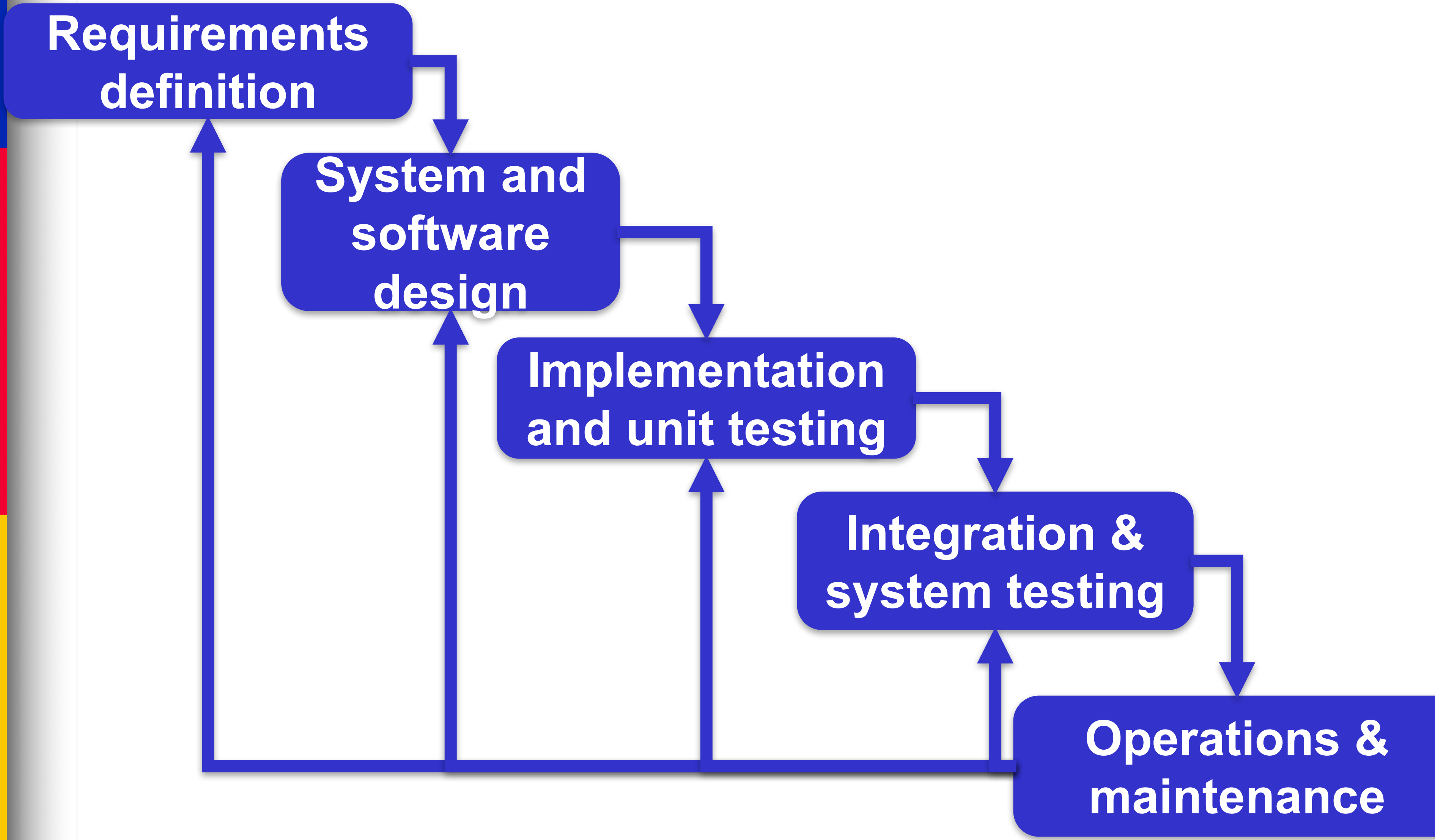
SOEN 341
Software Process

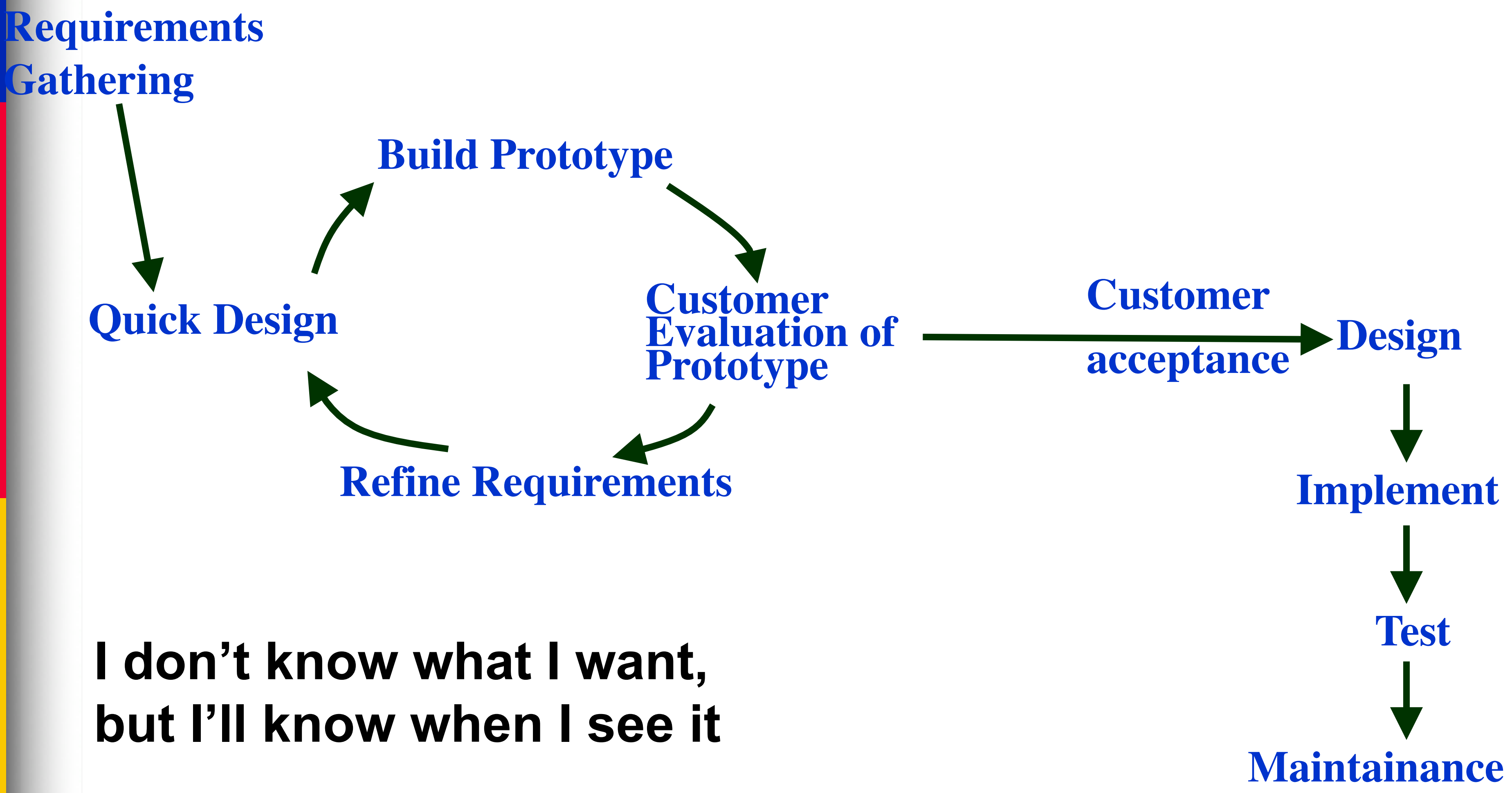
Lecture 04:
Agile Methods

Emad Shihab, PhD

Process 1: The Waterfall Model

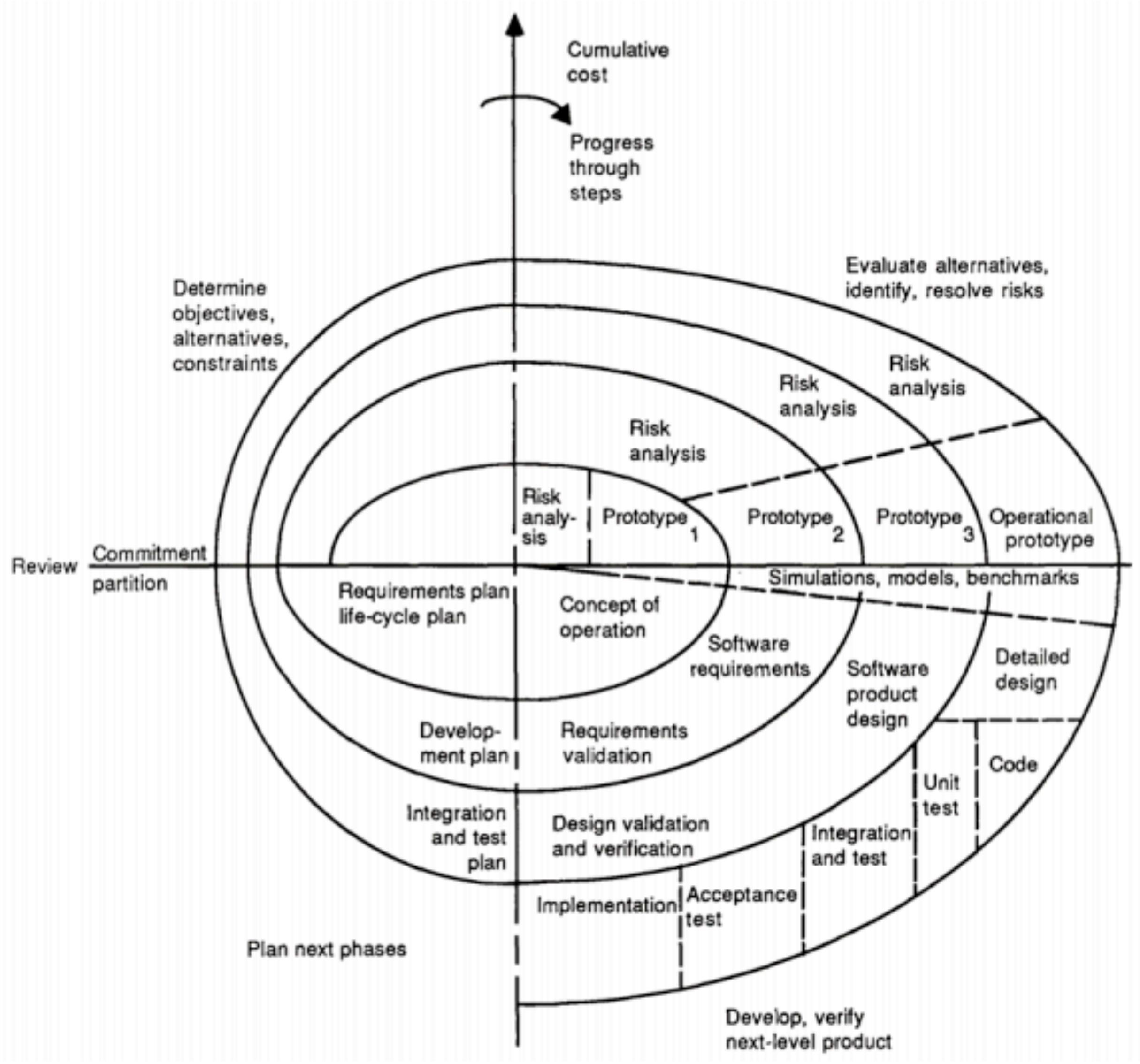

Requirements definition
System and software design
Implementation and unit testing
Integration & system testing
Operations & maintenance

Process 1.5: Prototype/evolutionary model



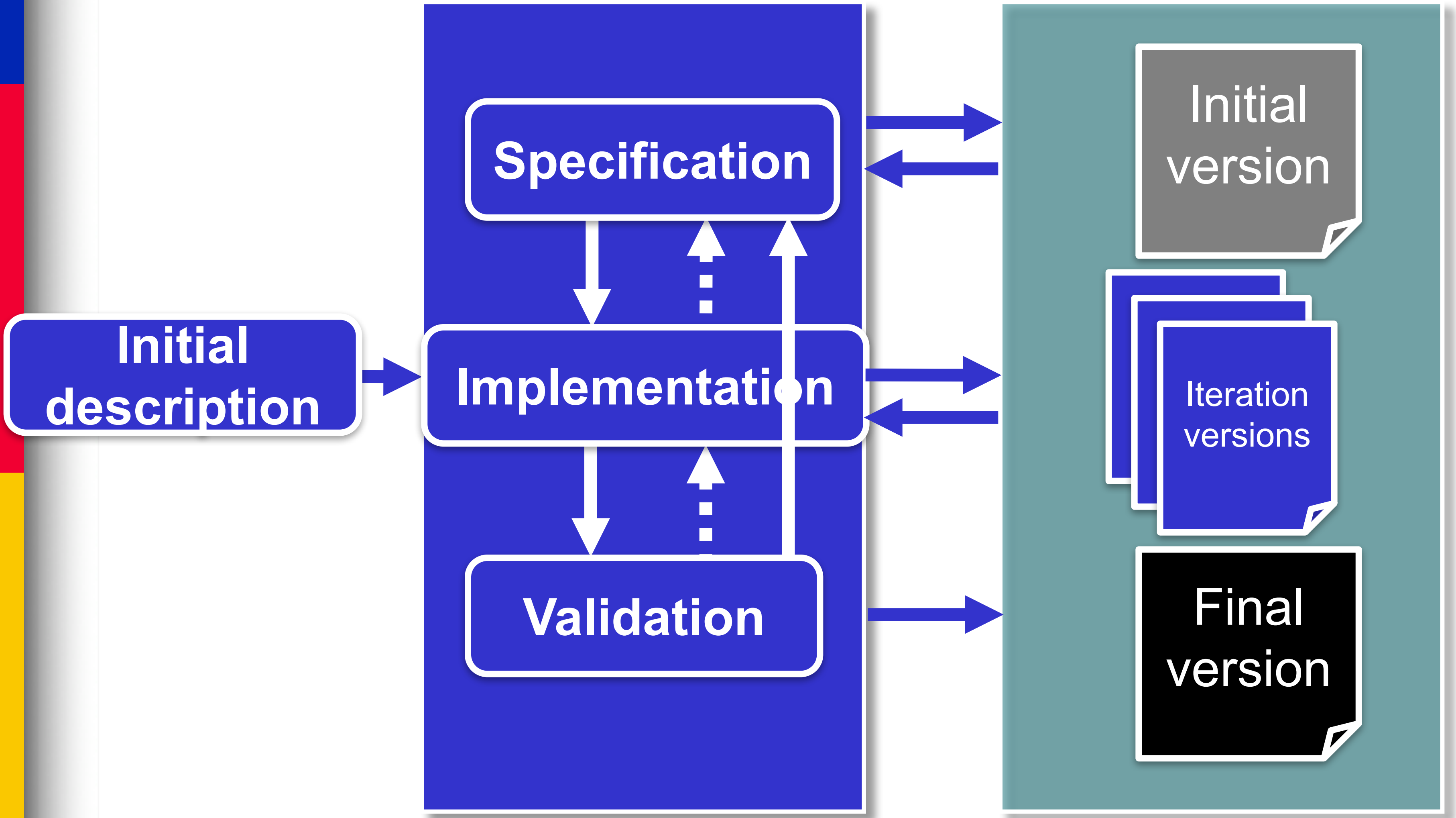
I don’t know what I want, but I’ll know when I see it

Process 2.0: Spiral Model



http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

Process 3: Incremental Development



Rapid software development

- **Rapid development** and delivery is now often the **most important requirement** for software systems
  - Software has to evolve quickly to reflect changing business needs.

- Since **plan-driven development is deemed too slow**, agile development methods emerged in the late 1990s, with the aim of radically reducing 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 frequent versions** with stakeholders involved in version specification and evaluation

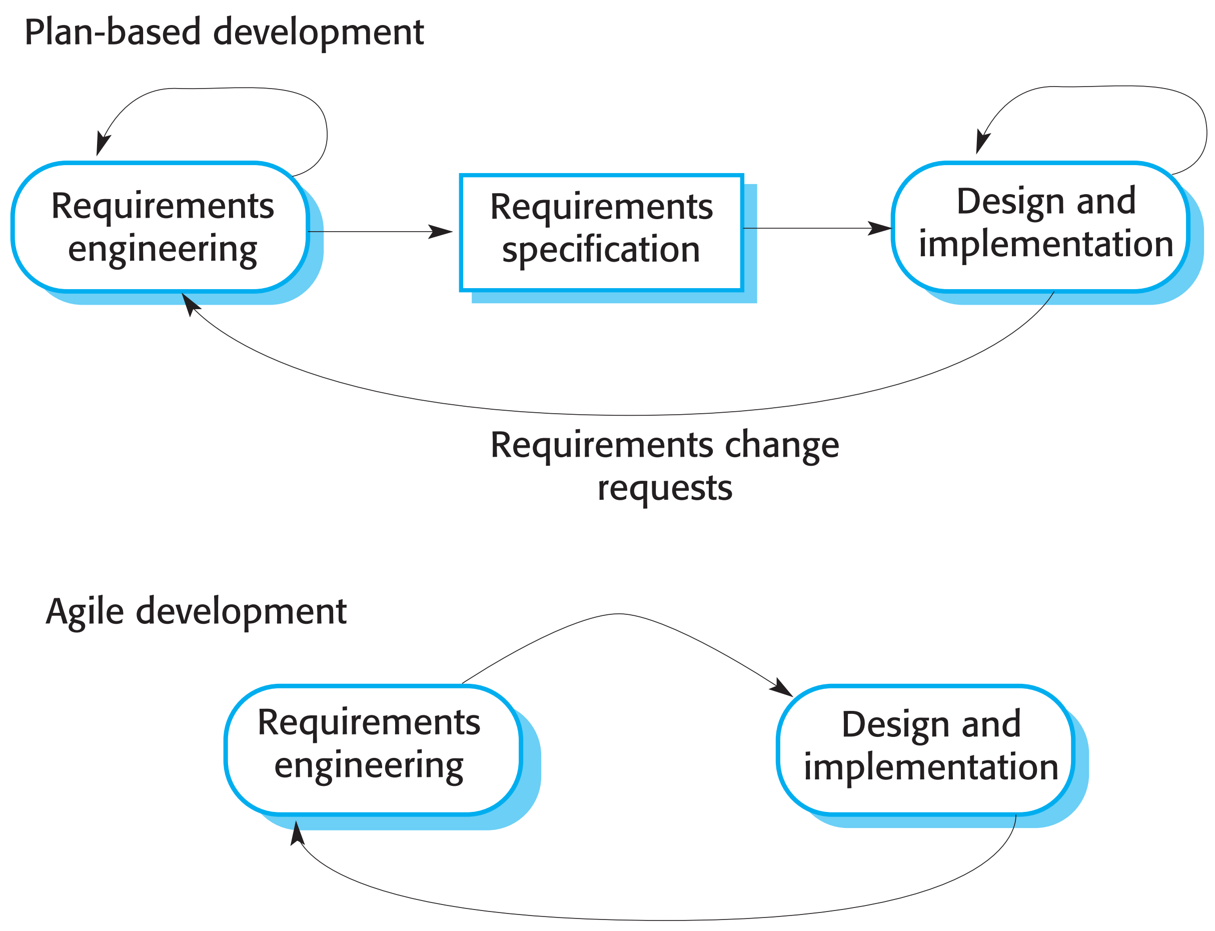
- **Extensive tool support** (e.g. automated testing tools) used to support development.

- **Minimal documentation** – focus on working code

Terminology

- **Rapid software development** gained traction with the idea of “**agile methods**”
- **Examples of agile methods**: Extreme programming (XP), Scrum, etc.
- At some point, **rapid software development became known as agile development or agile methods**

Plan-driven vs. Agile dev


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Design and implementation are the central activities. Iterations occur across activities.

Iteration within activities with formal documents used to communicate

Words of caution

- Plan-driven development need not only apply to the waterfall model – plan-driven, incremental development **is possible**
  - Iteration occurs within activities.

- Although specification, design, implementation and testing are inter-leaved in agile methods, the **output need not (always) be code**.

- In practice, a hybrid of the two approaches is applied.

The Agile Manifesto & Principles

Agile Manifesto

- *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*

Agile Manifesto principles -Individuals and interactions

- Build projects around motivated individuals
- Give them support they need
- Trust them to get the job done
- **Face-to-face conversation is the most effective method of conveying information**
- Applies to inter- and intra-team communication
- Emergent self-organizing teams produce the best software artifacts

Agile Manifesto principles -Working software

- Working software is the primary measure of progress
- **Top priority is customer satisfaction**
- Achieved through early and continuous delivery of valuable software
- Simplicity is essential
- **Maximizing the amount of work NOT done**

Agile manifesto principles -Customer collaboration

- Clients and developers must work together daily throughout the project
- **Deliver working software frequently**
- Every couple of:
  - Months
  - Weeks
  - Days
- Preference is given to shorter timescales!
- Project cadence should be sustainable
- Stakeholders should maintain a constant pace

Agile manifesto principles -Responding to change

- **Changing requirements are welcome**
- Even late in development!
- **Continuous attention to technical details**
- Good design enhances agility
- **Team regularly reflects on how to become more effective**
- Behavior is adjusted accordingly

Agile Manifesto (summary)

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**.

Agile method applicability

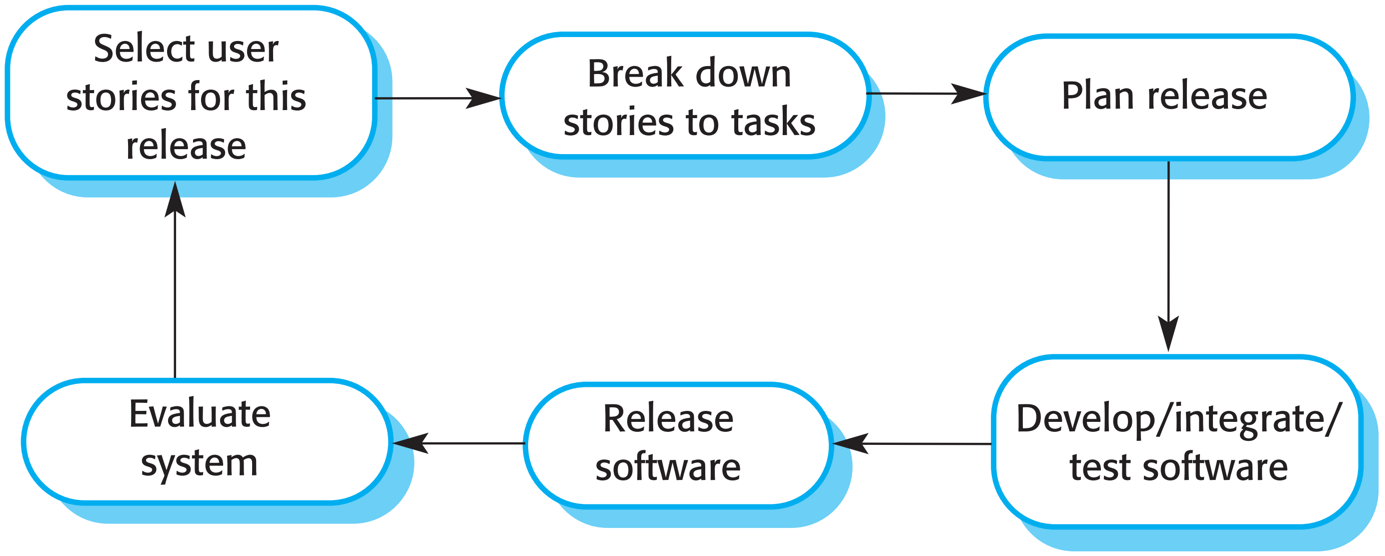
- Product development where a software company is developing a small or medium-sized product for sale.
  - Virtually all software products and apps are now developed using an agile approach

Agile Development Techniques

Agile development techniques

- 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.

Extreme programming release cycle


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

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.

Extreme programming practices

Pair programming	**Developers work in pairs**, checking each other's work and providing the support to always do a good job.

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

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 **original method is not widely used**.
- **Key adopted practices**
  - **User stories for specification**
  - **Refactoring**
  - **Test-first development**
  - **Pair programming**

User stories

User stories for requirements

- User **requirements** are **expressed as user stories** or scenarios.
- **User stories** are broken down by the development team into **implementation tasks**.
- The **customer chooses the stories** for inclusion in the next release based on their **priorities and the schedule** estimates.

Example 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.

Refactoring

Refactoring

- XP, however, maintains that **it is not worth ‘designing for change’** since changes cannot be reliably anticipated.

- Rather, it **proposes constant code improvement (refactoring)** to make changes easier when they have to be implemented. This **improves the understandability** of the software and so **reduces the need for documentation**.

- However, some changes requires **architecture refactoring** and this is **much more expensive**.

Example 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 method calls** that have been included in a program library.

Test-first development

Test-first development

- XP testing features:
  - Test-first development. This **clarifies the requirements to be implemented**
  - 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. Usually relies on a testing framework such as Junit.

Problems with test-first development

- **Programmers prefer programming to testing** and sometimes they take short cuts when writing tests.
  - may write incomplete tests that do not check for all possible exceptions that may occur.

- Some tests can be very difficult to write incrementally. E.x., testing a complex UI

- 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**.

Customer involvement

- 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, this is difficult to achieve since:
  - customer have limited time available
  - only want to provide requirements
  - reluctant to get involved in the testing process.

Pair programming

Pair programming

- Programmers working in pairs, developing code together. Which helps **develop common ownership** of code and spreads knowledge across the team.

- Serves as an **informal review process** as each line of code is looked at by more than 1 person.

- **Not necessarily inefficient**, but the verdict is still out on this one!

Next class

Agile planning

Quiz

XP argues that one need NOT ‘design for change’. What activity is used to ensure the high quality of the design?