Software Engineering Processes

Course Code: XB_0089

Fernanda Madeiral & Claudia Raibulet

Computer Science Department
Vrije Universiteit Amsterdam

VU

Bachelor in Computer Science, June 3rd 2024

Part 1: Software Engineering Definitions & Overview

A Short History

- 1968: “software engineering” terminology
  - **Objective**: to address software crisis
  - **Motivation**: individual approaches to program development did not scale up to large and complex systems
- Since 1968 to present: new software engineering approaches were developed (e.g., structured programming, object-oriented programming)

Software Engineering (SE)

- Definition:
  - Software engineering is an engineering discipline that is concerned with **all aspects of software production** from the early stages of system specification until its maintenance.
  - SE is concerned with **professional software development**.
- Main activities:
  - Software specification
  - Software development
  - Software validation
  - Software evolution

Are These Activities SE?

- Development of **tools**, **frameworks**, etc. to support/easy software development
- **Management** of the software development
- Specification of **steps, rules**, **processes**, etc. to develop software
- …

Importance of SE

- **Software is everywhere!**
- More and more individuals and society rely on software systems.
- Usually, it is cheaper (in terms of time and costs) to develop software using SE methods and techniques rather than just writing code (as in a personal programming project).

  => We should produce **software** characterized by a high **quality**!

General Issues of Software

- **Heterogeneity**
  - Software systems are distributed and include different types of computer and mobile devices.
- Business and social **change**
  - Business and society are changing quickly. Also their software needs to change quickly (i.e., change existing software and rapidly develop new software).
- **Security** and **trust** and **privacy**
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

How Can We Address These Issues?

Software Engineering Fundamentals

Common principles to all software

- Software should be developed using a managed and understood development **process**. Of course, different processes are used for different types of software.
- **Dependability** (e.g., availability, reliability, maintainability) and **performance** are important for all types of system.
- Understanding and managing the **software specification** and **requirements** (what the software should do) are important.
- Where appropriate, **reuse software** already developed rather than write new software.

Examples of Software Systems – What SEP to Use?

- Example 1: **Software Design Project**
  - Teams of 3/4 students
  - Time: 2 months
- Example 2: **Air Traffic Management**
  - Teams of 100 software engineers distributed all over the world
  - Time: 2 years
- Example 3: **Web Application for a New University**
  - Teams of 20 software engineers
  - Time: 6 months
- Example 4: **Lego Store**
  - Teams of 50 software engineers distributed all over the world
  - Time: 1 year

SEP & Other Courses

Analysis

Design

Implementation

Requirements Engineering
- Requirements identification
- Requirements specification

Software Design
- Design patterns
- UML based modeling

Programming
- Java
- C/C++

Software Engineering Processes

Part 2: Software Engineering Processes

What Is a Software Engineering Process?

- Definition of a process:
  - A series of actions or steps taken in order to achieve a particular end [Oxford Dictionary].

- Definition of a **Software Engineering Process** (SEP):
  - A **structured set of activities** required to **develop a software system** [Ian Sommerville].

Common Activities to All SEP

- **Specification** – What should the software do?
- **Design** – How should the software do it?
- **Implementation** – Do it!
- **Test and Validation** – Check that the software does what the customer asked!
- **Maintenance and Evolution** – Changing the software over time according to customer requests

Software Process Model

- Definition:
  - A software process model is an abstract representation of a process.

- It presents a description of a process from a particular perspective.
  - E.g., activities in the model and their relationships

Software Process Descriptions

- **Activities** – what steps are defined by the model
- **Products** – which are the outcomes of a process activity
- **Roles** – which are the responsibilities of the people involved in the process
- **Pre-** and **post-conditions** – which are the statements true before and after a process activity has been enacted or a product produced

Types of Software Process Models

Plan-driven

- All process activities are planned in advance
- Progress is measured against this plan

Agile

- Planning is incremental and it is easier to change the process to reflect changing customer requirements

- **No right or wrong software processes!**
- In practice, **applied processes include plan-driven and agile elements.**

Part 3: Software Process Models

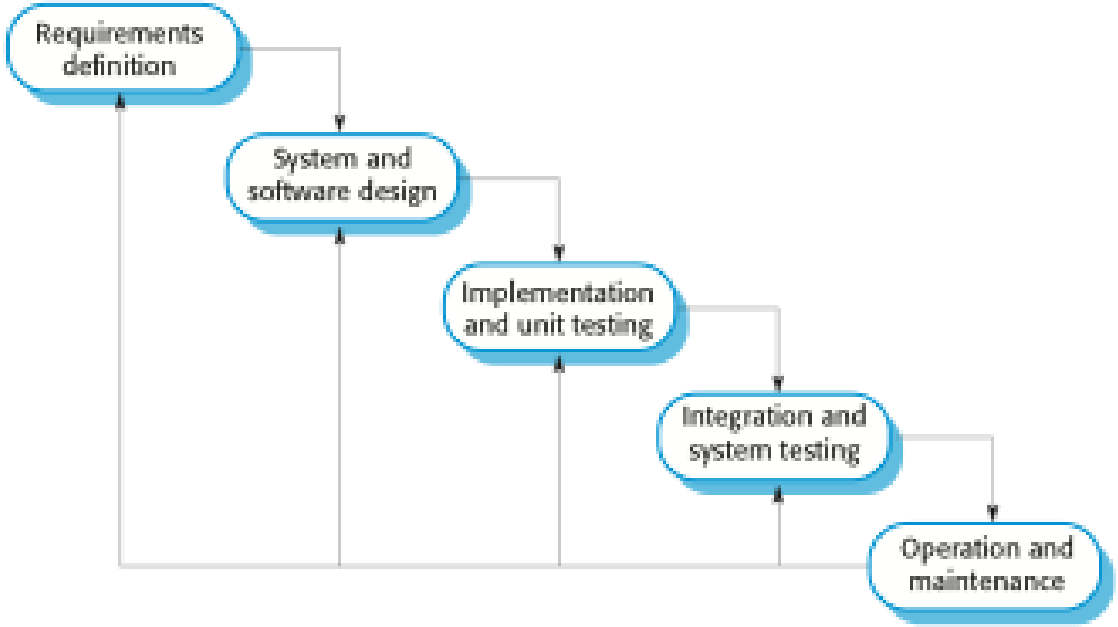
Software Process Models

- **Waterfall** (software life cycle)
  - Plan-driven
  - Sequential activities
  - Separate and distinct activities

- **Incremental** development
  - Activities are interleaved
  - Plan-driven or agile
- **Reuse-oriented** software engineering
  - Software is assembled from existing components
  - Plan-driven or agile

In practice, most large systems are developed using a process that incorporates activities from all these models.

The Waterfall Model



Analysis

Design

Implementation

Deployment

Maintenance

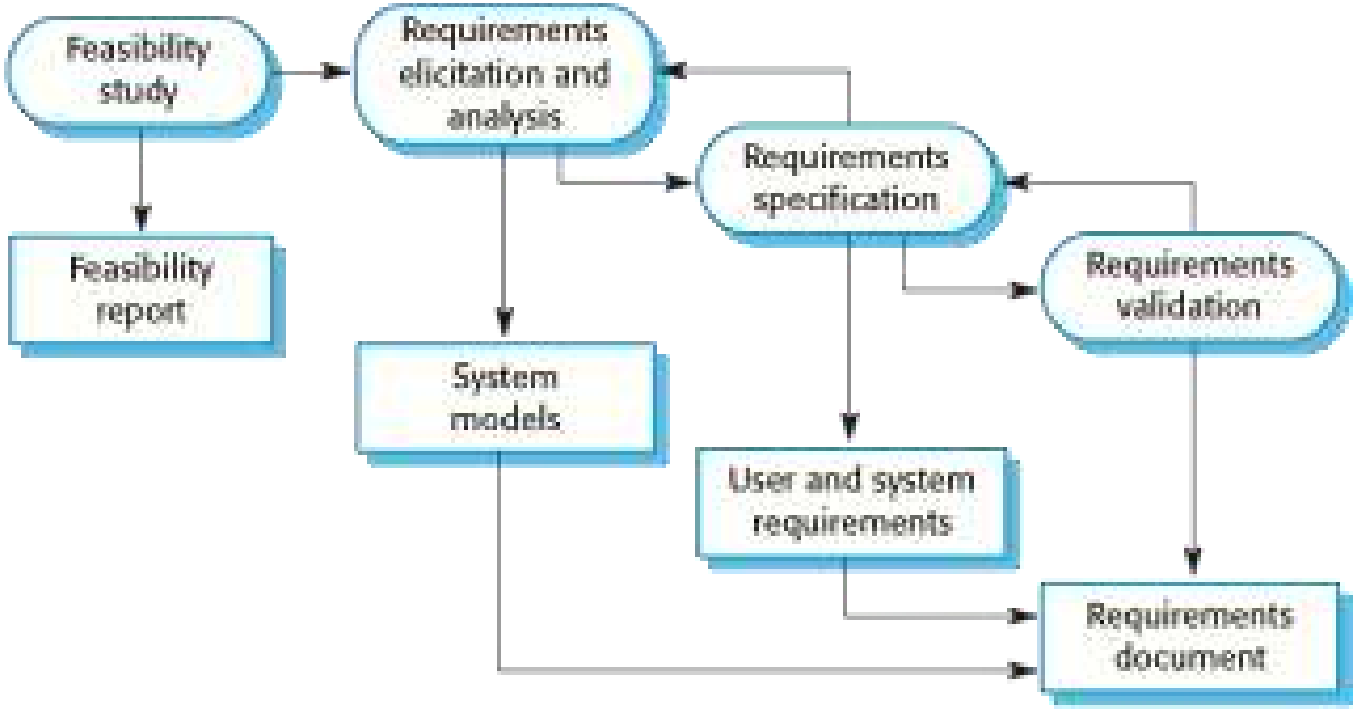
Waterfall

- **Analysis** – requirements analysis and specification
- **Design** – solution proposal and description
- **Implementation and unit testing** – coding and testing
- **Integration, system testing and deployment**
- **Maintenance and evolution** – problems fixing and addition of functionality
- **Important:** each activity should be finished before the next one starts!

Waterfall - Analysis

- Establishes the required functionality and constraints on the software operation and development
- Requirements engineering process
  - **Feasibility study**: Is it technically and financially feasible to build the system?
  - **Requirements elicitation and analysis**: What do the stakeholders require or expect from the software?
  - **Requirements specification**: Defining the requirements in detail.
  - **Requirements validation**: Checking the validity of the requirements

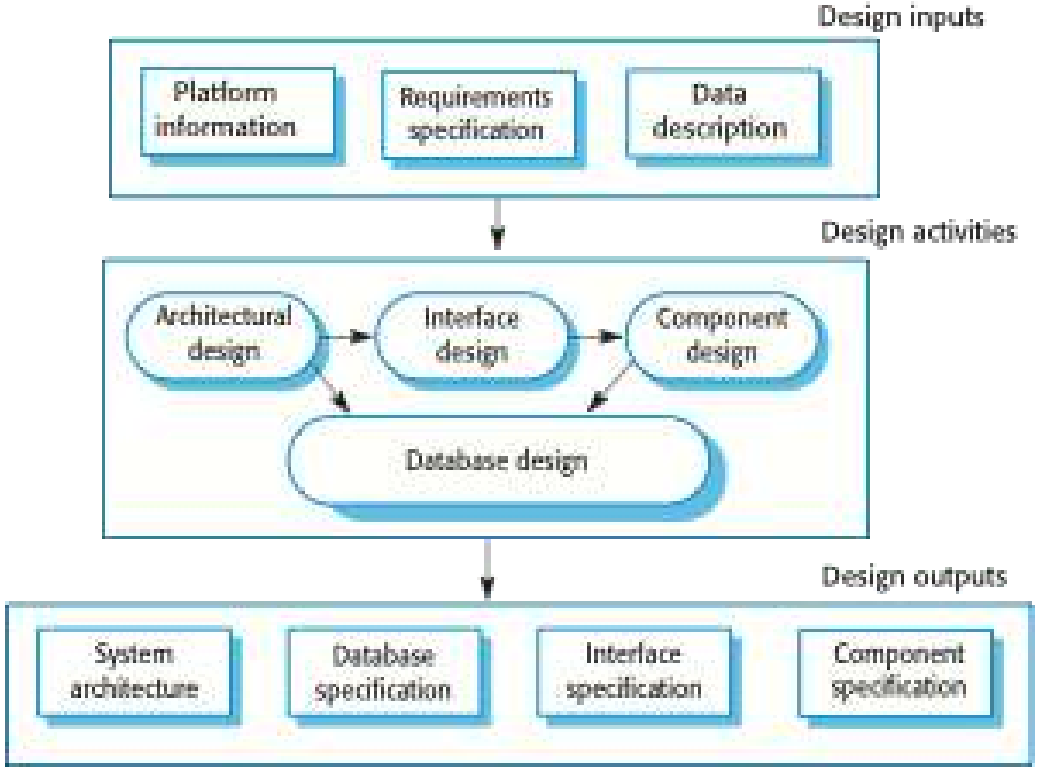
Waterfall - Analysis



Waterfall - Design

- Design a solution meeting the requirements specification
- Translate the requirements specification into a solution
- Design a software to meet the specification

Waterfall - Design



Waterfall - Implementation

- Translate the design into an executable program

- Observation: The activities of design and implementation are closely related and may be inter-leaved.

Waterfall – Integration, Testing and Deployment

- Verification and validation (V & V) – the software conforms to its specification and meets the requirements of the customer
- Testing involves executing the system with test cases
- Testing is the most commonly used V & V activity.

Waterfall - Testing

Waterfall – Maintenance and Evolution

- Solve bugs and other problems
- Improve the existent functionality
- Extend the software with additional functionality

Waterfall: Advantages and Limitations

- Advantages:
  - Applied where requirements are stable
  - Useful for work coordination in large projects where the development is done at several sites
  - Easy to follow the progress
- Limitations:
  - Does not cope with changes in requirements
  - Not flexible

Software Process Models

- **Waterfall** (software life cycle)
  - Plan-driven
  - Sequential activities
  - Separate and distinct activities
- **Incremental** development
  - Activities are interleaved
  - Plan-driven or agile
- **Reuse-oriented** software engineering
  - Software is assembled from existing components
  - Plan-driven or agile

In practice, most large systems are developed using a process that incorporates activities from all these models.

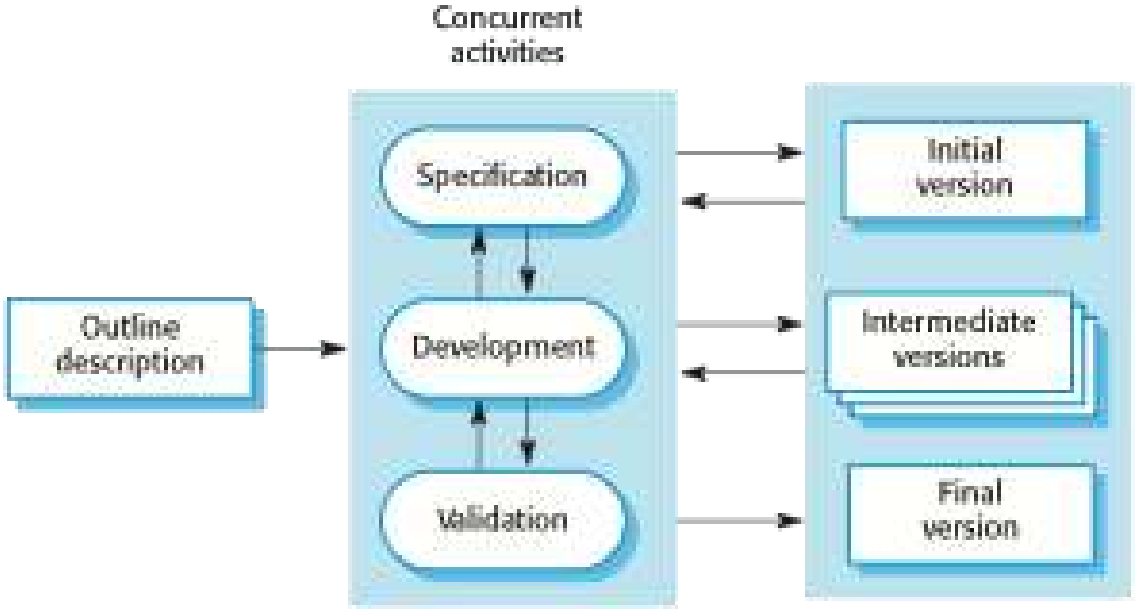
Incremental Development

- Objective: cope with **changes**
  - Changes in requirements
  - Changes in technology
  - …
- Develop the system in increments
- Evaluate each increment before proceeding to the next increment
- Normal approach used in agile methods
- Evaluation done by user/customer proxy

Incremental Development

- Requirements are **prioritised**
- The **highest priority requirements** are addressed in **early increments**
- During the development of an increment, **requirements are frozen**
- Other requirements for later increments can evolve

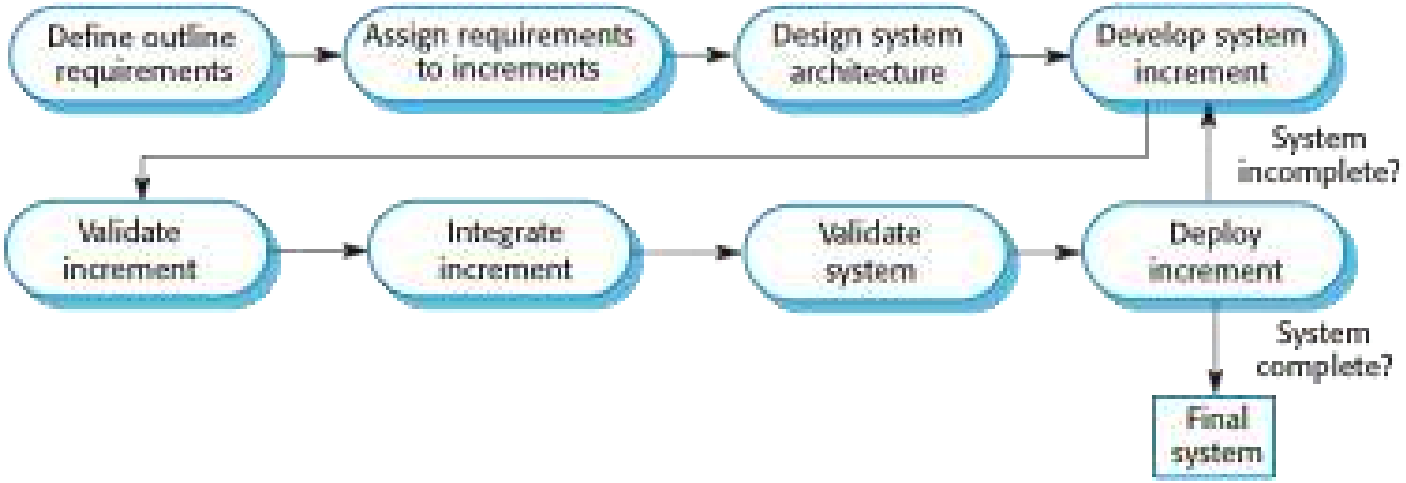
Incremental Development



Incremental Delivery

- Deploy an increment for use by end-users
- More realistic evaluation about practical use of software
- Difficult to implement for replacement systems as increments have less functionality than the system being replaced

Incremental Delivery



Incremental Development Advantages

- **Customer value** delivered with each increment
- **Functionality** is available **earlier**
- Early increments seen as a **prototype** to help elicit requirements for later increments
- **Lower risk** of overall project **failure**
- The **highest priority functionality** receives the most **testing**

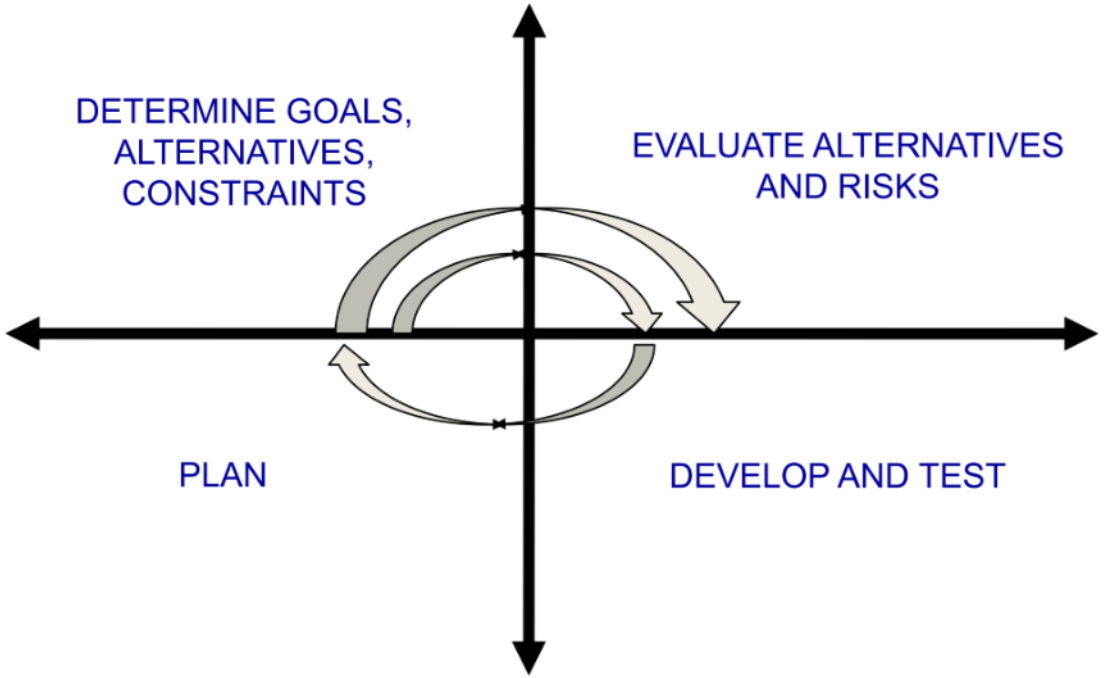
Incremental Development Limitations

- The process is not visible
- The structure/architecture tends to degrade by adding new increments
- Most systems require basic facilities used by different functionality
  - Requirements not defined in detail until an increment is implemented -> hard to identify common facilities for all increments
- The essence is that the specification is developed with the software
  - This conflicts with the procurement model of many organizations, where the complete specification is part of the development contract

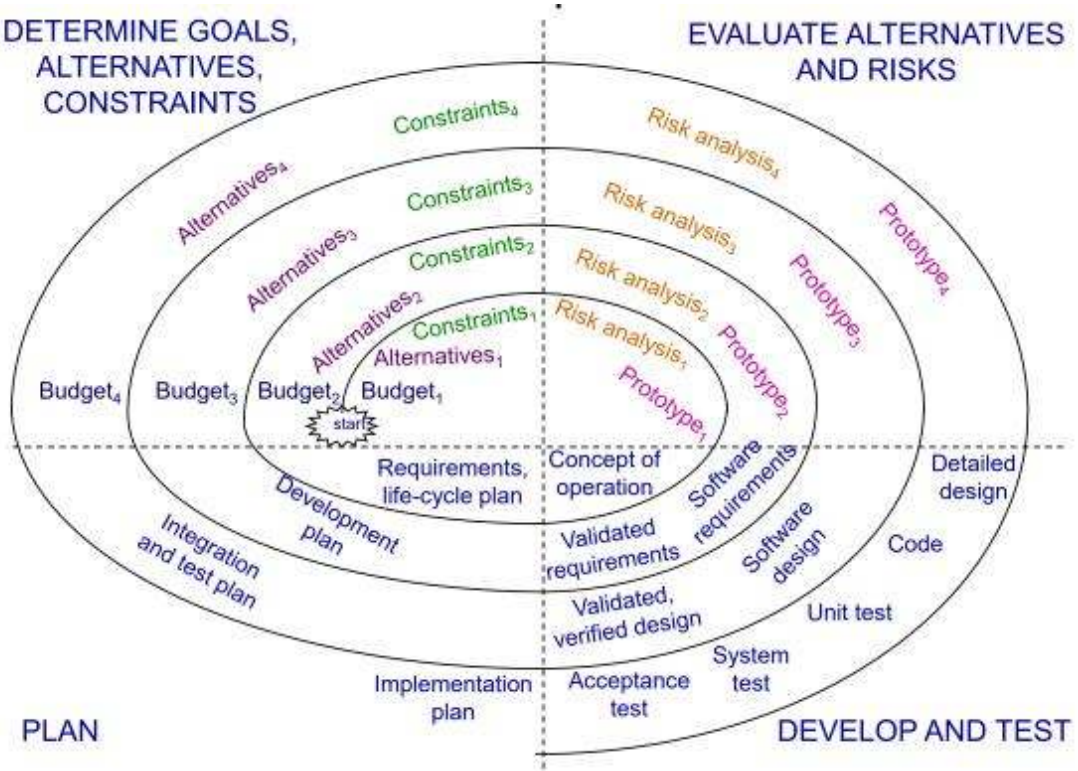
Boehm's Spiral Model

- Represented as a spiral
- Each loop in the spiral represents a phase in the process
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process

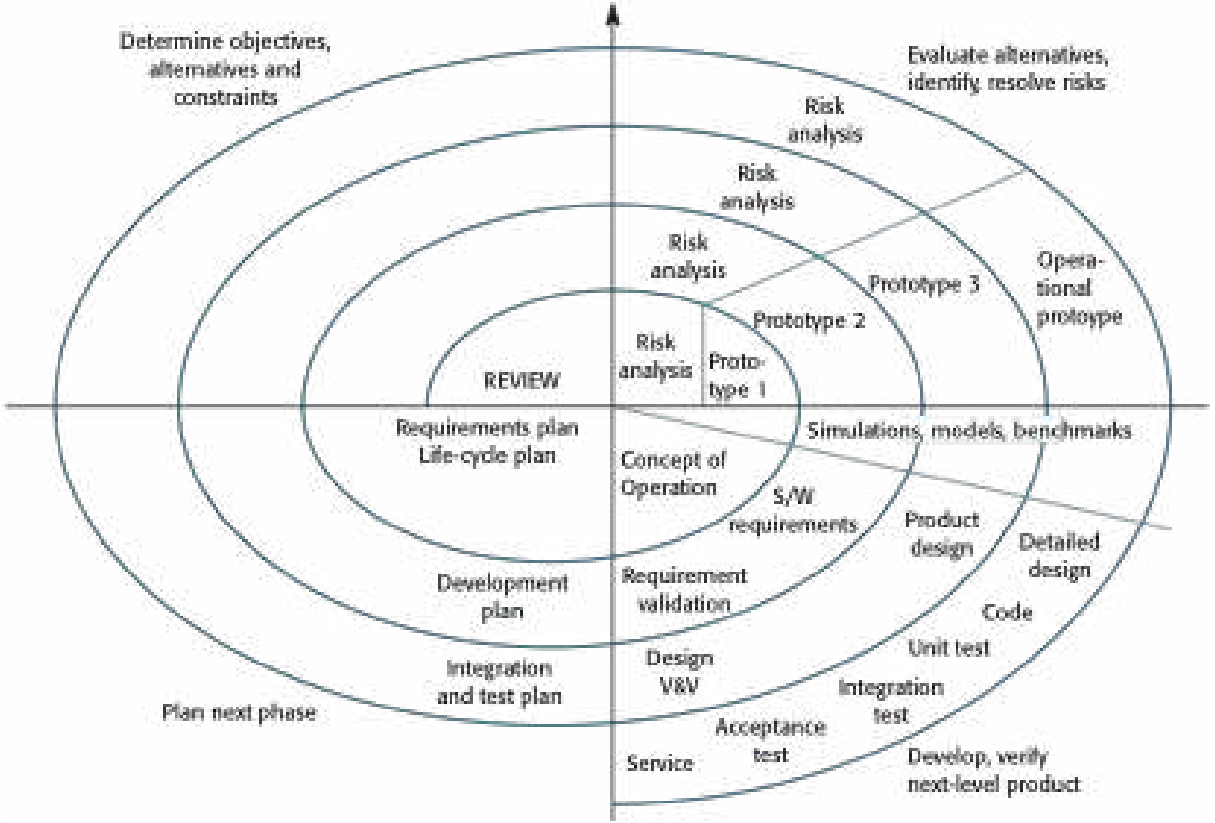
Boehm's Spiral Model



Boehm's Spiral Model



Boehm's Spiral Model



Spiral Model Sectors

- Objective setting
  - Specific objectives for the phase are identified
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase is planned

Spiral Model Usage

- Introduced **iteration** in software processes
- Introduced the **risk-driven** approach to development
- In practice, however, the model is rarely used as published for practical software development.

Software Process Models

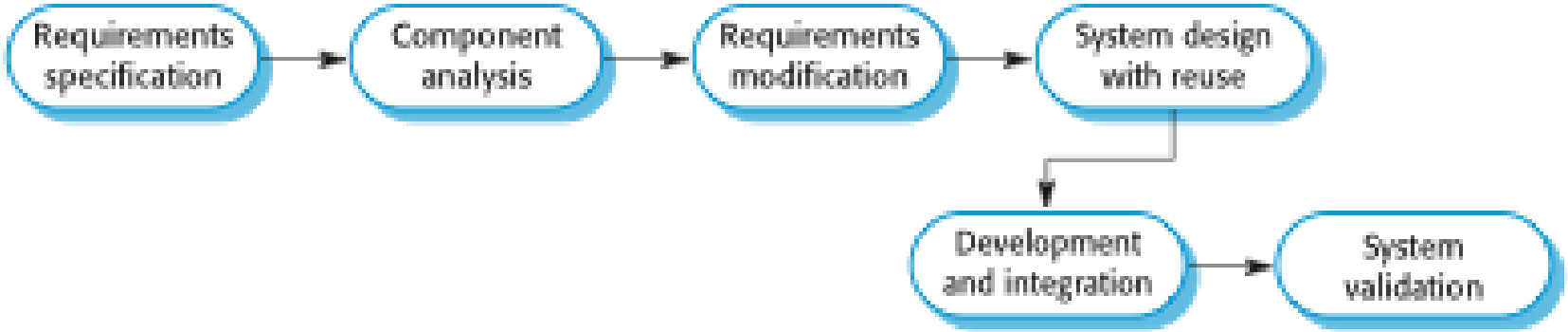
- **Waterfall** (software life cycle)
  - Plan-driven
  - Sequential activities
  - Separate and distinct activities
- **Incremental** development
  - Activities are interleaved
  - Plan-driven or agile
- **Reuse-oriented** software engineering
  - Software is assembled from existing componer
  - Plan-driven or agile

In practice, most large systems are developed using a process that incorporates activities from all these models.

Reuse Oriented SE

- Systematic reuse of available software
- Systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems
- Process steps:
  - Component analysis
  - Requirements modification/adaptation
  - System design with reuse
  - Development and integration
- Reuse is now the standard approach for building many types of business system

Reuse Oriented SE



Reuse Oriented SE

- Examples:
  - **Web services** developed according to service standards and available for remote invocation.
  - Collections of objects developed as a package to be integrated with a component **framework** such as .NET or J2EE.
  - **Stand-alone software systems** (COTS) configured for use in a particular environment

Incremental vs Iterative

- Similarities/Differences?
- Advantages/Limitations?

Examples of Software Systems – What SEP to Use?

- Example 1: **Software Design Project**
  - Teams of 3/4 students
  - Time: 2 months
- Example 2: **Air Traffic Management**
  - Teams of 100 software engineers distributed all over the world
  - Time: 2 years
- Example 3: **Web Application for a New University**
  - Teams of 20 software engineers
  - Time: 6 months
- Example 4: **Lego Store**
  - Teams of 50 software engineers distributed all over the world
  - Time: 1 year

To Do

Your TO DO List for the 1st Lecture:

- Enroll into groups
- Read the study material

Reading – For the 1st Lecture

- Exam material:
  - Ian Sommerville, Software Engineering, 9th or 10th edition – Chapter 1 and Chapter 2 (except Software engineering ethics section)
  - Spiral model: https://www.ou.nl/documents/40554/349790/IM0303_02.pdf
- Additional reading (highly recommended):
  - T Bhuvaneswari, S Prabaharan, A Survey on Software Development Life Cycle Models, IJCSMC, Vol 2, Issue 5, May 2013, pp. 262-267
  - Iqbal H. Sarker, Faisal Faruque, Ujjal Hossen, Atikur Rahman, A Survey of Software Development Process Models in Software Engineering, International Journal of Software Engineering and Its Applications, Vol. 9, No. 11, pp. 55-70

Takeaways?