agile is a mindset defined by 4 values, guided by 12 principles, and manifested through many different practices.

Agile practitioners select practices based on their needs.

4

Agile Values
The Four Values of the Agile Manifesto
(Manifesto for Agile Software Development, 2001)

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
Agile Principles
The Twelve Principles
Behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. **Working software** is the primary measure of progress.

8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. **Continuous attention** to technical excellence and good design enhances agility.

10. **Simplicity**—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from **self-organizing teams**.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Development Principles

• Involve the customer
  Involve customers closely with the software development team. Their role is to provide and prioritize new system requirements and to evaluate each increment of the system.

• Embrace change
  Expect the features of the product and the details of these features to change as the development team and the product manager learn more about it. Adapt the software to cope with changes as they are made.

• Develop and deliver incrementally
  Always develop software products in increments. Test and evaluate each increment as it is developed and feed back required changes to the development team.

Agile Development Principles

• **Maintain simplicity**
  Focus on simplicity in both the software being developed and in the development process. Wherever possible, do what you can to eliminate complexity from the system.

• **Focus on people, not things**
  Trust the development team and do not expect everyone to always do the development process in the same way. Team members should be left to develop their own ways of working without being limited by prescriptive software processes.

12

Project Management Principles
## Project Management Body of Knowledge (PMBOK Guide)

### PMBOK Guide v6

- **Project Management Body of Knowledge:**
  - Introduction
  - Project Environment
  - Role of the Project Manager
  - 10 Knowledge Areas
    - Integration
    - Scope
    - Schedule
    - Cost
    - Quality
    - Resources
    - Communications
    - Risk
    - Procurement
    - Stakeholders

### PMBOK Guide v7

- **The Standard for Project Management:**
  - Introduction
  - System for Value Delivery
  - 12 Project Management Principles:
    - 1. Stewardship, 2. Team
    - 3. Stakeholders, 4. Value
    - 5. Systems Thinking, 5. Leadership
    - 7. Tailoring, 8. Quality
    - 9. Complexity, 10. Risk
    - 11. Adaptability and Resiliency
    - 12. Change

### The Standard for Project Management (5 Process Groups):

- Initiating
- Planning
- Executing
- Monitoring and Controlling
- Closing

### Project Management Body of Knowledge (8 Project Performance Domains):

- Stakeholders, 2. Team,
- 3. Development approach and Life Cycle
- 4. Planning, 5. Project Work, 6. Delivery,
- 7. Measurement, 8. Uncertainty

- Tailoring
- Models, Methods, and Artifacts

---

Project Management Knowledge Areas
(PMBOK v6)

1. Project Integration Management
2. Project Scope Management
3. Project Schedule Management
4. Project Cost Management
5. Project Quality Management
6. Project Resource Management
7. Project Communications Management
8. Project Risk Management
9. Project Procurement Management
10. Project Stakeholder Management

Project Management Process Groups
(PMBOK v6)

1. Initiating Process Group
2. Planning Process Group
3. Executing Process Group
4. Monitoring and Controlling Process Group
5. Closing Process Group

Project Management 12 Principles
(PMBoK v7)

1. Stewardship
2. Team
3. Stakeholders
4. Value
5. Systems Thinking
6. Leadership
7. Tailoring
8. Quality
9. Complexity
10. Risk
11. Adaptability and Resiliency
12. Change

Project Management

8 Project Performance Domains
(PMBOK v7)

1. Stakeholders
2. Team
3. Development Approach and Life Cycle
4. Planning
5. Project Work
6. Delivery
7. Measurement
8. Uncertainty

A large number of ‘agile methods’ have been developed.

- There is no ‘best’ agile method or technique.
- It depends on who is using the technique, the development team and the type of product being developed.

Agile methods

• **Plan-driven development** evolved to support the engineering of large, long-lifetime systems
  
  • This approach is based on controlled and rigorous software development processes that include detailed project planning, requirements specification and analysis and system modelling.

  • However, plan-driven development involves significant overheads and documentation and it does not support the rapid development and delivery of software.

• **Agile methods** were developed in the 1990s to address this problem.

  • These methods focus on the software rather than its documentation, develop software in a series of **increments** and aim to reduce process bureaucracy as much as possible.

Incremental development

• All agile methods are based around incremental development and delivery.

• Product development focuses on the software features, where a feature does something for the software user.

• With incremental development, you start by prioritizing the features so that the most important features are implemented first.
  
  • You only define the details of the feature being implemented in an increment.
  
  • That feature is then implemented and delivered.

• Users or surrogate users can try it out and provide feedback to the development team. You then go on to define and implement the next feature of the system.

Incremental development

1. Choose features to be included in increment
2. Refine features descriptions
3. Implement and test feature
4. Integrate feature into system
5. Deliver system increment

Product feature list

If all features are complete, deliver system release

Incremental development activities

1. Choose features to be included in an increment
   Using the list of features in the planned product, select those features that can be implemented in the next product increment.

2. Refine feature descriptions
   Add detail to the feature descriptions so that the team have a common understanding of each feature and there is sufficient detail to begin implementation.

3. Implement and test
   Implement the feature and develop automated tests for that feature that show that its behaviour is consistent with its description.

4. Integrate feature and test
   Integrate the developed feature with the existing system and test it to check that it works in conjunction with other features.

5. Deliver system increment
   Deliver the system increment to the customer or product manager for checking and comments. If enough features have been implemented, release a version of the system for customer use.

Extreme programming

• The most influential work that has changed software development culture was the development of Extreme Programming (XP).

• The name was coined by Kent Beck in 1998 because the approach was developed by pushing recognized good practice, such as iterative development, to ‘extreme’ levels.

• Extreme programming focused on 12 new development techniques that were geared to rapid, incremental software development, change and delivery.

• Some of these techniques are now widely used; others have been less popular.

Extreme Programming Practices

- Test-first development
- Refactoring
- Continuous integration
- Small releases
- Simple design
- On-site customer
- Sustainable pace
- Pair programming
- Collective ownership
- Incremental planning

Extreme Programming (XP)

Widely adopted XP practices

• Incremental planning/user stories
  • There is no ‘grand plan’ for the system. Instead, what needs to be implemented (the requirements) in each increment are established in discussions with a customer representative.
  • The requirements are written as user stories.
  • The stories to be included in a release are determined by the time available and their relative priority.

Widely adopted XP practices

• 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 previous release.

Widely adopted XP practices

• Test-driven development
  
  • Instead of writing code then tests for that code, developers write the tests first.
  
  • This helps clarify what the code should actually do and that there is always a ‘tested’ version of the code available.
  
  • An automated unit test framework is used to run the tests after every change.
  
  • New code should not ‘break’ code that has already been implemented.

Widely adopted XP practices

• Continuous integration
  • As soon as the work on a task is complete, it is integrated into the whole system and a new version of the system is created.
  • All unit tests from all developers are run automatically and must be successful before the new version of the system is accepted.

Widely adopted XP practices

• Refactoring
  • Refactoring means improving the structure, readability, efficiency and security of a program.
  • All developers are expected to refactor the code as soon as potential code improvements are found.
  • This keeps the code simple and maintainable.

Widely adopted XP practices

• Incremental planning/user stories
  • There is no ‘grand plan’ for the system. Instead, what needs to be implemented (the requirements) in each increment are established in discussions with a customer representative.
  • The requirements are written as user stories.
  • The stories to be included in a release are determined by the time available and their relative priority.

Scrum

• Software company managers need information that will help them understand how much it costs to develop a software product, how long it will take and when the product can be brought to market.

• Plan-driven development provides this information through long-term development plans that identify deliverables - items the team will deliver and when these will be delivered.

• Plans always change so anything apart from short-term plans are unreliable.

• Scrum is an agile method that provides a framework for agile project organization and planning. It does not mandate any specific technical practices.

Scrum Terminology

• **Scrum**
  A daily team meeting where progress is reviewed and work to be done that day as discussed and agreed.

• **Sprint**
  A short period, typically two to four weeks, when a product increment is developed.

• **ScrumMaster**
  A team coach who guides the team in the effective use of Scrum.

Scrum Terminology

• **Product**
  The software product that is being developed by the Scrum team.

• **Product owner**
  A team member who is responsible for identifying product features and attributes. They review work done and help to test the product.

• **Product backlog**
  A to-do list of items such as bugs, features and product improvements that the Scrum team have not yet completed.

Scrum Terminology

• Development team
  A small self-organising team of five to eight people who are responsible for developing the product.

• Potentially shippable product increment
  The output of a sprint which should be of high enough quality to be deployed for customer use.

• Velocity
  An estimate of how much work a team can do in a single sprint.

Key roles in Scrum

• **The Product Owner** is responsible for ensuring that the development team are always focused on the product they are building rather than diverted into technically interesting but less relevant work.
  
  • In product development, the product manager should normally take on the Product Owner role.

• **The ScrumMaster** is a Scrum expert whose job is to guide the team in the effective use of the Scrum method. The developers of Scrum emphasize that the ScrumMaster is not a conventional project manager but is a coach for the team. They have authority within the team on how Scrum is used.
  
  • In many companies that use Scrum, the ScrumMaster also has some project management responsibilities.

Scrum and sprints

• In Scrum, software is developed in sprints, which are fixed-length periods (2 - 4 weeks) in which software features are developed and delivered.

• During a sprint, the team has daily meetings (Scrums) to review progress and to update the list of work items that are incomplete.

• Sprints should produce a ‘shippable product increment’. This means that the developed software should be complete and ready to deploy.

Scrum cycles

Start

1. Review product backlog

Select items to implement

Plan sprint

Product backlog

Sprint backlog

Sprint

Test software

Develop software

Scrum

Shippable product increment

Key Scrum practices

• Product backlog
  This is a to-do list of items to be implemented that is reviewed and updated before each sprint.

• Timeboxed sprints
  Fixed-time (2-4 week) periods in which items from the product backlog are implemented,

• Self-organizing teams
  Self-organizing teams make their own decisions and work by discussing issues and making decisions by consensus.

The product backlog is a list of what needs to be done to complete the development of the product.

The items on this list are called product backlog items (PBIs).

The product backlog may include a variety of different items such as product features to be implemented, user requests, essential development activities and desirable engineering improvements.

The product backlog should always be prioritized so that the items that be implemented first are at the top of the list.

Examples of Product Backlog Items (PBIs)

1. As a teacher, I want to be able to configure the group of tools that are available to individual classes. (feature)

2. As a parent, I want to be able to view my children’s work and the assessments made by their teachers. (feature)

3. As a teacher of young children, I want a pictorial interface for children with limited reading ability. (user request)

4. Establish criteria for the assessment of open source software that might be used as a basis for parts of this system. (development activity)

5. Refactor user interface code to improve understandability and performance. (engineering improvement)

6. Implement encryption for all personal user data. (engineering improvement)

Product backlog item states

• **Ready for consideration**
  These are high-level ideas and feature descriptions that will be considered for inclusion in the product. They are tentative so may radically change or may not be included in the final product.

• **Ready for refinement**
  The team has agreed that this is an important item that should be implemented as part of the current development. There is a reasonably clear definition of what is required. However, work is needed to understand and refine the item.

• **Ready for implementation**
  The PBI has enough detail for the team to estimate the effort involved and to implement the item. Dependencies on other items have been identified.

Product backlog activities

PRODUCT BACKLOG

- PBI 1
- PBI 2
- PBI 3
- PBI 4
- PBI 5

Refinement

Estimation

Prioritization

Creation

REVISED PRODUCT BACKLOG

- PBI 1.1
- PBI 1.2
- PBI 2E
- PBI 3E
- PBI 4
- PBI 5
- PBI 6

Sprint activities

Managing External Interactions

External interactions

Team-focused external interactions
- ScrumMaster

Product-focused external interactions
- Product owner

Project Management Responsibilities

- **Budget**
- **Schedule**
- **Risks**
- **Problems**
- **Progress**

**Reporting**

- **Finance**
- **Compliance**
- **Procurement**
- **Liaison**

**Administration**

- **Vacations**
- **Absence**
- **Work quality**
- **Reviewing**
- **Hiring**

**People**

Summary

• The best way to develop software products is to use agile software engineering methods that are geared to rapid product development and delivery.

• Agile methods are based around iterative development and the minimization of overheads during the development process.

• Extreme programming (XP) is an influential agile method that introduced agile development practices such as user stories, test-first development and continuous integration. These are now mainstream software development activities.

• **Scrum** is an agile method that focuses on agile planning and management. Unlike XP, it does not define the engineering practices to be used. The development team may use any technical practices that they believe are appropriate for the product being developed.

• In **Scrum**, work to be done is maintained in a **product backlog** – a list of work items to be completed. Each increment of the software implements some of the work items from the product backlog.

Summary

- **Sprints** are fixed-time activities (usually 2–4 weeks) where a product increment is developed. Increments should be ‘potentially shippable’ i.e. they should not need further work before they are delivered.

- A **self-organizing team** is a development team that organizes the work to be done by discussion and agreement amongst team members.

- **Scrum practices** such as the product backlog, sprints and self-organizing teams can be used in any agile development process, even if other aspects of Scrum are not used.

References


• Titus Winters, Tom Manshreck, and Hyrum Wright (2020), Software Engineering at Google: Lessons Learned from Programming Over Time, O'Reilly Media.

