

Software Engineering

Prepared by Dr. Yasser Abdelhamid

IT 382

Software Engineering

Lecture 5





LECTURE OUTLINE

- ◉ **Agile software development**
- ◉ Agile methods
- ◉ Principles of Agile methods
- ◉ Problems with agile methods
- ◉ When to use agile
- ◉ Extreme programming

RAPID SOFTWARE DEVELOPMENT

- ❖ 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

AGILE METHODS

- ❖ Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of 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 quickly to meet changing requirements.**

AGILE METHODS

- ❖ 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.

AGILE MANIFESTO

- ❖ *We are uncovering better ways of developing software by doing it and helping others do it.*
 - *Working software over comprehensive documentation*
 - *Customer collaboration over contract negotiation*
 - *Responding to change over following a plan*

PRINCIPLES OF AGILE METHODS

- Customer involvement
- Incremental delivery
- People not process
- Embrace change
- Maintain simplicity

CUSTOMER INVOLVEMENT

❖ Customers should be closely involved throughout the development process. Their role is **providing** and **prioritizing** 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.

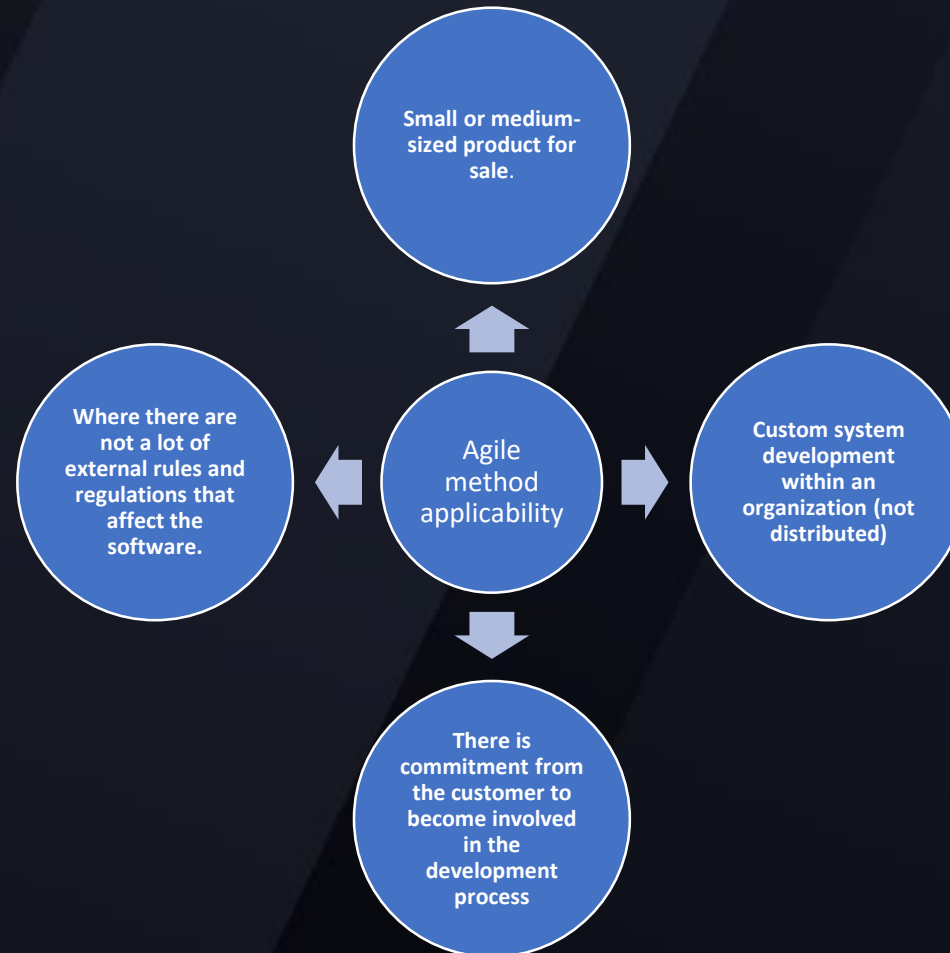
EMBRACE CHANGE

❖ **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.

WHEN AGILE IS SUITABLE?



PROBLEMS WITH AGILE METHODS

keeping the interest of customers

Suitability of team members

Multiple stakeholders

Extra work for maintaining simplicity

Contracts

KEEPING THE INTEREST OF CUSTOMERS

- ❖ It can be difficult **to keep the interest** of customers who are involved in the process.

SUITABILITY OF TEAM MEMBERS

- ❖ Team members may **be unsuited to the intense** involvement that characterizes agile methods.

MULTIPLE STACKHOLDERS

- ❖ Prioritizing changes can be difficult where there are multiple stakeholders.

EXTRA WORK FOR MAINTAINING SIMPLICITY

❖ Maintaining simplicity requires extra work.

CONTRACTS

❖ Contracts may be a problem as with other approaches to iterative development.

WHEN TO USE AGILE

Is it important to have a very detailed specification and design before moving to implementation?

Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic?

How large is the system that is being developed?

What type of system is being developed?

What is the expected system lifetime?

What technologies are available to support system development?

How is the development team organized?

Are there cultural or organizational issues that may affect the system development?

How good are the designers and programmers in the development team?

Is the system subject to external regulation?

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.

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?

❖ Plan-driven is appropriate 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.

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 must be approved by an external regulator (e.g. the FAA approve 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.

EXTREME PROGRAMMING

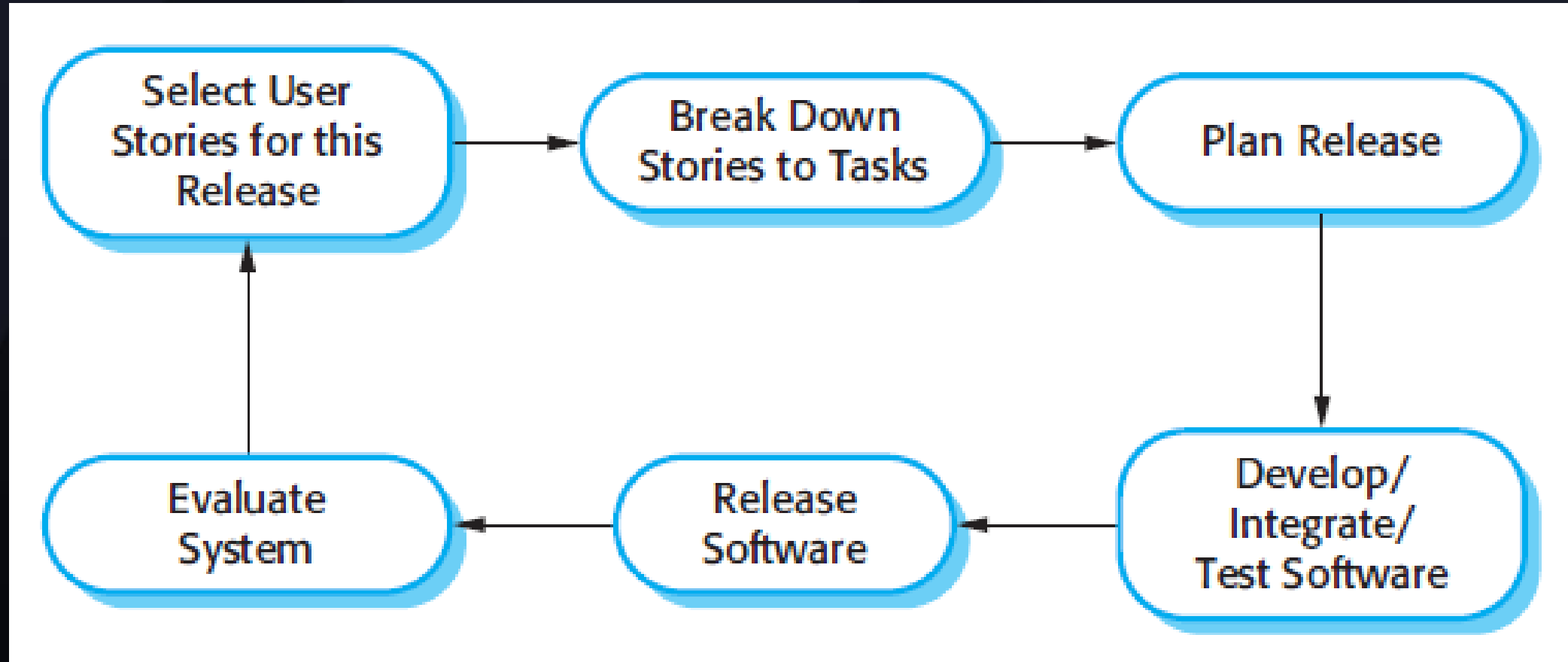
EXTREME PROGRAMMING

- ❖ Perhaps the best-known and most widely used agile method.
- ❖ 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.
- ❖ **Pair programming, collective ownership.**
- ❖ Change supported through **regular system releases**.
- ❖ Maintaining simplicity through constant **refactoring** of code.

THE EXTREME PROGRAMMING RELEASE CYCLE



EXTREME PROGRAMMING PRACTICES

Incremental planning

Small releases

Simple design

Test-first development

Refactoring

Pair programming

Collective ownership

Continuous integration

Sustainable pace

On-site customer

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.

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 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.

EXAMPLE: A 'PRESCRIBING MEDICATION' STORY

Prescribing Medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select 'current medication', 'new medication' or 'formulary'.

If she selects 'current medication', the system asks her to check the dose. If she wants to change the dose, she enters the dose and then confirms the prescription.

If she chooses 'new medication', the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose and then confirms the prescription.

If she chooses 'formulary', the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose and then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn't, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks 'OK' or 'Change'. If she clicks 'OK', the prescription is recorded on the audit database. If she clicks on 'Change', she reenters the 'Prescribing medication' process.

EXAMPLES OF TASK CARDS FOR PRESCRIBING MEDICATION

Task 1: Change Dose of Prescribed Drug

Task 2: Formulary Selection

Task 3: Dose Checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary ID for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

REFACTORING

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.

SUMMARY

- 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.



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

