Agile Software Development

Ohjelmankehityspr., versionhallinta ja testaus – Chapter 8

Rapid software development

- Delivery most important requirements for software systems
  - Business operate in a fast changing requirements
  - Practically impossible to produce set of stable software requirements

- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System developed as a series of version
  - Stakeholders are involved in version evaluation
  - User interfaces developed using an IDE & graphical toolset

Agile Methods

↗ Focus on the code rather than design

↗ Based on iterative approach to software development

↗ Focus on delivering on working software quickly

↗ Focus on evolving quickly to meet changing requirements

↗ Overall aim
  - ↗ to reduce overheads in the software process (limiting documentation)
  - ↗ To be able to respond quickly to changing requirements without excessive rework

Agile Mainefesto

↗ We are uncovering better ways of developing software by doing it and helping others do it. 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.

Agillemainfesto.org

Principle 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

↗ Software product development (small or medium-sized product).

↗ 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

↗ Might be difficult to keep the interest (long time) 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 methods & software maintenance

- ↗ Software maintenance costs more to many organization than on new software development

- ↗ Agile methods to be successful, there should be enough support maintenance as well as original development

- ↗ Key issues
  - ↗ Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  - ↗ Can agile methods be used effectively for evolving a system in response to customer change requests?

- ↗ Problems may arise if original development team cannot be maintained.

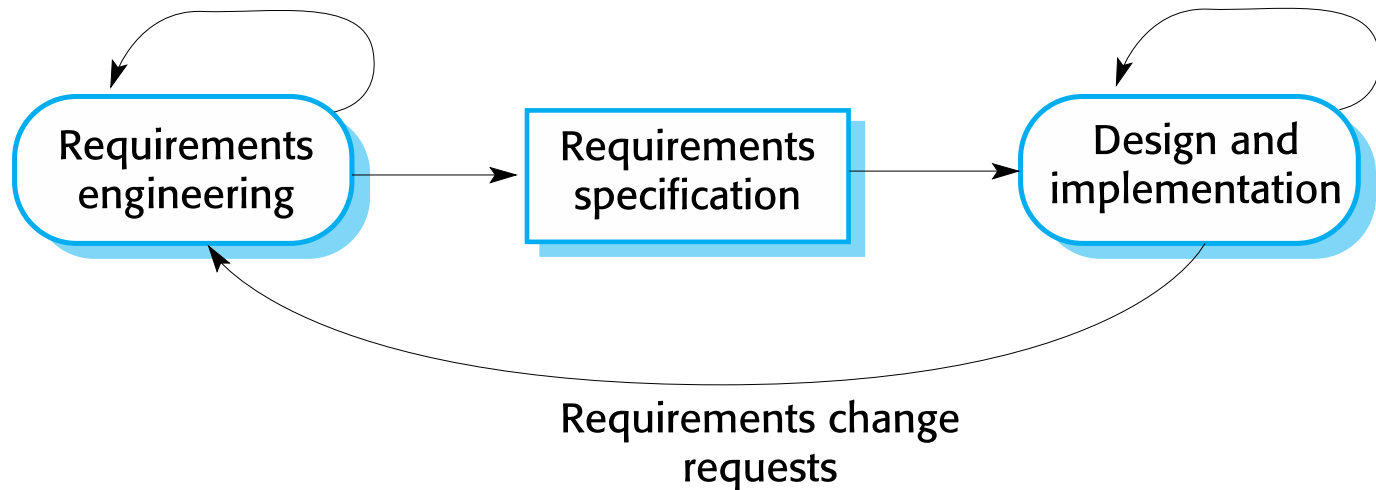# Plan drive & agile development

Plan driven development

↗ A plan driven approach is based on

  ↗ Separate development stages

  ↗ Outputs to be produced at each of these stages

  ↗ Outputs planned in advance

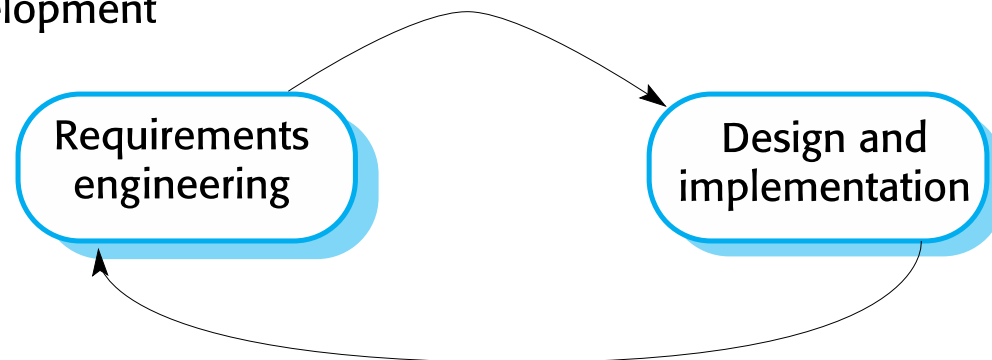 ↗ Iteration occurs within activities

Agile development

↗ Specification, design & implementation interleaved

↗ 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

Requirements engineering → Requirements specification → Design and implementation

Requirements change requests

Agile development

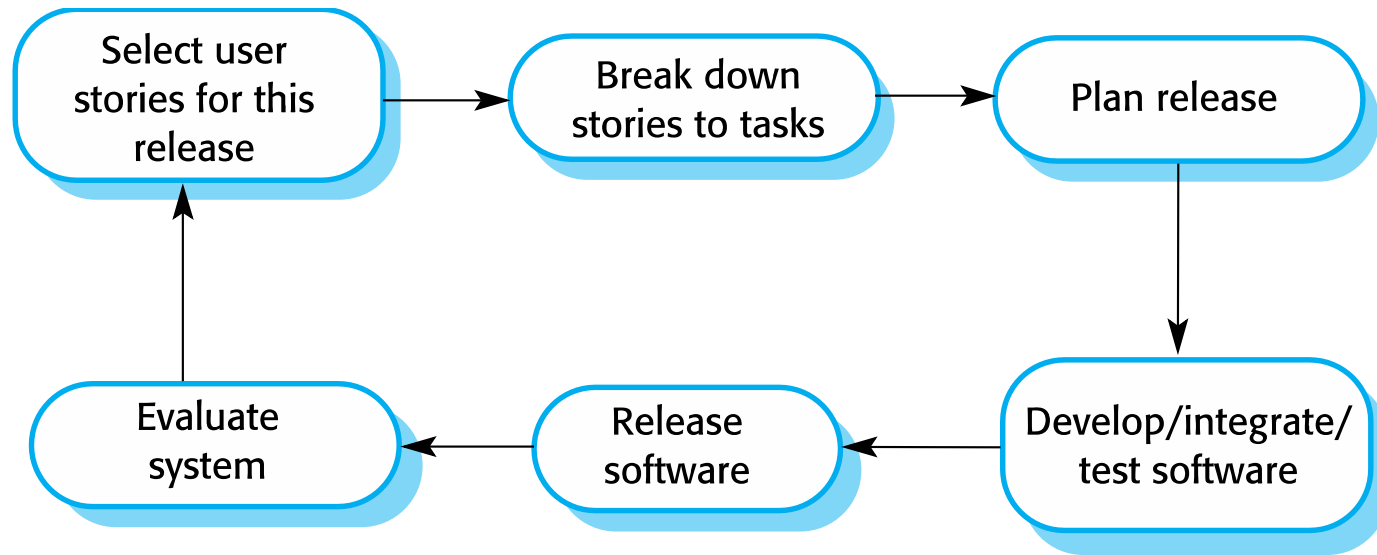Requirements engineering → Design and implementation

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.

↗ People not process 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.

Release cycle – Extreme programming


```
Select user        →   Break down       →   Plan release
stories for this       stories to tasks
release                                          |
   ↑                                             ↓
Evaluate          ←    Release         ←    Develop/integrate/
system                 software             test software
```

The rules of extreme programming: http://www.extremeprogramming.org/rules.html

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.

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.

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.

XP & change

- ↗ 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.

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.

Exercise

↗ Prepare at least 2 -3 user stories (task cards) for your own project. Refer to the previous exercises done.

References

↗ Software Engineering, 9th Edition by Ian Sommerville