Friedrich-Alexander-Universität
Technische Fakultät

FAU

Introduction to Software Engineering

Introduction

A. Maier, A. Sindel, S. Zeitler

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 18, 2024

Who are we? - Lab Members

Andreas
Maier

Aline
Sindel

Sally
Zeitler

1. Motivation
2. Organizational Matters
3. What is Software & Software Engineering
4. Software Processes
5. Software Development Life Cycle
6. Software Process Models

1. Motivation
2. Organizational Matters
3. What is Software & Software Engineering
4. Software Processes
5. Software Development Life Cycle
6. Software Process Models

Motivation

Why is it needed?

Software is needed in the modern world in almost all areas such as

- Infrastructure
- Industry
- Financial systems
- Security
- Entertainment

Sources:
Icons from Flaticon.com

Why is it needed?

Why is software development challenging?

Software has unique properties[1]

- Immaterial
- Largely not bound by physical laws
- Hard to measure
- Does not wear down, but still ages

As such, software can easily become too complex to understand and maintain

[1] Metzner: *Software Engineering - kompakt* (2020) [1] p. 7

Why is it needed?

Why is software development challenging?

Software development is also subject to technical advances which leads to[2]

- Increasing demands
- Diversity
- Faster delivery times
- Demand for security
- Subjection to user trust

[2] Sommerville: *Software Engineering* (2016) [2] p. 20

Why is it needed?

A little bit of history

In the early days of computing, massive improvements in computing power have overwhelmed the developers.[3] As a result, in the 1960s, the “Software Crisis" occurred with software that was[4]

- Inefficient
- Low quality
- Over budget
- Over time
- Not meeting the requirements
- Hardly maintainable
- Not delivered at all

The term “Software Engineering" was introduced in 1968 with the Nato conference on software engineering being held to discuss the software crisis.[5]

[3] Metzner: *Software Engineering - kompakt* (2020) [1] p. 10

[4] Nathan Bean, CIS 642 - Software Engineering Project I/CIS 643 - Software Engineering Project II, Kansas State University, Chapter 1: Link

[5] Sommerville: *Software Engineering* (2016) [2] p. 19

Why is it needed?

A little bit of history

Some examples of failed software projects:[6]

- 1963 - 1965 IBM multi-platform operating system:
  Product was delivered, but 1000 additional developers and one additional month were needed
- 1985 - 1987 Therac-25:
  Software issues as well as poor UI led to the admission of up to 100 times the intended radiation, which lead to the death of three patients
- 1995 Denver airport baggage system:
  Airport inoperable for 16 months, 560 million dollars over budget, requirements only partially fulfilled
- 1996 Ariane 5 launcher failure:
  The rocket exploded shortly after take-off due to a fault in the software in the inertial navigation system

[6] Nathan Bean, CIS 642 - Software Engineering Project I/CIS 643 - Software Engineering Project II, Kansas State University, Chapter 1: Link

Why is it needed?

Ariane 5 launcher failure

Click for video

- Software of inertial navigation system was reused from Ariane 4
- Incorrect signals (due to integer overflow) was sent to the engines
- Inertial navigation system shut down which led to explosion
- Software failure and critical systems validation failure[7]

Source: www.youtube.com/watch?v=gp_D8r-2hwk

[7] Ian Sommerville, Software Engineering, Supplementary: Link

Why is it needed?

A little bit of history

Even in the 21th century immense software failures occurred:[8]

- 2003 German Toll collection:
  Over 156 million euro revenue were lost due to uncollected tolls - the system was not delivered on time and the project was canceled in the end
- 2018 Boeing 737 Max:
  Due to faulty sensors and a non-redundant sensor system two deadly crashes occurred

[8] Nathan Bean, CIS 642 - Software Engineering Project I/CIS 643 - Software Engineering Project II, Kansas State University, Chapter 1: Link

Why is it needed?

Boeing 737 max crash[9]

Click for video

- Boeing 737 max with increased engine size on a 737 airframe to increase fuel efficiency
- Large engine was moved forward on the airframe, which changes the aerodynamic properties
- Software “Maneuvering Characteristics Augmentation System” (MCAS) to reduce the effect of the plane to pitch up (i.e. raise its nose)
- Software system only relied on a single angle-of-attack sensor
- Faulty sensor led to two deadly crashes

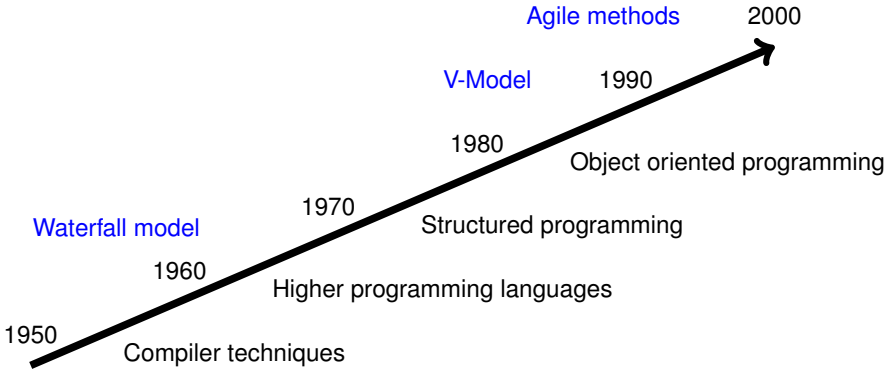
Source: https://www.youtube.com/watch?v=H2tuKiiznsY

[9] Nathan Bean, CIS 642 - Software Engineering Project I/CIS 643 - Software Engineering Project II, Kansas State University, Chapter 1: Link

Why is it needed?

A little bit of history

With advances in computing power, new methods of software development emerged:



Source: Metzner 2020 [1] p. 9 (adapted)

Why is it needed?

A little bit of history

CHAOS reports over the years

- 1994:[10]
  - 16.2 % on time and on budget
  - Unsuccessful projects cost on average 189% of original estimate
  - Unsuccessful projects required 222% of original target time

- 2011 - 2015:[11]
  - 44% of projects on budget
  - 40% of projects on time
  - 56% of projects on target

[10] CHAOS report 1994: Link

[11] CHAOS report 2015: Link

Why is it needed?

A little bit of history

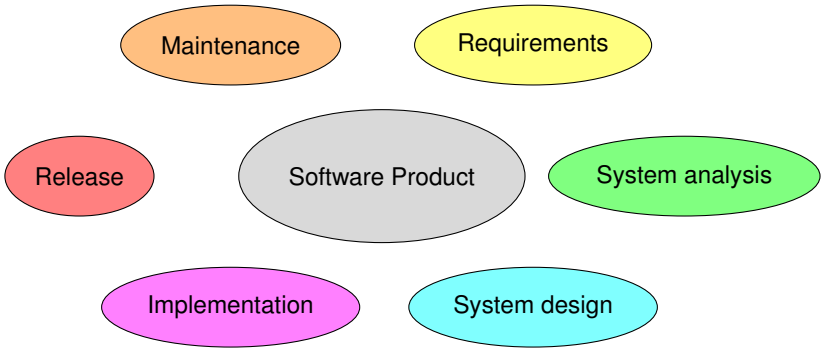
The use of software engineering methods directly influences the success rate of projects:
CHAOS report on different project methods 2011-2015:[12]

Project size	Method	Success	Challenged	Failed
Large	Agile	18%	59%	23%
Large	Waterfall	3%	55%	42%
Medium	Agile	27%	62%	11%
Medium	Waterfall	7%	68%	25%
Small	Agile	58%	38%	4%
Small	Waterfall	44%	45%	11%

[12] CHAOS report 2015: Link

What is Software Engineering?

Software Engineering focuses on practical aspects of developing and delivering software.



Lecture content

- Introduction & Software Processes
- Agile Software Development
- Requirements Engineering
- System Modeling
- Architectural Design
- Design Patterns
- Implementation
- Software Testing
- Software Evolution
- Software Project Management
- Software Engineering and Machine Learning

Organizational Matters

Dates

- Lecture: **Thursday, 12:15 - 13:45**, H4, Martensstr. 1
- Exercises:
  - **Tuesday, 12:15 - 13:45**, 00.152-113 Übungsraum, Martensstr. 3
  - **Thursday, 10:15 - 11:45**, 00.152-113 Übungsraum, Martensstr. 3
  - **Friday, 12:15 - 13:45**, 00.151-113 Übungsraum, Martensstr. 3
  - **Friday, 14:15 - 15:45**, 01.150-128 Seminarraum, Cauerstr. 11
- First exercise: Python recap
- Exercises start in the **second week**
- Please register for the StudOn course until **Friday, April 19, 2024 (23:59)** to choose your exercise group

Grading

- Module consists of the lecture **and** exercises (together 5 ECTS)
- Written exam (90 min) in the semester break, determines grade
- Exercises are **highly recommended**; there will be questions regarding the theoretical and practical exercises in the exam

Exercise Content

Theoretical and practical exercises – bring your laptop to the practical sessions

- Requirements engineering
- Software processes
- Software modeling (e.g. UML, communication diagram)
- Software architectures
- Design patterns
- Software testing
- Software reuse
- Project management
- Software engineering and machine learning

Exercises

Pattern Recognition Lab

FAU

Requirements are:

- Basic knowledge of Python and Numpy
- Basic knowledge of object oriented programming

How it works:

- 12 exercises throughout the semester
- First exercise: Python recap
- Last exercise: Mock exam
- Assistance during exercise sessions

Please set up your IDE and Python installation **before** the first exercise!
Test your setup e.g. by running a script to print "Hello World".

What is Software & Software Engineering

Starting with the basics

Some key questions to get started:

- What is Software?
- What is **good** Software?
- What is a Software system?
- What is Software Engineering?

Starting with the basics

What is Software?

According to the IEEE Standard Glossary of Software-Engineering [3] a **software product** consists of:

- Computer programs
- Procedures
- Rules
- Associated documentation and data

Starting with the basics

What is Software?

Good Software is[13]

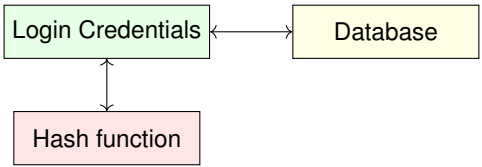
- Maintainable
- Dependable
- Efficient
- Acceptable

[13] Sommerville: *Software Engineering* (2016) [2] p. 22

Starting with the basics

Software systems

A **software system** is a system whose subsystems and components are also software.[14]



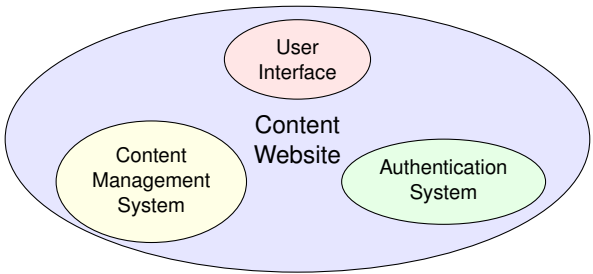
Exemplary authentication system

[14] Metzner: *Software Engineering - kompakt* (2020) [1] p. 3

Starting with the basics

System of systems[15]

A **system of systems** combines multiple heterogeneous systems to form a more complex system.
Examples for this can be found in healthcare, transportation, and military services.
In this course we will focus on software and software systems.



Exemplary system of systems for a website

[15] Sommerville: *Software Engineering* (2016) [2] p. 581-583

Starting with the basics

Software products[16]

There are different kinds of software products:

- **Generic**:
  Stand-alone systems, sold on the open market → company controls the specification
- **Custom**:
  Comissioned by a customer → customer controls the specification
- **Inbetween**:
  Generic base with adaption to customer needs

Usually the costs of software are divided into 60% development and 40% testing costs.
For custom systems however, the system evolution costs can exceed development costs.

[16] Sommerville: *Software Engineering* (2016) [2] p. 20-21

Starting with the basics

Software applications

Additionally, there are different types of applications such as[17]

- Stand-alone
- Interactive transaction-based
- Embedded
- Batch processing
- Entertainment
- Modeling and simulation
- Data collection
- System of systems
- Web-applications

Depending on the software product, different domains can be interwoven.

[17] Sommerville: *Software Engineering* (2016) [2] p. 25

Starting with the basics

Challenges

Within all types of software systems, the key challenges remain the same:[18]

- Heterogeneity (system of many platforms)
  - Websites on different browsers
  - Applications on different operating systems
- Business / social change
  - Restructuring of the application
  - Customizable systems
- Security & Trust
  - New security mechanisms such as hash codes
  - Trust of the user is required

[18] Sommerville: *Software Engineering* (2016) [2] p. 24

Starting with the basics

Common problems during software development

Additionally, there are common problems during software development[19]

- Communication between involved groups
- Acceptance and Integration
- Configuration and version management
- Portability
- Requirement specification

[19] Metzner: *Software Engineering - kompakt* (2020) [1] p. 8

Starting with the basics

What is software engineering?

These problems can be minimized with the use of[20]

- Standards
- Methods
- Tools

> “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." (IEEE Standard Glossary of Software-Engineering [3])

[20] Metzner: *Software Engineering - kompakt* (2020) [1] p. 8

Starting with the basics

What is software engineering?

Software engineering is related to both computer science and systems engineering:[21]

Computer science is concerned with theories and fundamental methods of Software.
Software engineering can make use of these theories, but focuses on the practical realization.

System engineering covers all aspects of producing complex systems, including hard- and software.
Software engineering focuses specifically on software and is included on system engineering.

[21] Sommerville: *Software Engineering* (2016) [2] p. 20

Starting with the basics

Fundamentals of software engineering

Areas of responsibility and knowledge of a software engineer:[22]

Software product
- Application area
- Usability and ergonomics
- Architecture, models and plans

Resources
- Team management
- Software, hardware and platforms
- Tools

Software process
- Software process methods
- Software development
- Reuse of software

Project
- Project management
- Project models
- Estimation methods

[22] Metzner: *Software Engineering - kompakt* (2020) [1] p. 14

Starting with the basics

Fundamentals of software engineering

Areas of responsibility and knowledge of a software engineer:[23]

Software product
- Application area
- Usability and ergonomics
- Architecture, models and plans

Resources
- Team management
- Software, hardware and platforms
- Tools

Software process
- **Software process methods**
- Software development
- Reuse of software

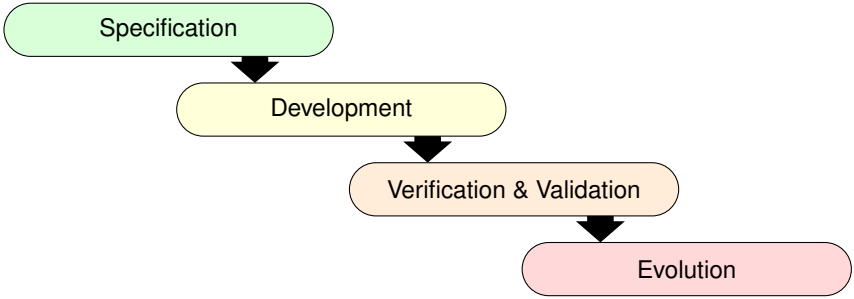
Project
- Project management
- Project models
- Estimation methods

[23] Metzner: *Software Engineering - kompakt* (2020) [1] p. 14

Software Processes

Software Processes

The sequence of activities that lead to the development of a software product are called **software process**.[24]
All software processes includes the following activities:



Source: Sommerville 2016 [2] p. 44 (adapted)

[24] Sommerville: *Software Engineering* (2016) [2] p. 44

Software Processes[25]

Specification
→ Main functionalities and constraints are defined

Development
→ The software is devised and realized

Verification & Validation
→ Compliance with specification and customer needs is checked

Evolution
→ Prevent aging of software by updating to new requirements and needs

[25] Sommerville: *Software Engineering* (2016) [2] p. 44

Software Processes

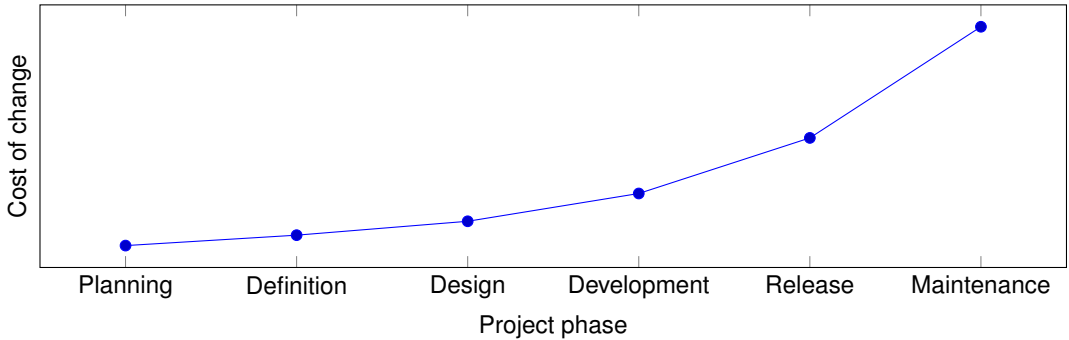
Terms that are often used:[26]

- Products
  Outcomes of a process activity
- Roles
  Reflect responsibilities, e.g.: project manager, configuration manager, developer, etc.
- Pre- & post-conditions
  Statements that need to be fulfilled before or after a process activity

[26] Sommerville: *Software Engineering* (2016) [2] p. 44

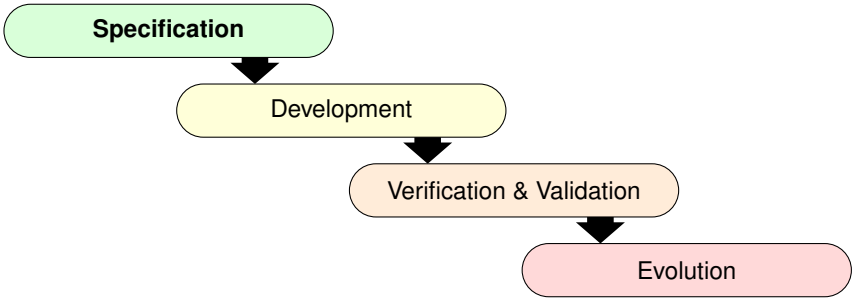
Software Processes

The later the change, the more it generally costs.
Due to this, the use of software engineering methods is usually cost efficient.



Example how costs of changes typically increase within the software development life cycle (exemplary numbers)

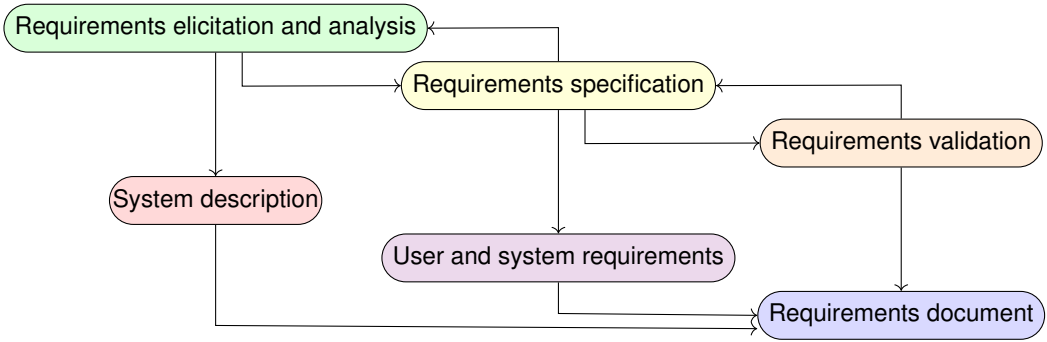
Software processes



Source: Sommerville 2016 [2] p. 44 (adapted)

Software processes

Software specification



Source: Sommerville 2016 [2] p. 55 (adapted)

Software processes

Software specification[27]

Requirements elicitation and analysis

- Observation of existing systems
- Discussion with users and producers
- Task analysis
- Development of models and prototypes

Requirements specification

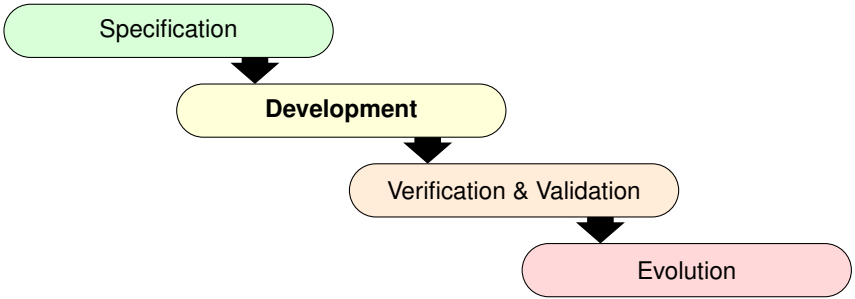
- Formulating and documenting requirements
- User requirements (abstract)
- System requirements (detailed)

Requirements validation

- Realism
- Consistency
- Completeness
- Correction of errors

[27] Sommerville: *Software Engineering* (2016) [2] p. 55

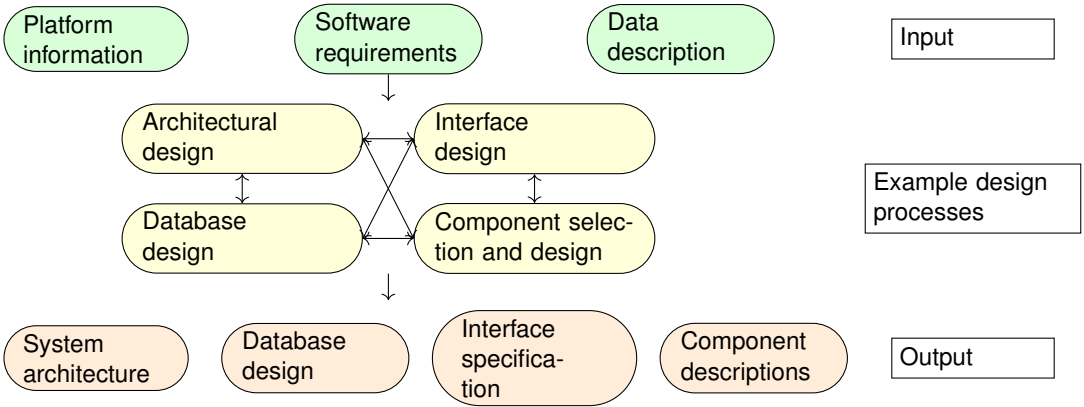
Software processes



Source: Sommerville 2016 [2] p. 44 (adapted)

Software processes

Software design and implementation



Source: Sommerville 2016 [2] p. 56 (adapted)

Software processes

Software design and implementation[28]

Architectural design

- System structure
- Principal components and relationships
- Distributions

Database design

- Data structures
- Representation in the database

Interface design

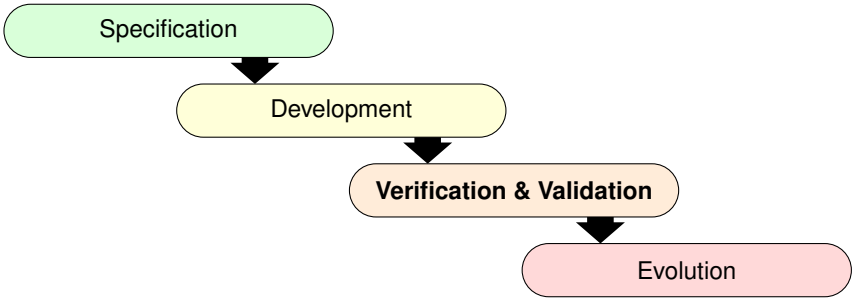
- Unambiguous interface specification
- Communication between components without implementation knowledge

Component selection and design

- Search for reusable components
- Define changes to reused components
- Design new components

[28] Sommerville: *Software Engineering* (2016) [2] p. 57

Software processes



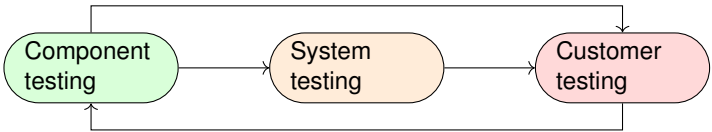
Source: Sommerville 2016 [2] p. 44 (adapted)

Software processes

Software verification and validation[29]

Software verification and validation (V & V) to show that:

- **Verification**: The system conforms to its specification
- **Validation**: The requirements specification meets the expectations of the system customer



Source: Sommerville 2016 [2] p. 58 (adapted)

[29] Sommerville: *Software Engineering* (2016) [2] p. 58

Software processes

Software verification and validation[30]

Component testing

- Components tested by developers
- Individual tests without other components

System testing

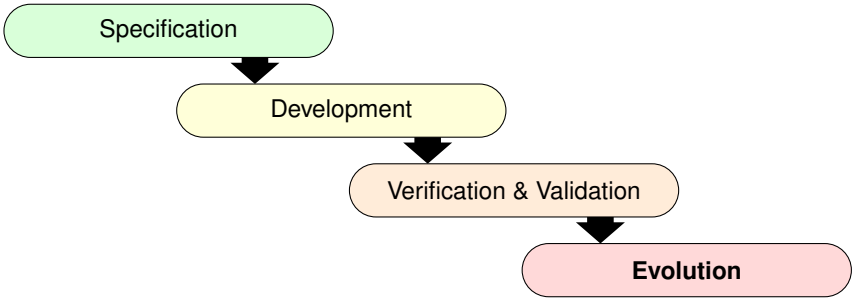
- Complete system is tested
- Errors from unforeseen interactions and interfaces are solved
- Prove the system meets requirements

Customer testing

- Final stage before acceptance for operational use
- System is tested by customer with real data
- Requirements problems need to be solved

[30] Sommerville: *Software Engineering* (2016) [2] p. 59

Software processes



Source: Sommerville 2016 [2] p. 44 (adapted)

Software processes

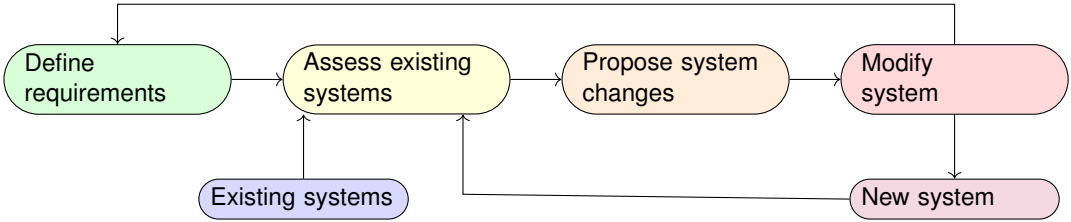
Software evolution[31]

- Software is highly flexible
- Changes are usually cheaper than hardware changes
- Software development and evolution are a continuum
- Software is continually changed throughout its lifetime

[31] Sommerville: *Software Engineering* (2016) [2] p. 60-61

Software processes

Software evolution



Source: Sommerville 2016 [2] p. 61 (adapted)

Software processes

Software evolution[32]

There are two major ways of coping with change:

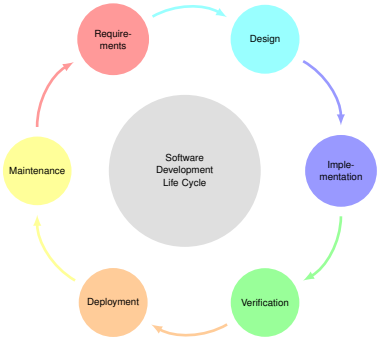
- **Change anticipation**
  - Predict possible changes
  - Minimize rework
  - E.g. through prototyping to refine requirements before committing to the full product
- **Change tolerance**
  - Design accommodates changes to the system
  - Usually through incremental development
  - A single increment is enough to incorporate change

[32] Sommerville: *Software Engineering* (2016) [2] p. 61

Software Development Life Cycle

Software development life cycle

The software development life cycle (SDLC) is an infinite loop covering all software processes



Source: Stephens 2015 [4] p. 277 (adapted)

Software development life cycle

Life cycle phases[33]

The software development life cycle (SDLC) is an infinite loop covering all software processes

1. Initiation, Concept development, Preliminary planning, Requirements analysis ($\rightarrow$ Specification)
2. Design: High-level design & Low-Level Design
3. Implementation
4. Validation & Verification
5. Deployment: Roll out of application
6. Maintenance ($\rightarrow$ Evolution)
7. Iterate from beginning (refine requirements etc.) ...
8. Until Disposal: Planning of software removal (clean up & archiving)

Source: Stephens 2015 [4] p. 277 (adapted)

[33] Stephens: *Beginning Software Engineering* (2015) [4] p. 277-279

Software development life cycle

Software process models

Software process models (or SDLC models)[34,35,36]:

- Are high-level, abstract descriptions of software processes
- Conceptually describe the steps for the software development life cycle
- Have specifically predefined steps which are followed during software development

There are various different process models, such as Waterfall model, V-model, agile models etc.

[34] Sommerville: *Software Engineering* (2016) [2] p. 45

[35] Amazon AWS, What Is SDLC (Software Development Lifecycle)?: Link

[36] SD Solutions LLC, What is the Software Development Life Cycle?: Link

1. Motivation
2. Organizational Matters
3. What is Software & Software Engineering
4. Software Processes
5. Software Development Life Cycle
6. **Software Process Models**

Software Process Models

Software Process Models

And why we use them

A **process model** is an abstract representation of the activities during the software engineering process to

- define activities
- specify the order of activities
- determine phases with: activities, goals, roles, methods

The use of process models leads to

- a guideline for system development
- a common view for logical and temporal planning
- better planability
- independence from single persons
- possible certifications (e.g. TÜV)
- earlier identification of errors through testing

Source: Metzner 2020 [1] p. 18-19 (adapted)

Software Process Models[37,38]

Process models can be divided into agile and plan-driven models, although hybrid models exist as well.

For **plan-driven processes** all activities are planned in advance, progress is measured against the plan.

Agile processes are planned incrementally, which leads to easier changes in the process.

Plan-driven models:

- Waterfall
- Incremental
- Spiral model
- V-model

Agile models:

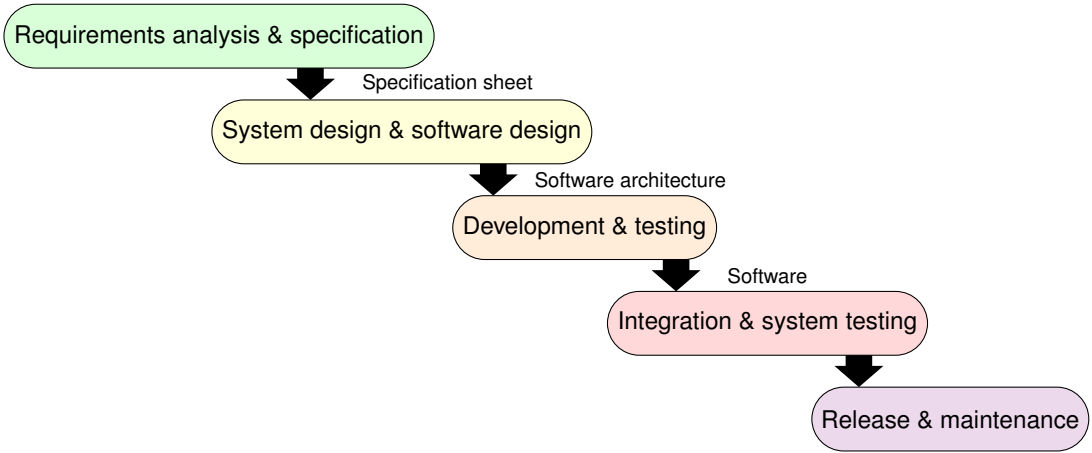
- Scrum
- Kanban
- Extreme programming
- Dynamic Systems Development Method
- Adaptive Case Management

[37] Sommerville: *Software Engineering* (2016) [2] p. 73-75

[38] Cha, Taylor, and Kang: *Handbook of Software Engineering* (2019) [5] p. 30

Process models

Waterfall model



Source: Sommerville 2016 [2] p. 47 (adapted)

Process models

Waterfall model[39]

The waterfall model is historically the oldest process model. The phases are clear cut, deliver predefined documents and need to be finished before starting the next phase. It is a completely plan-driven approach.

Requirements analysis and definition:	Project management starts, problems and specifications are compiled, requirements are defined and documented
System and software design:	Drafts, models and software architecture are created
Development and testing:	Programming of the software, verification with unit tests
Integration and system tests:	Software components are combined, the complete system is tested
Release and maintenance:	Installing the system, correcting errors, prevent software aging, incorporate new requirements

[39] Sommerville: *Software Engineering* (2016) [2] p. 47-49

Process models

Waterfall model[40]

Pros

- Linear process
- Intuitive
- Easily understood
- Top-down
- Plannable
- Non-interruptible

Cons

- Fixed phases
- Early commitment
- No reiteration
- No integration of new requirements
- Often not practical

[40] Metzner: *Software Engineering - kompakt* (2020) [1] p. 20

Process models

Waterfall model[41]

Usually used in small projects where ...

- requirements are easily defined and not expected to change
- project, budget and process are predictable
- a strict process is needed

[41] Metzner: *Software Engineering - kompakt* (2020) [1] p. 20

Process models

Improved Waterfall model[42]

In many cases with the traditional Waterfall model, customers approached the software manufacturers afterwards with request for:

- new requirements or
- process changes

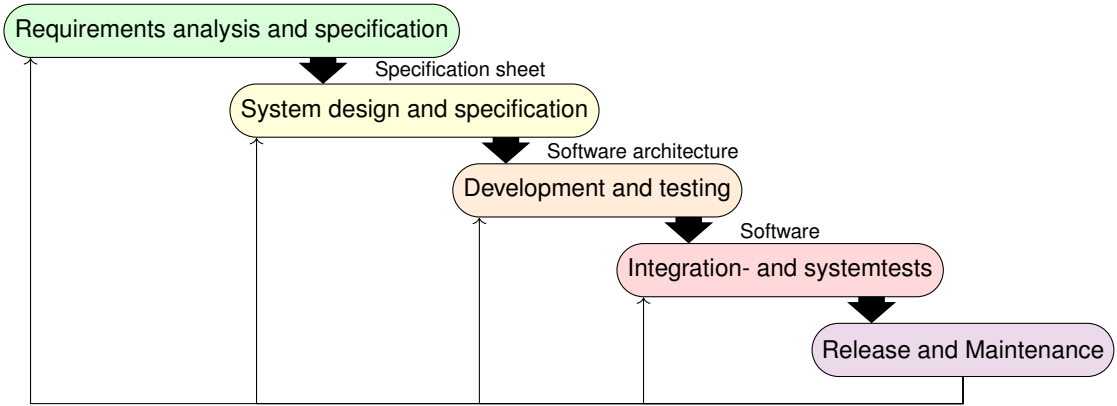
Consequently, the improved Waterfall models came up:

- **Iterative**: Run all processes of the life cycle then optionally jump back to an arbitrary phase
- **Incremental**: After running a single phase optionally jump back to an arbitrary preceding phase

[42] Metzner: *Software Engineering - kompakt* (2020) [1] p. 20-22

Process models

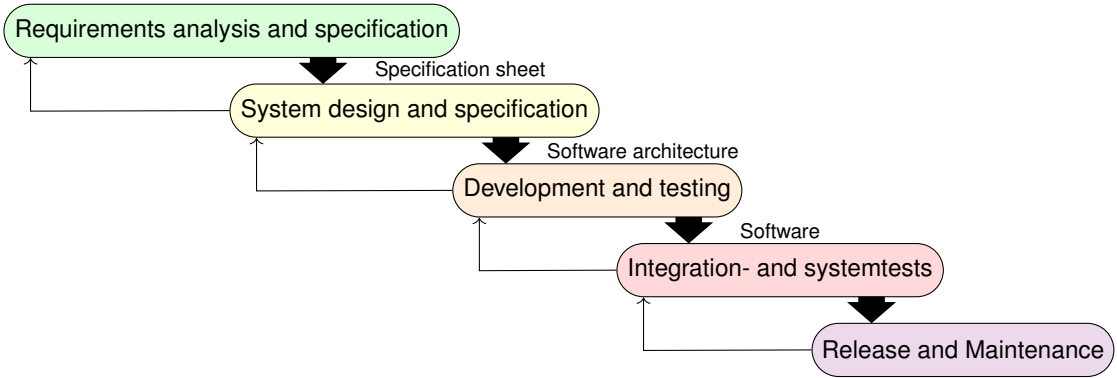
Iterative Waterfall model



Source: Metzner 2020 [1] p. 21 (adapted)

Process models

Incremental Waterfall model



Source: Metzner 2020 [1] p. 21 (adapted)

Process models

Improved Waterfall model[43]

Pros

- Linear process
- Intuitive
- Easily understood
- Top-down
- Plannable
- Non-interruptible
- Repeatable

Cons

- Fixed phases
- Early commitment, but new requirements can be integrated
- Changed requirements may lead to high costs
- Structure tends to degrade

[43] Metzner: *Software Engineering - kompakt* (2020) [1] p. 22

Process models

Improved Waterfall model[44]

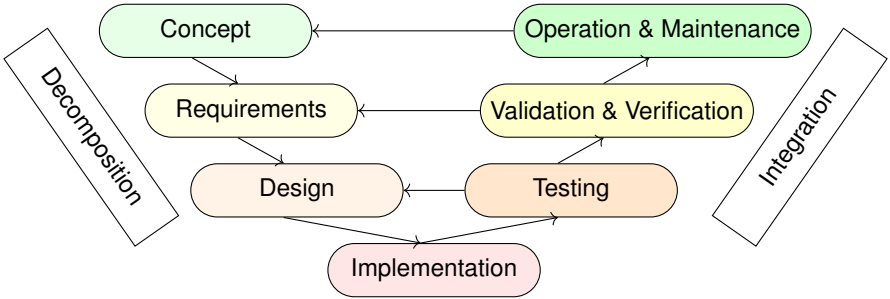
Usually used when ...

- requirements are easily defined
- only small changes may occur and are planned for in the budget
- project, budget and process are predictable
- a strict, but slightly flexible process is needed

[44] Metzner: *Software Engineering - kompakt* (2020) [1] p. 22

Process models

V-model



Source: Stephens 2015 [4] p. 276 (adapted)

Process models

V-model[45,46]

- V-model is an improved Waterfall model
- Each phase is basically repeatable
- For each phase an associated test phase has to be defined with a completion deadline ($\rightarrow$ V-shape)
- **Decomposition**: Tasks on the left side are broken down from an high conceptual level into detailed tasks (top-down)
- **Integration**: Tasks on the right side check the finished application at different levels of abstraction (bottom-up)

[45] Metzner: *Software Engineering - kompakt* (2020) [1] p. 22

[46] Stephens: *Beginning Software Engineering* (2015) [4] p. 275-276

Process models

V-model[47]

Pros

- Applicable to large and complex systems
- Detailed specifications, role definitions, result patterns
- Quality oriented

Cons

- Large overhead for small projects
- Testing starts relatively late
- Strict phases
- Participants need to be trained to follow the model

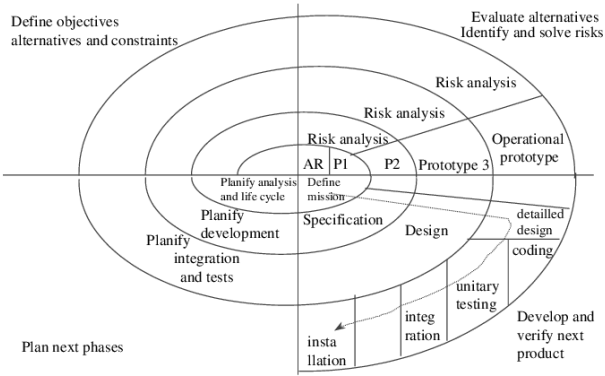
Usually used in ...

- Software development for government agencies
- Security relevant projects

[47] Metzner: *Software Engineering - kompakt* (2020) [1] p. 23-24

Process models

Spiral model



Source: Boehm 1988 [6] p. 64

Process models

Spiral model[48]

Pros

- Incremental
- Flexible
- Usable product after short time
- Every cycle leads to improvements

Cons

- Reiteration of phases
- Prototypes may require completely new development
- Huge management effort
- The more spirals, the more the project costs

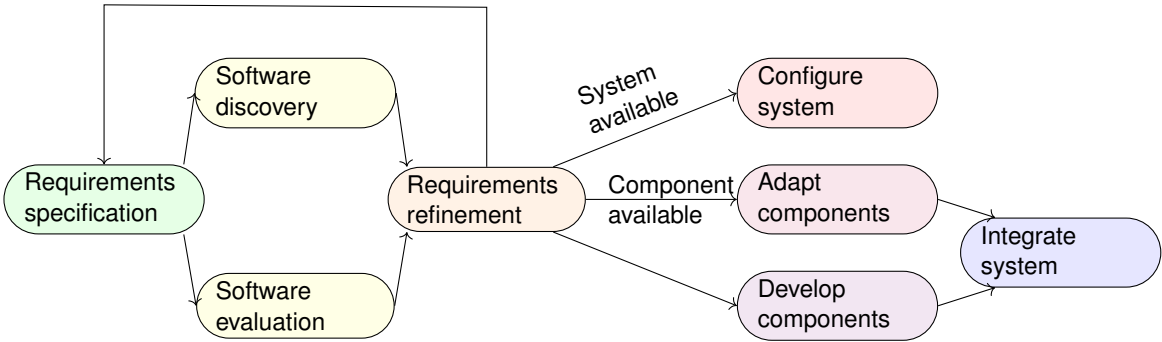
Usually used for ...

- projects with prototyping
- large projects

[48] Metzner: *Software Engineering - kompakt* (2020) [1] p. 25

Process models

Reuse-oriented approach



Source: Sommerville 2016 [2] p. 52 (adapted)

Process models

Reuse-oriented approach[49]

Pros

- Reduced cost and risk
- Faster delivery

Commonly used for ...

- Web Services
- Collections of objects or packages
- Configurable stand-alone software systems

Cons

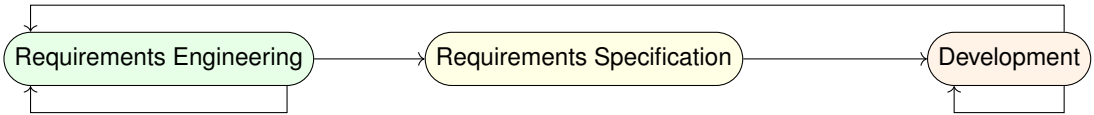
- Compromises in requirements
- System may not meet real user needs
- No or limited control over system evolution

[49] Sommerville: *Software Engineering* (2016) [2] p. 52-54

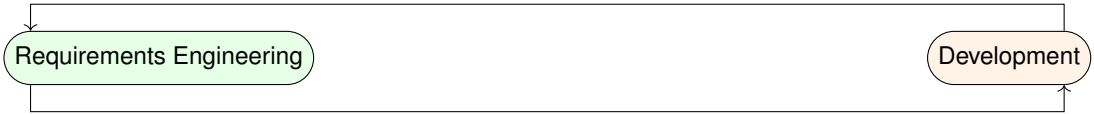
Process models

Agile models

General difference between **plan-driven**



and **agile** approach



Agile models are discussed next week!

Source: Sommerville 2016 [2] p. 74 (adapted)

Questions & References

Comprehensive questions

- What is Software Engineering?
- Which are the fundamentals of Software Engineering?
- Name four attributes of good software
- What are common problems during the software development?
- Name and briefly explain the different software processes
- What are software process models?
- Explain the difference between incremental and iterative Waterfall model
- Draw the V-model and briefly explain its keypoints

Course books & further reading

Course books:

- Ian Sommerville, Software Engineering, 10th edition, 2016, Ebook
- Anja Metzner, Software Engineering - kompakt, 2020, in German, Ebook
- Sungdeok Cha et al., Handbook of Software Engineering, 2019, Ebook

Further reading:

- Rod Stephens, Beginning Software Engineering, 2015, Ebook
- Volker Gruhn and Rüdiger Striemer, The Essence of Software Engineering, 2018, Ebook
- Software crisis – Sommerville 2016 Supplementary: Link
- Examples of failed software projects: Link
- Software development life cycle: Amazon AWS, SD Solutions LLC

References I

[1] Anja Metzner. *Software Engineering - kompakt*. Carl Hanser Verlag GmbH & Company KG, 2020.

[2] Ian Sommerville. *Software Engineering*. Always learning. Pearson, 2016.

[3] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (1990), pp. 1–84.

[4] Rod Stephens. *Beginning Software Engineering*. Wrox beginning guides. Wiley, 2015.

[5] Sungdeok Cha, Richard N. Taylor, and Kyochul Kang. *Handbook of Software Engineering*. Springer International Publishing, 2019.

[6] B. W. Boehm. "A spiral model of software development and enhancement". In: *Computer* 21.5 (1988), pp. 61–72.

[7] Volker Gruhn and Rüdiger Striemer. *The Essence of Software Engineering*. Springer International Publishing, 2018.