Software Engineering
Course description

Bogumiła
Hnatkowska

About the course

- Lecturer: Bogumila Hnatkowska, PhD
- Mail: Bogumila.Hnatkowska@pwr.edu.pl
- Materials and recommended tools:
  - Board: https://eportal.ii.pwr.wroc.pl/w08
  - Login: student
  - Password: zima2018
- Exam dates (?): 25.06, 2.07

Course assessment rules

- Three forms: lectures, labs, classes
- All forms must be passed to pass the course
- Two tests are planned during the lectures: week 7th, and week 15th.
- Attendance to the tests is optional (exam can be taken instead)
- Final grade =

  0,3 * exam + 0,4 * lab + 0,3 * classes

Tools planned to be used within the course

- Text editor – for textual documents
- UML modelling tools (one of):
  - Visual Paradigm version 15.1 (community, standard)
- OCL modelling tools
- Eclipse or NetBeans (for java programmers), Visual Studio (for C++, C# programmers)

Literature

- Sommerville Ian, Software engineering, Addison-Wesley, 2007.
- Bruegge Bernd. Object-oriented software engineering: using UML, Patterns, and Java. Pearson/Prentice Hall, cop. 2004.
- Pfleeger Shari Lawrence. Software engineering: theory and practice. Pearson/Prentice Hall, 2006.
- http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf (free book, 2012)

Course objectives

- To present fundamental processes of software engineering
- To present modeling languages
- To prepare the students for:
  - reading project documentation written in commonly used modeling languages;
  - preparing basic software models;
  - preparing tests.
- To make the students familiar with CASE tools

Course content (general)

- Software life cycles
- Software requirements
- Software modeling: UML, OCL
- User interface prototyping
- Software verification and validation

Lecture 1

What is a software (software product)?

- Software is:
  - Computer programs that when executed provide desired function and performance to their users
  - Data pertaining to the operation of the program(s).
  - Documents that describe the operation and use of the program(s)

Participants in the software product

Customer

Needs, Money

Contractual obligations

Needs

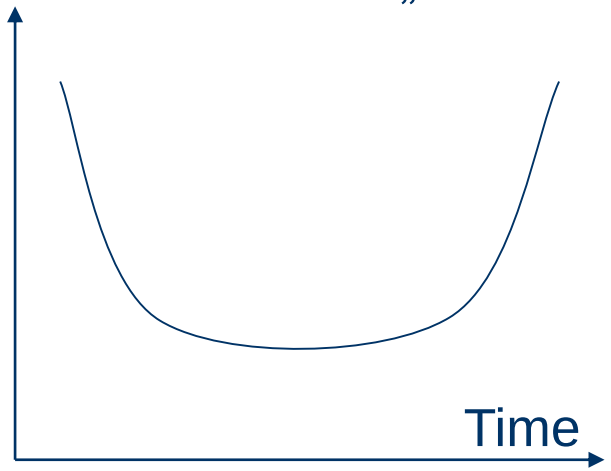
Software system

End-user

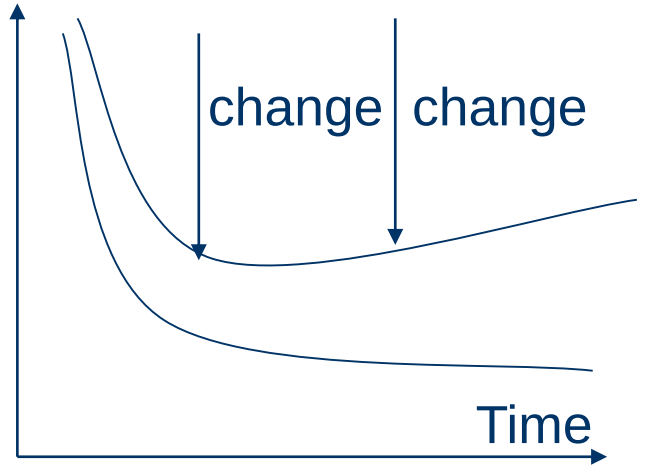
Developer

Why software is not a typical product?

- Software is developed or engineered; it is not manufactured in the classical sense
- Most software is custom-built, rather than being assembled from existing components
- Software doesn't „wear out"



Failure curve for hardware



Failure curve for software

Variety of software products

- System software: operating systems, compilers
- Real time: air traffic control
- Data processing: telephone billing, pensions
- Information systems: web sites, digital libraries
- Offices: text proc., spreadsheets
- Embedded systems: digital camera, GPS
- Communications: routers, mobile telephones
- Engineering & Scientific: simulations, CASE tools, CADs
- Graphical: film making, design
- Etc.

The question: how to build a software product in a proper way?

What is software engineering?

- The application of a systematic, disciplined, quantifiable approach to the **development, operation, and maintenance of software**; that is, the **application of engineering to software**

SWEBOK – Software Engineering Book of Knowledge

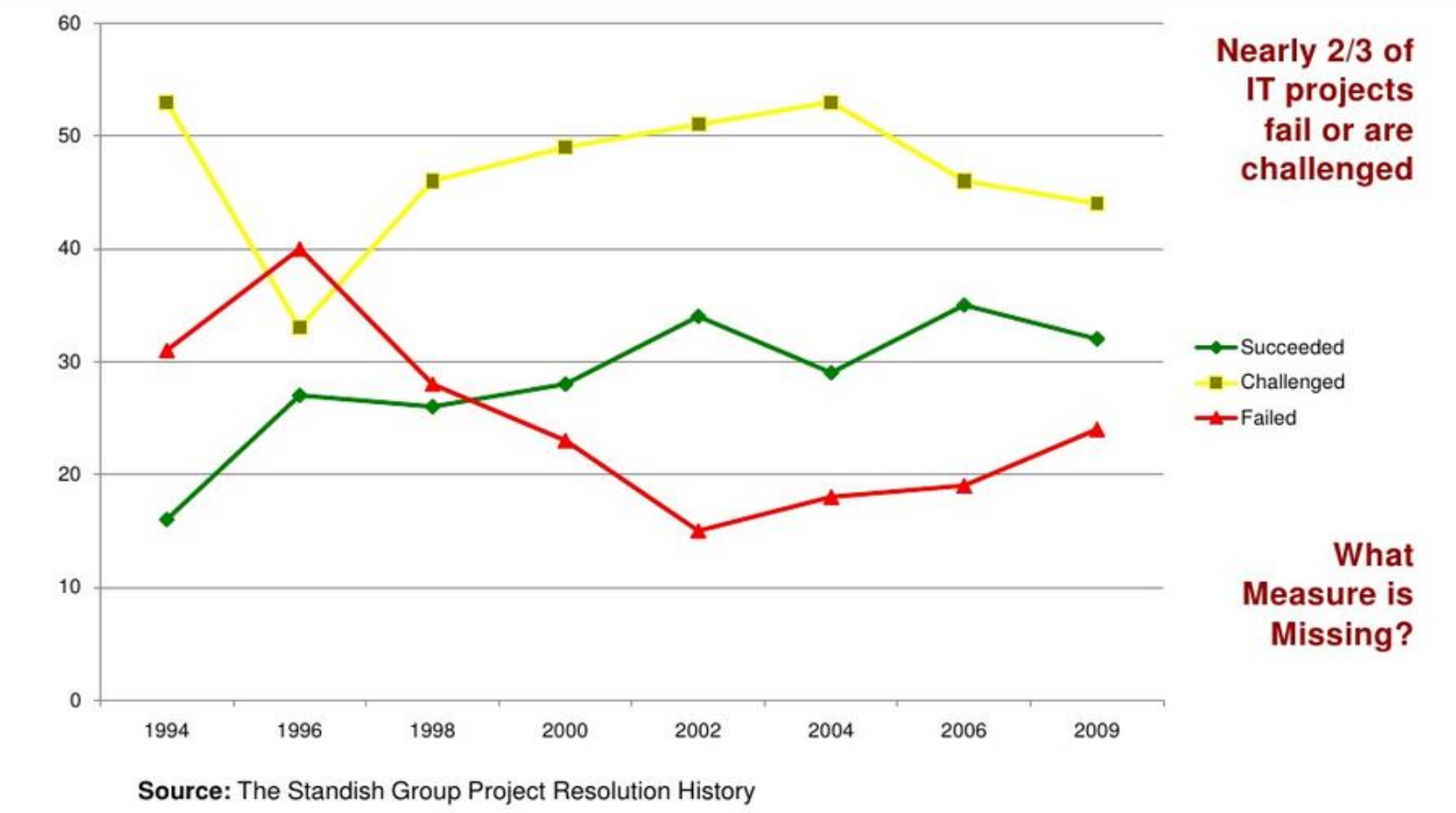
What is software engineering? cont.

- The aim of software engineering (SE) – to translate user needs into a good quality software product, i.e.:
  - Satisfying customer needs
  - Reliable
  - Usable
  - Effective
  - Easy to maintain

Why SE is important?

- Software development is expensive (the major costs are salaries)
- Every software project is a trade-off between:
  - Functionality
  - Resources (cost)
  - Schedule

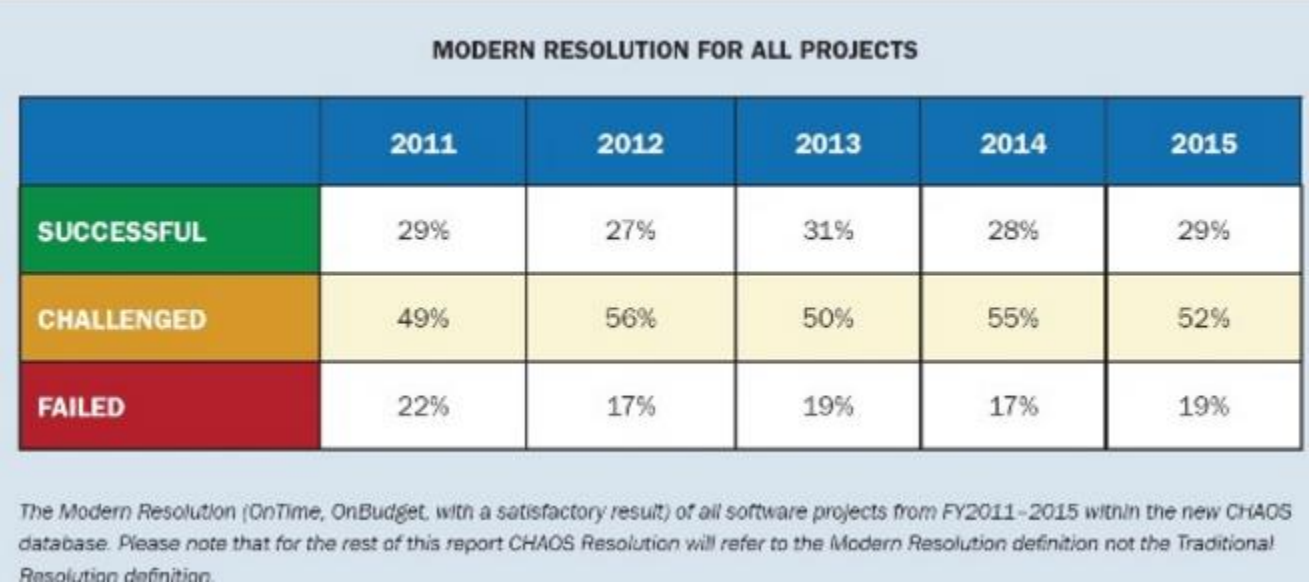
Why SE is important?



Source: The Standish Group Project Resolution History

Why SE is important?, cont.

- http://www.infoq.com/articles/standish-chaos-2015



MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

Craft of software development

- Software products are very varied
  - client requirements are different
  - many approaches to software development exists
  - many languages, operating systems, tools, databases exists
- A skilled software developer knows about a wide variety of approaches, methods, and tools.
- The craft of software development is to select appropriate methods for each project and apply them effectively.

What is software life cycle?

- The period from software conception until the software is retired is called ***software life cycle***
- Within this period people perform many activities which can been organized according to different models so called ***software life cycle models (or software development processes)***
- Typically software life cycle models ***are divided into phases and further sub-phases (activities, tasks)*** aiming with producing special kind of deliverables

What is software life cycle model?

- **A software lifecycle model** is a description of the set of activities carried out in an SE project, and the relative order of these activities.
- Software life cycle model (software development process) defines:
  - The majority process activities together with work products
  - The precedence relationship between process activities
  - Entry and exit criteria for each process activity

Main phases in SE

- Definition phase („what”):
  - What is the problem to be solved?
  - What information is to be processed?
  - What system function and performance are expected?
  - What validation criteria are required?
- Development phase („how”):
  - How will the solution be realized?
  - How will the software be constructed and tested?
  - How data are to be structured?
  - How functions are to be implemented as a software architecture?
- Maintenance phase („keep in operation”):
  - How the software will be supported over the long term, when corrections, adaptations, and enhancements are requested by users?

Activities within phases

- Definition phase activities (sub-phases):
  - (Feasibility study)
  - Requirements
  - Analysis
- Development phase activities (sub-phases):
  - Design
  - Coding
  - Testing
  - (Deployment) – in many life-cycle models this sub-phase is not addressed directly
- Additional managerial and quality activities:
  - Project management
  - Software configuration management
  - Quality assurance

Activities – brief description

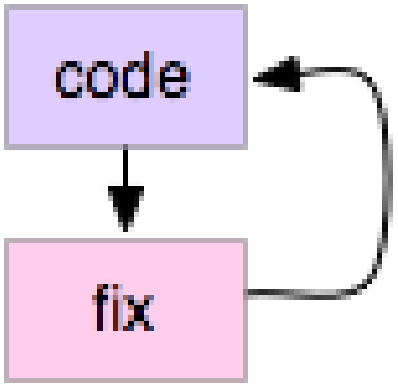
- Feasibility study – answers the question if there exist the possibility to build the system and what are the expected benefits
- Requirements – answers the question what the system should do and under what circumstances
- Analysis – answers the question about the key data processed by the system, system states and behavior
- Design – answers the question how the software should be built (shows its internal structure) in terms of selected implementation platform
- Coding – the software implementation according to design decisions
- Testing – the verification if the result of coding phase satisfies requirements
- Deployment – package code into a product or a project deliverable, install the product at the customer's sites
- Maintenance – aims in keeping the software alive in good health (updates and upgrades)

Software life cycle models

- Code and fix model
- Waterfall model
- Incremental Model (version: Incremental & Iterative Model)
- Prototyping Model
- Spiral Model

Code and fix

- No design
- No specifications
- Typical steps:
  - Recognize requirements
  - Write the code (Code)
  - Test the program and Fix errors (Fix)



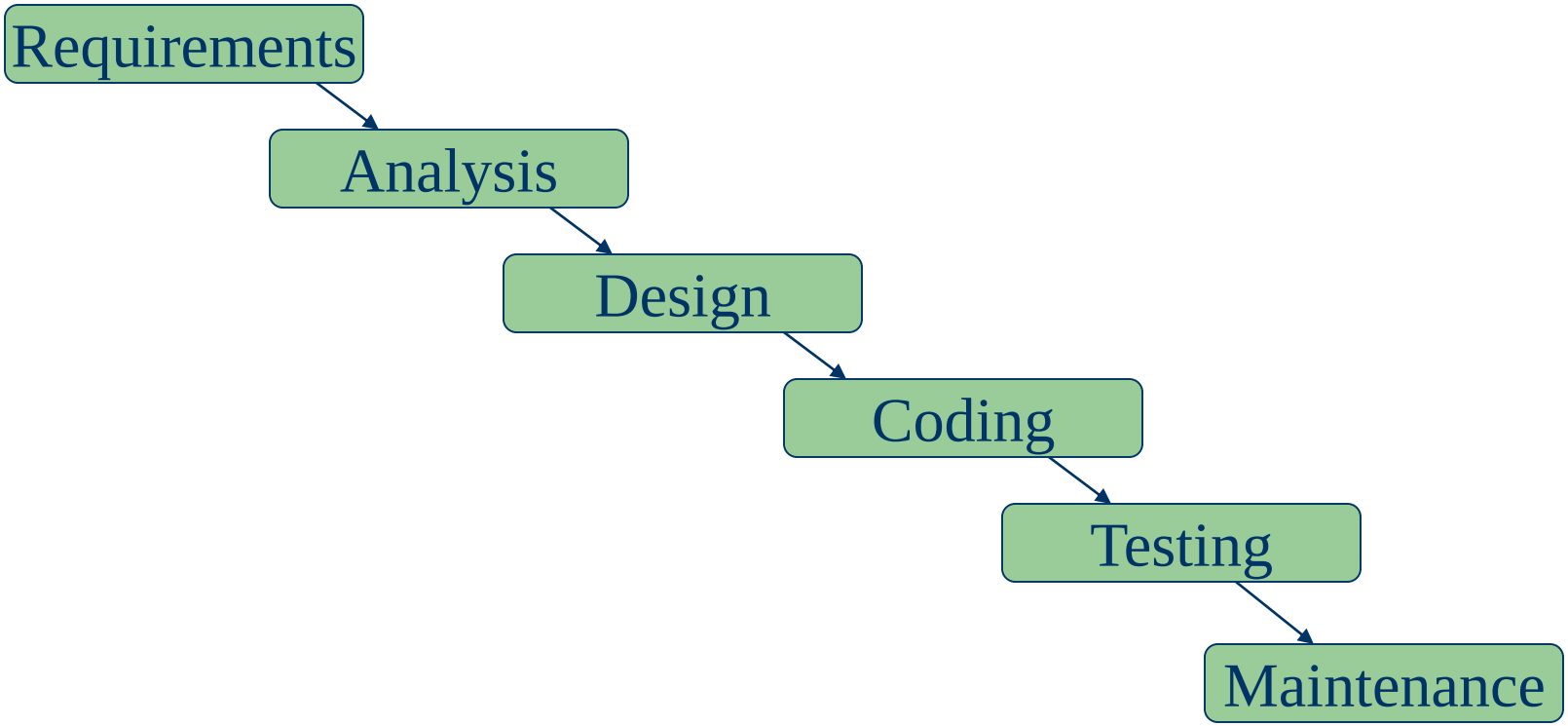
Code and fix – benefits

- Very cheap during the development stage (no administrative overhead)
- Quick results
- Good for small, simple projects (e.g. prototypes, short-lived demos)

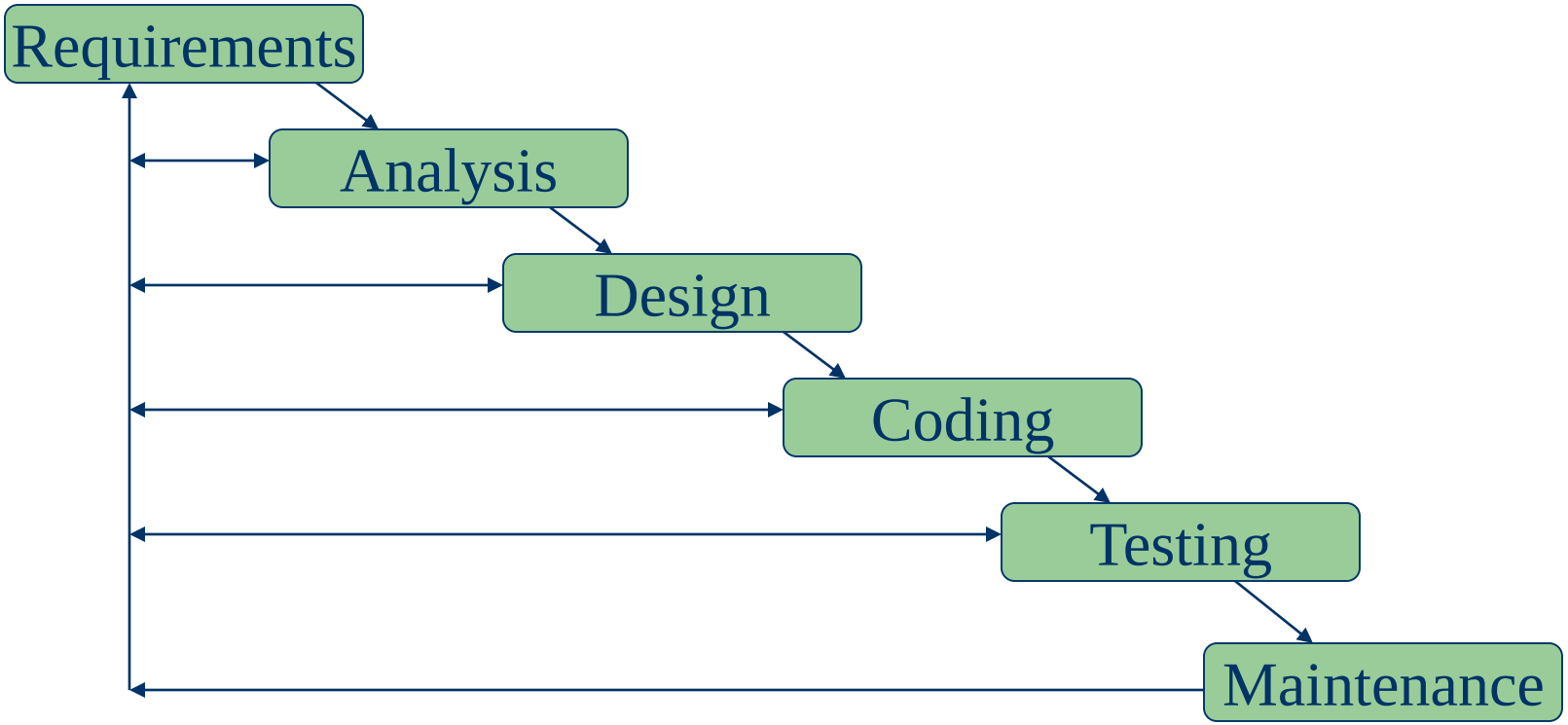
Code and fix – weaknesses

- Poor analysis (if any)
- Very expensive during the maintenance stage (lack of documentation, probably dirty code)
- Dangerous
  - No visibility/control
  - No resource planning

Waterfall Model



Waterfall Model – alternatives



Waterfall Model, cont.

- The basic principles of waterfall model:
  - linear, sequential, little splash back
  - each stage completes before the next starts
  - documentation and review at each phase transition
  - specifications serve as “contracts”

Waterfall Model – benefits

- Structured and disciplined
- Simple and easy to understand
- Progress of system development is measurable
- Enforces stability of requirements
- Early validation of intermediate products
- Low influenced by staff migration

Waterfall Model – weaknesses

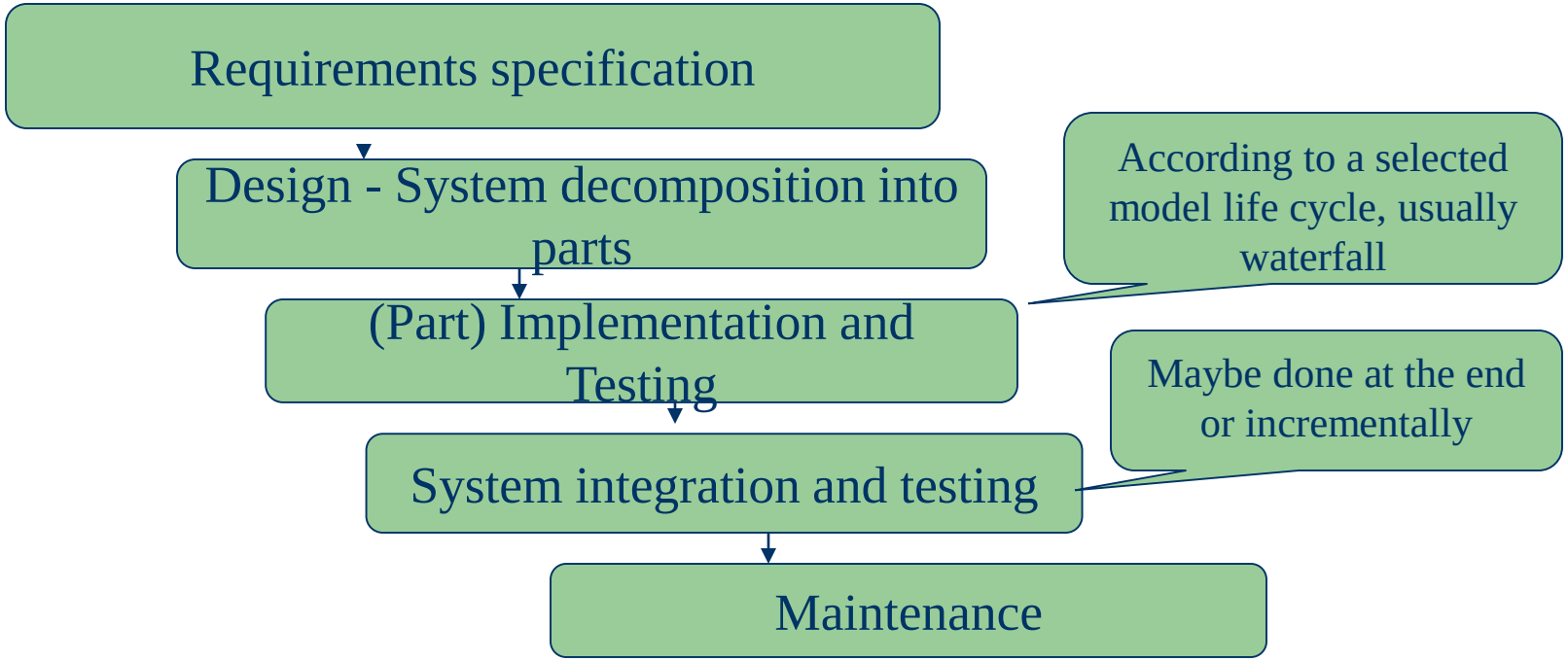
- All requirements must be known upfront
- Lack of flexibility (slow, and costly process)
- Product validation is delayed for a long time
- Increases the gap between users and developers

Waterfall Model – when to use

- Product with clear objectives and stable requirements
- Technology is understood and known
- Product needn’t be delivered quickly
- There is a demand for formal approvals of specifications at designated deadlines
- Team composition can be unstable and expected to fluctuate

Incremental Model

- Incremental development - staging and scheduling strategy in which the various parts of the system are developed at different times or rates, and integrated as they are completed.



Iterations vs. Increments

- Increments

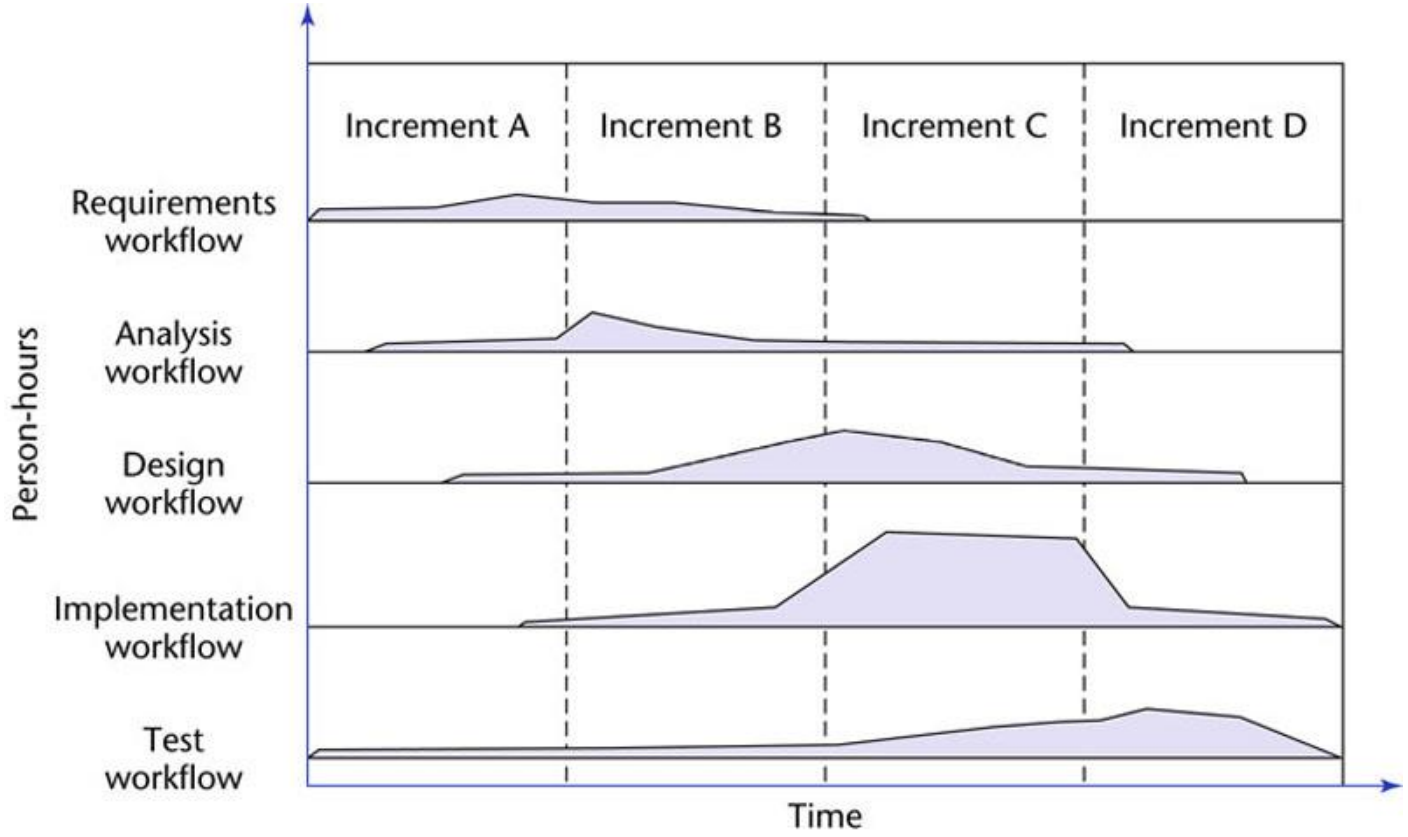
- Iterations

Incremental Model, cont.

- Approaches in which incremental model can be applied:
  - Classical: Requirements and Design first; Break down into parts (sub-systems) to be pursued in parallel (interfaces must be designed carefully)
  - Iterative & Incremental: Sequence of mini-waterfalls; each adds more functionality to the product or refines existing ones; requirement analysis and design stages are repeated within subsequent iterations

Incremental Model, cont.

Iterative & Incremental



Incremental Model – benefits

- Iterative & Incremental:
  - Can exploit knowledge from earlier increments
  - Mitigates integration risks early (through increments)
  - Easier for monitor
  - Reduced risk of changing requirements
- Both:
  - Customer gets important functionality early

Incremental Model – weaknesses

- Iterative & Incremental:
  - Mini-waterfalls do not encourage thinking ahead about the big picture
  - Difficult functionality may be deferred to be implement later
- Classic:
  - Definition of good interfaces between parts is needed
  - Not having all requirements upfront can reveal incompatibilities later

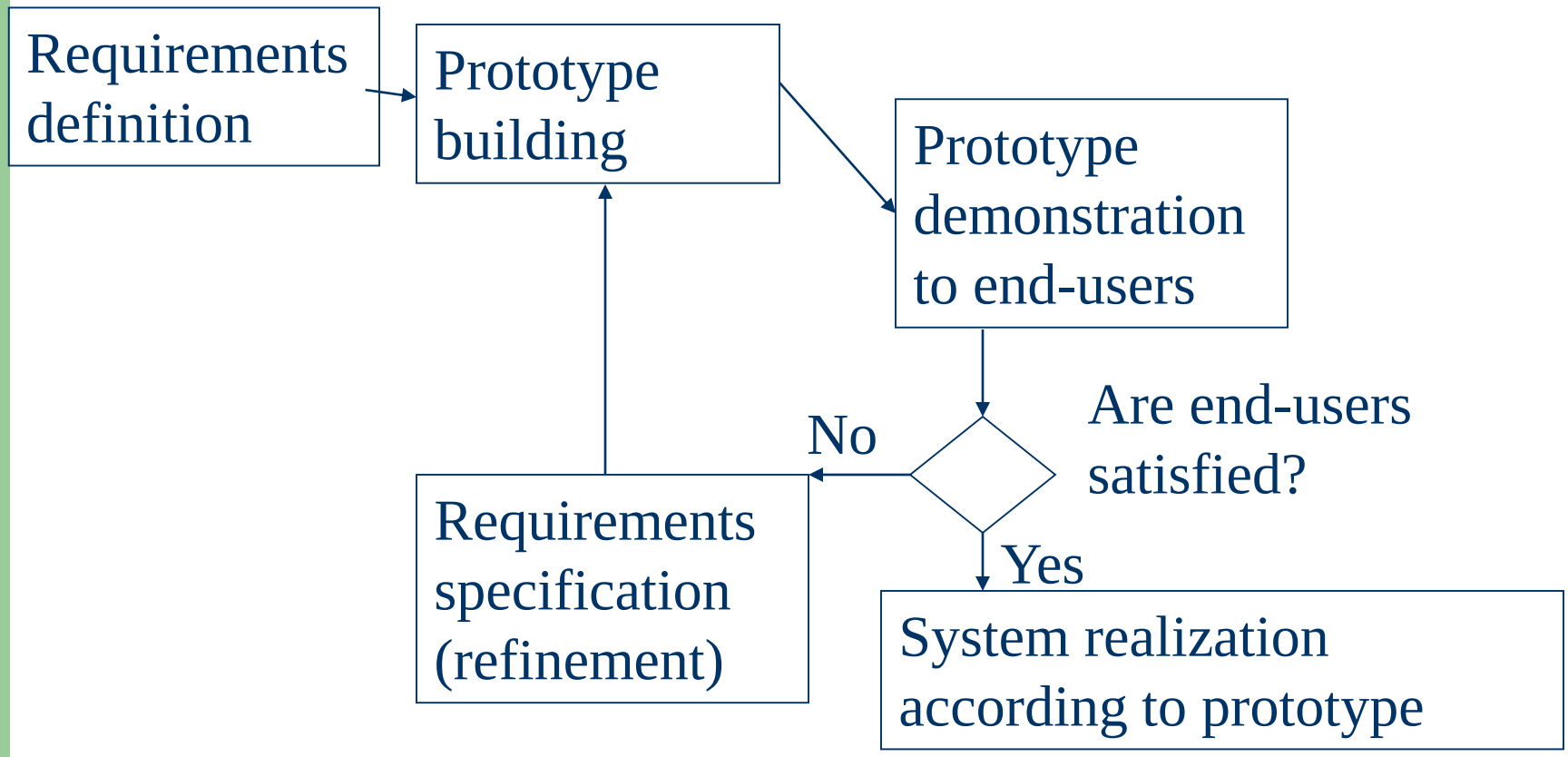
Incremental Model – when to use

- Iterative & Incremental:
  - Large project with not stable requirements
  - Projects implemented with a new technology, not experienced yet
  - Projects with long development schedules
- Classic:
  - Stable requirements known upfront
  - Rather known technologies, as increments are realized typically in waterfall

Prototyping Model

- A **prototype** – incomplete version of the software program being developed.
- A prototype focuses only on a few aspects of the eventual program, e.g. performance, user interaface, naviagation betweem screens, etc.

Prototyping Model, cont.



Prototyping Model, cont.

- Types of prototyping:
  - Throw-away (rapid) prototyping:
    - Paper prototyping
    - Running prototypes (prepared with special programs)
  - Evolutionary prototyping:
    - Running prototypes prepared in the target environment later refined and rebuilt

Prototyping Model – benefits

- Reduced time and cost by early determination of what the users really want even when they have problems in stating that
- Especially useful for resolving unclear objectives; developing and validating user requirements; investigating human computer interface
- Improved and increased user involvement

Prototyping Model – weaknesses

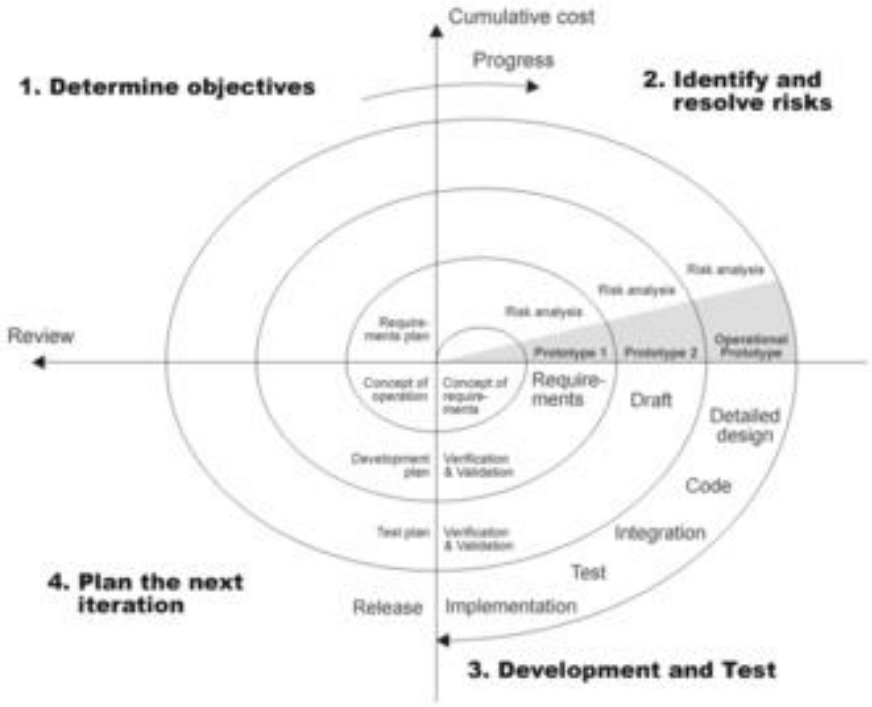
- Excessive development time of the prototype
- User confusion of prototype and finished system; users can demand new features after the prototype has been accepted
- Insufficient analysis, and a risk of bad design

Prototyping Model – when to use

- Rich computer-user interaction systems (prototypes are especially good for designing user interfaces)
- Large project with many users, and functions
- Project objectives are unclear. Requirements are unstable or have to be clarified
- Team composition is stable and team members are experienced

Spiral model

- Combination of iterative & incremental model with risk analysis and management
- Each iteration identifies and solve sub-problems with the highest risk; follows waterfall model



Spiral Model, cont.

- Four stages (repeated iteratively):
  - Determine objectives, alternatives, and constraints:
    - Objectives: functionality, performance, interfaces, etc.
    - Alternatives: build, reuse, buy, sub-contract, etc.
    - Constraints: cost, schedule, interface, etc.
  - Evaluate alternatives, identify and resolve risks:
    - Potential risks: lack of experience, new tech, tight schedule, etc.
    - Build prototype
  - Development and Test
    - Design, review it, develop code, inspect code, test product
  - Plan Next Iteration
    - Develop project plan
    - Develop an installation plan
    - Develop a test plan

Spiral Model – benefits

- Well suited for complex and large projects
- Strong support for risk analysis
- Critical high-risk functions are developed first
- Early and frequent feedback from users possible
- Emphasis on alternatives, supports the reuse of existing solutions

Spiral Model – weaknesses

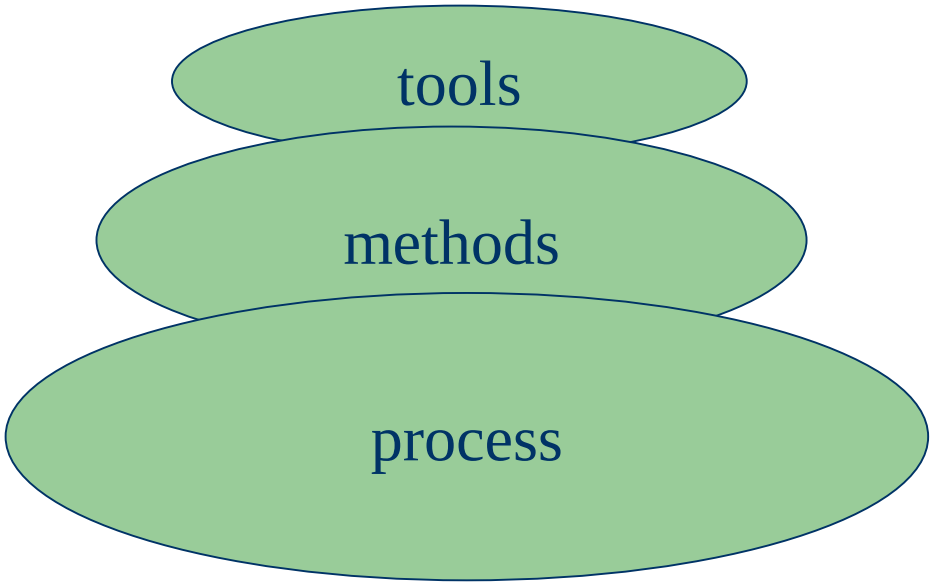
- Complex
- Requires qualified staff
- Time consuming risk analysis (maybe unnecessary for low-risk projects)

Spiral Model – when to use

- For medium to high-risk projects
- For unstable, and complex requirements
- Strong approval and documentation control is demanded

Summary

- SE provides the technical „how to” for building software. Describes processes (activities) of software development that include requirements specification, analysis, design, coding, testing, maintenance
- SE apart processes describe methods and tools that provide automated and semi-automated support for the process



Revision

- What is software engineering?
- Why software engineering is important?
- What is a software life cycle model?
- What software life cycle models were presented during the lecture?