

Agile Software Development

Fattane Zarrinkalam

1

Rapid software development

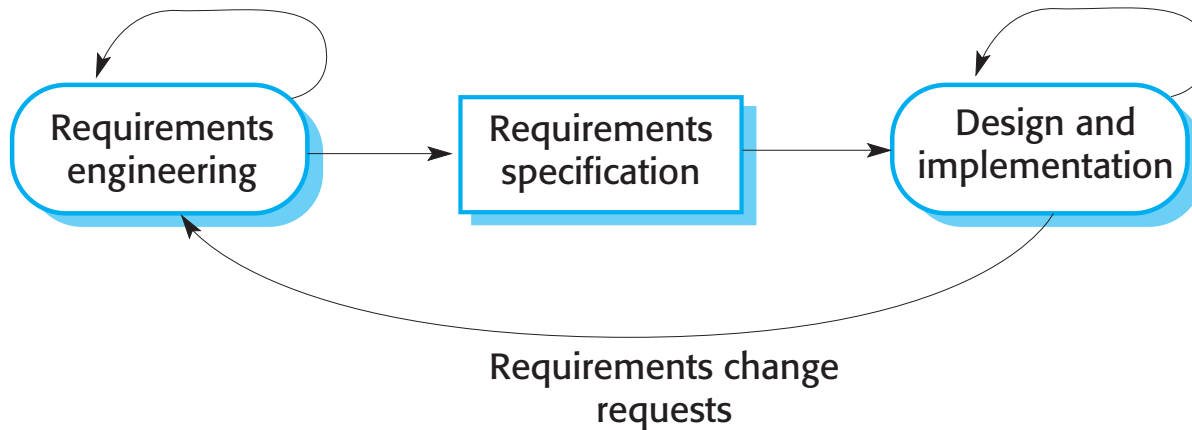
- **Rapid development and delivery** is now often the most important requirement for software systems
 - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements.
 - Software has to evolve quickly to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically **reduce the delivery time** for working software systems.

Agile development

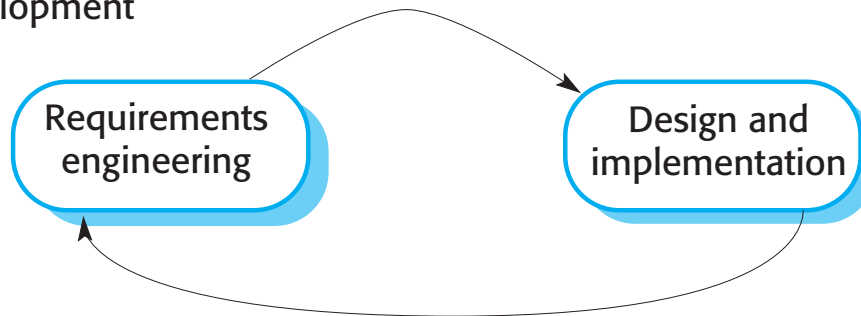
- Program specification, design and implementation are **inter-leaved**
- The system is developed as a series of versions or **increments** with stakeholders involved in version specification and evaluation.
- **Frequent delivery of new versions** for evaluation
- Extensive **tool support** (e.g., automated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

Plan-driven and agile development

Plan-based development



Agile development



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.

Manifesto for Agile Software Development

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

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.

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.
 - Virtually all software products and apps are now developed using an agile approach.
- **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.

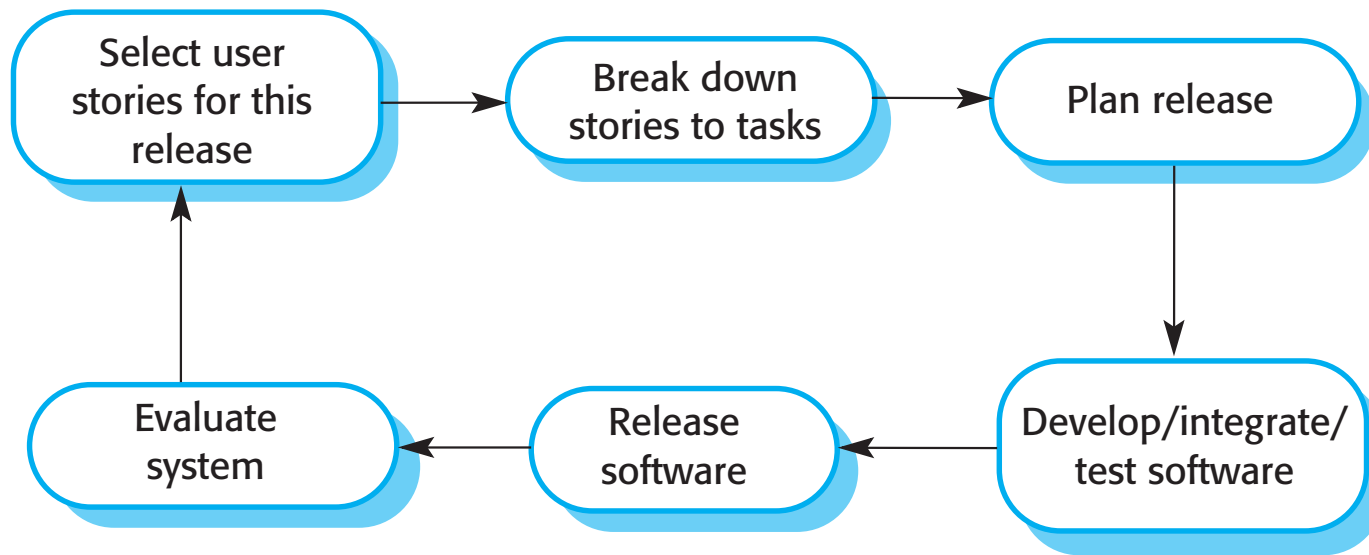


Agile development techniques

Extreme programming

- 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



A sample user story for a mobile game: “As a player, I want to see the scoreboard, so that I can compare my performance with other players”

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'. 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 (b)

Principle or practice	Description
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.

Influential XP practices

- 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

User stories for requirements

User stories for requirements

- 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 user stories or scenarios.
- 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.

A 'prescribing medication' story

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

User stories for requirements

- User stories should always be **personalized**. Names of people should be used
 - This help users to relate to these stories
- User stories should always be written in **simple language** without jargon
 - This makes them understandable to non-technical users

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, lookup 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.

User stories for requirements

- User stories are effective in **engaging users** and other stakeholders in the requirements engineering process.
- User stories are **not good for expressing detailed requirements** such as how to do a complex stories how to do a calculation or for describing system-wide non-functional requirements.
- User stories **should not just be used on their own** but alongside other techniques for understanding other techniques.



Refactoring

Refactoring

- 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 reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

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.

Test-first development

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 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

- 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.

Test case description for dose checking

Test 4: Dose checking

Input:

1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:

1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:

OK or error message indicating that the dose is outside the safe range.

Test automation

- 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.

Problems with test-first development

- 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 **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.

Pair programming

Pair programming

- Pair programming involves programmers working in pairs, developing code together.
- 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.

Pair programming

- In pair programming, programmers **sit together** at the same computer 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 some evidence that suggests that a pair working together is more **efficient than 2 programmers working separately**.

A grayscale photograph of a large crowd of people, likely at a sporting event, with the word 'Summary' overlaid in the center.

Summary

Summary

- 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.
- **Extreme Programming** (XP) takes an '**extreme**' approach to iterative development.
 - User stories for specification
 - Refactoring
 - Test-first development
 - Pair programming
 - ...