Peter Mileff PhD

SOFTWARE ENGINEERING

The Basics of Software Engineering

University of Miskolc
Department of Information Technology

Introduction

- Péter Mileff - Department of Information Engineering
  - Room 210.
  - Email: mileff@iit.uni-miskolc.hu
    - **main contact form**
- **Software Engineering is mainly a theoretical course**
  - No practical lessons
    - We learn the theory of making better softwares
- We use only the second two lessons (10am-12am)
  - the second part (10-12) would be the practice
  - Practical lessons are not feasible at the University
    - there will be homeworks :)

Important: lectures will be every two weeks

Course requirements

Task 1: make a Software Specification

- Imagine your new software
  - Can be anything: Game, Social site, Image converter, Music player, Mobile application, Operating System, etc
- Collect any requirements, limitations, considerations about this software
  - Formulate them into a Documentation.
- Task can be performed in groups
  - max. 2 persons in a group

- The expected size of the document
  - ~10 pages / person

Recommended Books

- Ian Sommerville:
  - Software Engineering 10th Edition, 2015.
- A. Bijlsma, B.J. Heeren, E.E. Roubtsova, S. Stuurman:
  - Software Architecture, 2011.
- Ivan Marsic:
  - Software Engineering Textbook, 2009.

Software engineering history…

The Sixties - the 60s

- Typically very early computers
  - rudimentary hardware infrastructure
- **Problems to be solved:**
  - only specific problems
  - small programs
- **Developer?**
  - Special skilled person (researcher)
    - no teams, typically only one person
- **Development tool:**
  - mainly Assembler or Memory map (machine code) was used to make programs
    - lack of high level programming languages

The Software crisis

- **Preliminaries:**
  - falling hardware prices
    - more and more people/company were able to buy a computer
  - increasing hardware performance
  - increasing demand for softwares

- **The problem:**
  - more software was needed
  - there was a sharp increase in the cost of software
  - **the software quality was not sufficient!**

The 70s

- The first high-level programming languages appeared
  - Algol, Fortran, Cobol
- The programming is becoming a profession
- **Developers realized:**
  - more effective programming tools are necessary
    - high level tools
  - a systematic approach requires to develop for making better software
- **Result:** The the first programming methods was developed
  - structural and modular programming

The born of Software technology

The software…

What is software?

- There is no exact definition
- Software is more than just a program code!
  - A program is an executable code, which serves some computational purpose
    - E.g. totalcommander.exe
- Software is considered to be:
  - collection of executable programming code,
  - associated libraries and documentations
  - data and configuration files
    - e.g. totalcommander has also help and config files
- Software, when made for a specific requirement is called **Software product**

Software products

- **Generic products**
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
    - Examples – PC software such as editing, graphics programs, project management tools; CAD software; Games, Voip applications, etc.
- **Customized products**
  - Software that is commissioned by a specific customer to meet their own needs
    - E.g. embedded control systems, air traffic control software, traffic monitoring systems
- Important difference: Who writes the specification?
  - Generic products: the organization that develops the software
  - Customized products: the organization that is buying the software
- The border is often blurred

Other considerations

- Software itself is an IT-industry product
  - It can be compared to other industrial products
    - E.g: Keyboard, Table, Door, etc
  - But Software is different:
    - It is more complicated (consist of algorithms, complex program codes)
    - It does not wear out with use
    - It does not need to be repaired like a table, or bicycle
    - Instead it becomes outdated (E.g. browsers)
  - Software needs continuous development
    - Due to the continuous development of hardware devices and operating systems
    - Because new user requirements arise during usage
      - E.g. a website needs facebook login function

Why is software engineering needed?

- **Developing a software is a complex process**
  - Usually performed by a team
- **It has multiple stages**
  - E.g: Analysis, Design, Implementation, etc

- **Without managing the developing process**
  - The project may fail
  - The whole development process may result in chaos
  - The product will have poor quality
    - bugs
    - rely on bad design concepts

What is Software engineering?

- **Software is costly**
  - Software costs often dominate computer system costs
  - Software costs more to maintain than it does to develop.
  - For systems with a long life, maintenance costs may be several times higher than development costs.
- **Software engineering:**
  - It is an engineering discipline that is concerned with all aspects of software production
  - It is a collection of different models and concepts
  - it is concerned with cost-effective software development.
- **The main objective is:**
  - make the development process more effective
  - improve software quality

The software process…

The Software process

- **A software process is a sequence of activities that leads to the production of a software product.**
- It can be very complex
  - a lot of activities, developers, processes, etc
- It highly depends on human activities
- There is no ideal, suitable development process for every software
  - The process is different at every organization!
    - different members, different rules, different environment, stb
- **Problem:** it cannot really automate with CASE tools
  - e.g: robots cannot write softwares yet

The Software process

- **Objective:**
  - It should be designed to exploit the capabilities of the people in the organization and the characteristics of the development environment.

Example:

- For some systems, such as **critical systems**
  - a very structured development process is required.
    - Because requirements do not change

- **For business systems**
  - a less formal, flexible process is likely to be more effective.
    - Because the requirements may change rapidly

Fundamental Activities

- **Software specification:**
  - where customers and engineers define the software that is to be produced and the constraints on its operation.
- **Software design and implementation:**
  - where the software is designed and programmed.
- **Software validation:**
  - where the software is checked to ensure that it is what the customer requires.
- **Software evolution:**
  - where the software is modified to reflect changing customer and market requirements.

The software process models...

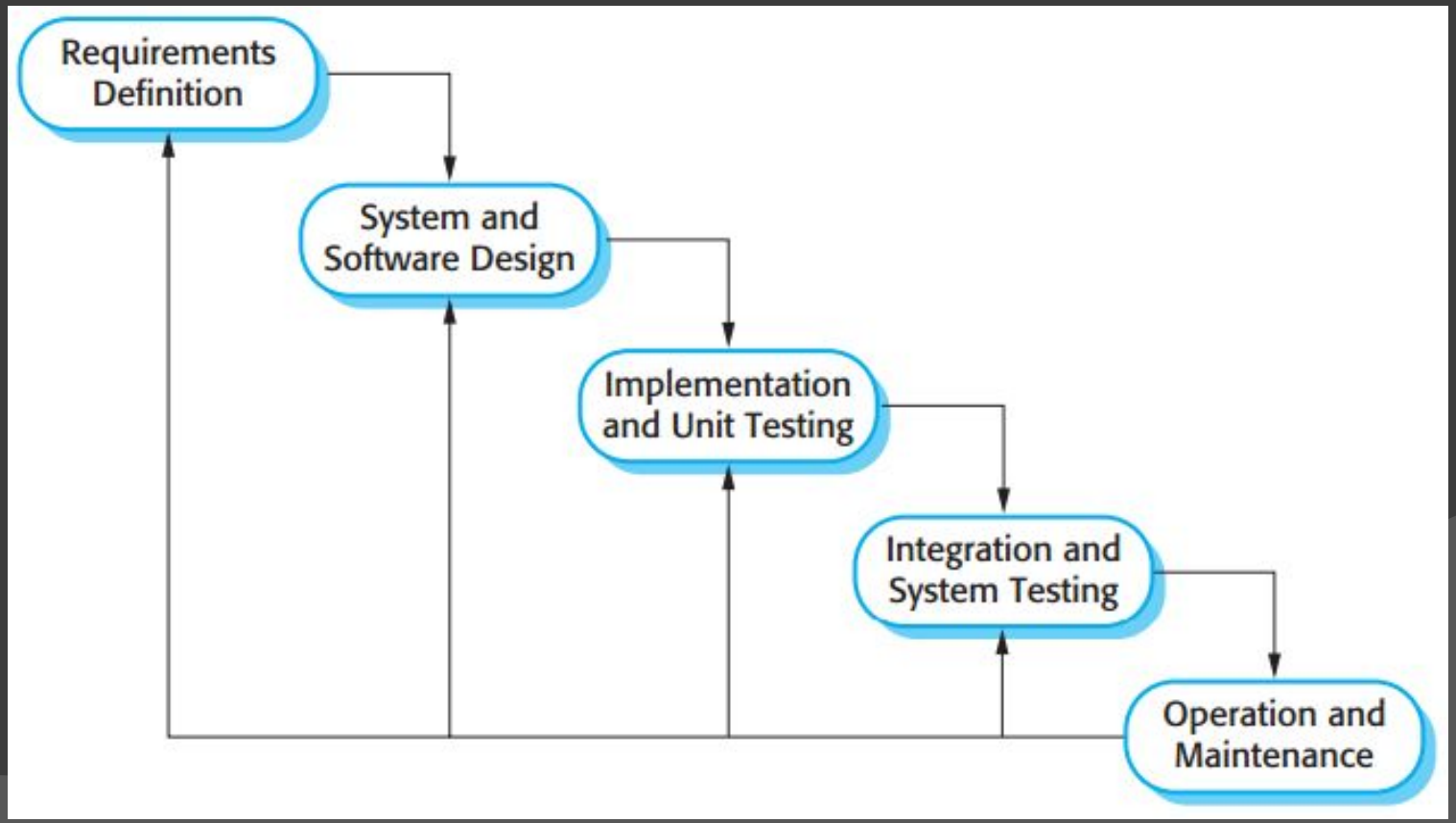
What is a Software process model?

- **A software process model is a simplified representation of a software process.**
- Each process model represents a process from a particular perspective
  - thus provides only partial information about that process.
- **There are generic models**
  - they are not definitive descriptions of software processes
  - they are abstractions of the process that can be used to explain different approaches to software development

Well known models…

Waterfall model

- The first Process Model to be introduced.
  - It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use.



Stage 1: Requirements analysis and definition

- The system's services, constraints, and goals are established
  - by consultation with system users and the customer
- They are then defined in detail and serve as a **system specification**

Stage 2: System and software design

- Software design involves
  - identifying and describing the fundamental software system abstractions
  - describe the system components and their relationships
- Establish the overall architecture of the system
  - What type of architecture will be used?
    - E.g: p2p, client-server, webservices, etc.
  - Determine the subsystems and their relationships
  - How the subsystems will be controlled?
    - centralized or decentralized approach

Stage 3: Implementation and unit testing

- During this stage, the software design is realized
  - as a set of programs or program units.
    - applying programming languages
    - and tools (IDE, Compiler, Debugger, etc)

- **The objective of unit testing**
  - to verifying that each unit meets its specification.

Stage 4: Integration and system testing

- Integrate the individual program units or programs
  - such as playing “Lego”
- All the programs are tested as a complete system
  - to ensure that the software meets the requirements
- After testing, the software system can be delivered to the customer

Stage 5: Operation and maintenance

- **Objective:**
  - Install the system and put into practical use.
- Normally this is the longest lifecycle stage.
- **Maintenance involves**
  - correcting errors which were not discovered in earlier stages,
    - bugfix
  - improving the implementation of system units
    - E.g: long response time
  - enhancing the system's services as new requirements are discovered.
    - E.g: customer needs a new button

Advantages

- This model is simple and easy to understand and use.
  - documentation is produced at each stage
- It is easy to manage due to the rigidity of the model
  - each stage has specific deliverables and a review process.
- In this model stages are processed and completed one at a time.
  - Stages do not overlap each other
- Waterfall model works well
  - for smaller projects
  - where requirements are very well understood
    - E.g: airplane control system

Disadvantages

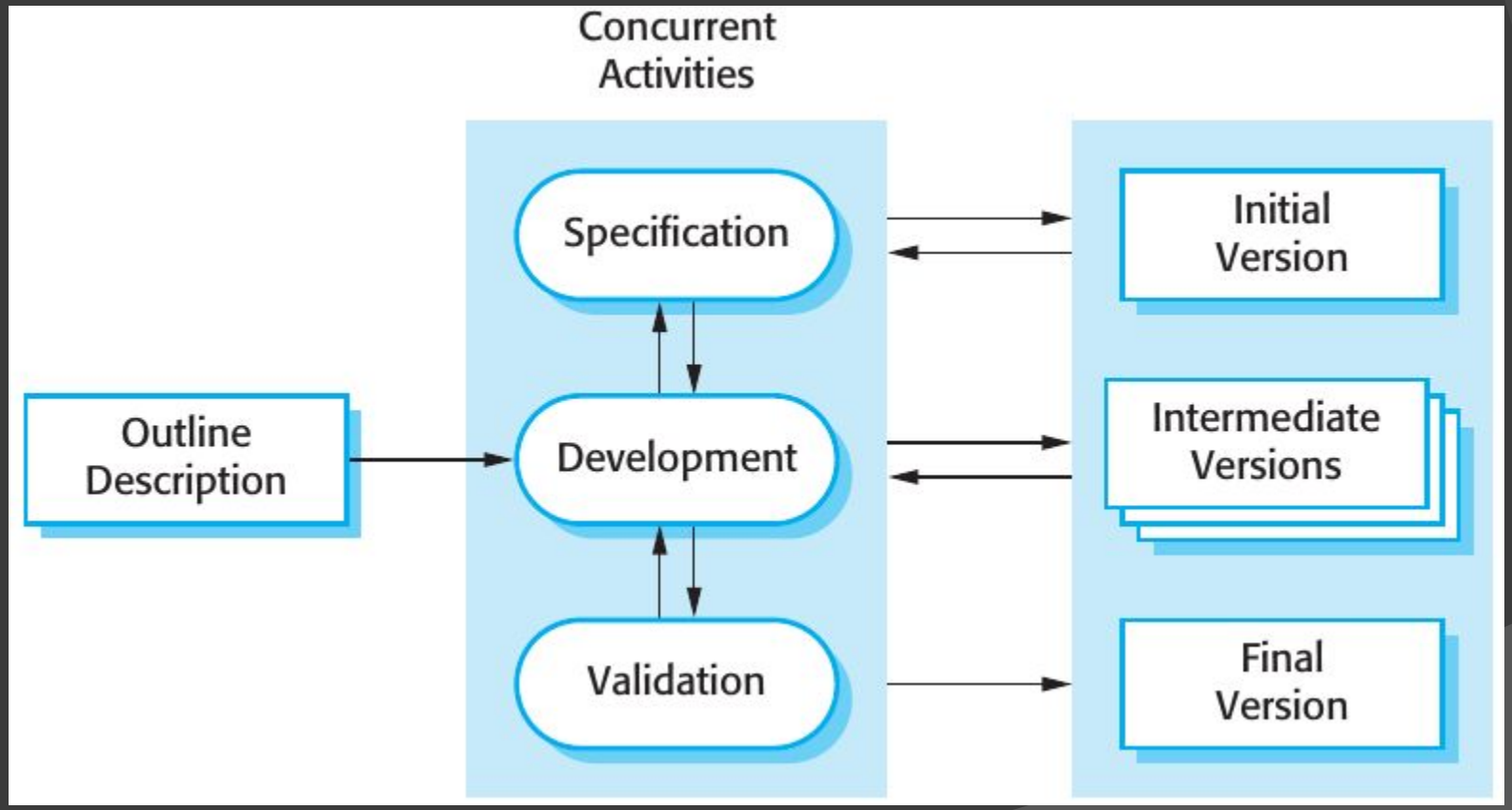
- In the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
  - results inflexibility
- No working software is produced until late during the life cycle.
- The model has high amounts of risk and uncertainty
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

Incremental development...

Incremental development

- **Basic conception:**
  - developer team creates an initial implementation
  - review this basic implementation
    - with users and customer
  - evolving it through several versions until an
  - adequate system has been developed
- **This is a different approach:**
  - better enforce parallelism between activities
  - and rapid feedbacks

Incremental model



Incremental development

- The model is better than a waterfall approach
  - for most business, e-commerce, and personal systems
- It reflects the way like we solve problems
  - We rarely work out a complete problem solution in advance
- By developing the software incrementally
  - it is cheaper and easier to make changes in the software as it is being developed
  - Usually early increments include the most important features
    - Therefore these functions are more stable
- **Rapid feedbacks**
  - Due to continual iterations any feedback can be performed at any time

Disadvantages

- **From a management perspective it has two problems:**
  - **The process is not visible:** managers need regular deliverables to measure progress.
    - **Problem:** continuous iteration, lack of full specification
  - **The systems are often poorly structured:** regular changes tend to corrupt the system structure.
    - from time to time the system requires to be refactored, restructured
- It is suggested to use in case of:
  - short-life systems
  - small and medium-sized systems.
  - (~ 500,000 lines of code)

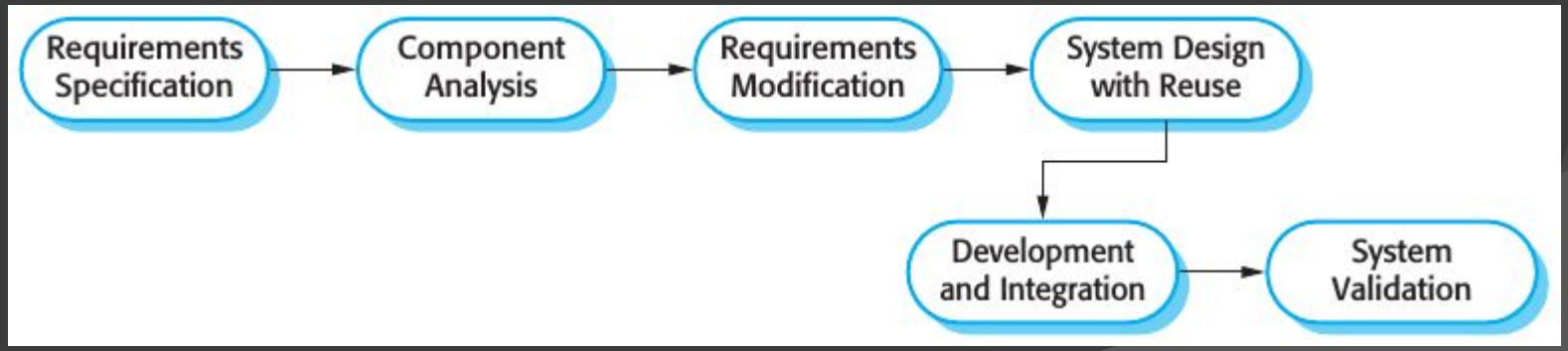
Reuse-oriented software engineering...

Reuse-oriented software engineering

- In majority of software projects, there is software reuse
  - It is natural, we wouldn't want to write everything from zero
    - it is cheaper and faster to reuse something
- **<u>This approach relies on</u>**
  - a large base of reusable software components
  - and an integrating framework for composition of these components

Reuse-oriented software engineering

- A general process model for reuse-based development
  - The initial requirements specification stage and the validation stage are comparable with other software processes,
  - the intermediate stages in a reuse-oriented process are different



Reuse-oriented software engineering

1. Component analysis

- A search is made for components
  - which are suitable to implement the specification
- Usually, there is no exact match
- **Problem:**
  - the components that may be used only provide some of the functionality required.

Reuse-oriented software engineering

2. Requirements modification (!)

- During this stage, the requirements are analyzed
  - we use information about the components that have been discovered.
  - They are often modified to reflect the available components.
- **Where modifications are impossible:**
  - the component analysis activity may be re-entered to search for alternative solutions.
  - Or decide to implement a new component in house

Reuse-oriented software engineering

3. System design with reuse

- During this phase, the framework of the system is designed
  - or an existing framework is reused
- **The designers take into account**
  - the components that are reused
  - and organize the framework to cater for this.
- Some new software may have to be designed if reusable components are not available.

Reuse-oriented software engineering

4. Development and integration

Objective: create a new system

- If a software component cannot be externally procured is developed in house
  - more time and money
- Components and COTS (commercial off-the shelf systems) systems are integrated
- System integration, in this model, may be part of the development process rather than a separate activity.

Advantages / Disadvantages

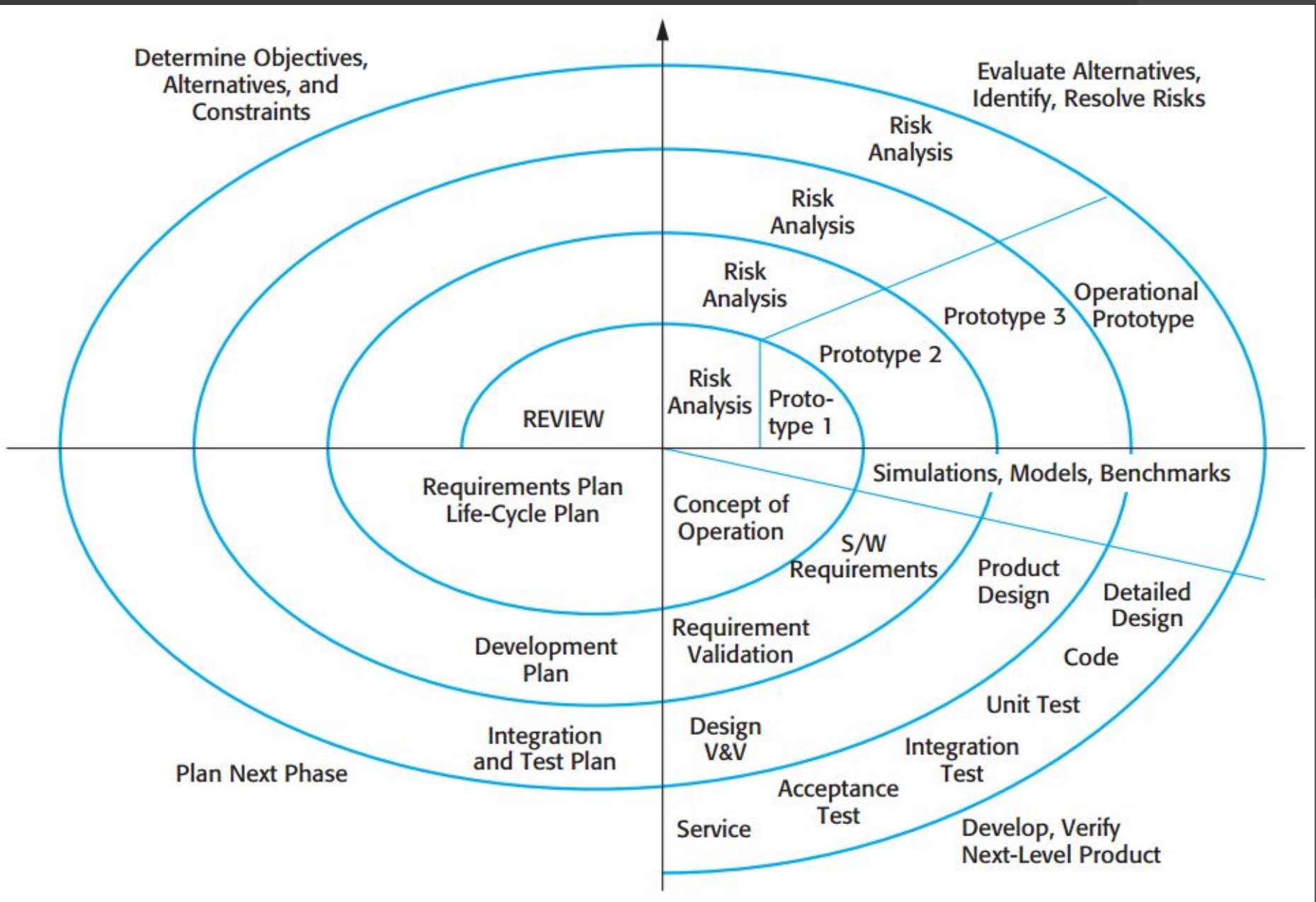
- Reduce the amount of software to be developed
  - reduce cost and risk!
- Usually leads to faster delivery of the software
- Requirements compromises are inevitable
  - this may lead to a system that does not meet the real needs of users
- Some control over the system evolution is lost
  - as new versions of the reusable components are not under the control of the organization using them.

The Spiral model...

The Spiral Model

- Proposed by Boehm in 1988
- It is a risk driven process framework
- **The software process:**
  - represented as a spiral
    - rather than a sequence of activities with backtracking
  - each loop in the spiral represents a phase of the software process
    - the innermost loop might be concerned with system feasibility
    - the next loop with requirements definition,
    - the next loop with system design, etc
- **The spiral symbolizes that the software development process never ends!**

The Spiral Model



The Spiral Model

- Each loop in the spiral is split into four sectors:

- **1. Objective setting:**
  - Specific objectives for that phase of the project are defined
  - Constraints on the process and the product are identified
  - A detailed management plan is drawn up.
  - Project risks are identified. Alternative strategies, depending on these risks, may be planned.

The Spiral Model

- **2. Risk assessment and reduction**
  - For each of the identified project risks, a detailed analysis is carried out.
  - Steps are taken to reduce the risk.
  - For example:
    - if there is a risk that the requirements are inappropriate, a prototype system may be developed
      - to better understand the requirements
      - to investigate, the requirements are feasible

The Spiral Model

- **3. Development and validation**
  - After risk evaluation, a development model for the system is chosen.
  - The software is developed in this phase
    - design, implementation, validation
  - This phase can contain a whole software development model like Waterfall or Component based development, etc.

The Spiral Model

- **4. Planning**
  - Closes the current spiral loop
  - **The project is reviewed** and a decision made whether to continue with a further loop of the spiral.
    - it is appropriate to continue?
      - objectives are good?
      - human and financial resources?
  - If it is decided to continue, plans are drawn up for the next phase of the project.

The Spiral Model

Why is this model different?

- The **risk** is an integral part of the model
  - not only a software process model, because risk handling is a project management activity
- Risk minimization is a very important:
  - risks lead to proposed software changes and project problems
    - such as schedule and cost overrun
- This model explicitly calculates with alternative ways to achieve objectives

Thank you for your attention!