Welcome to EIST

Einführung in die Softwaretechnik (EIST)
Introduction to Software Engineering

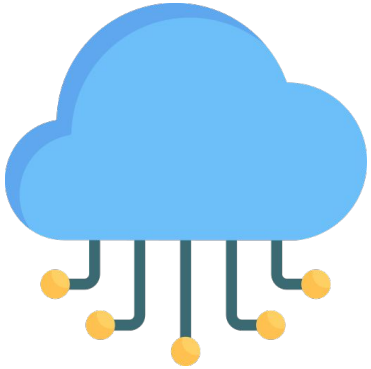
EIST is the largest course @ TUM!

"~2,200 students"

A major update – We've moved to the cloud!

The new curriculum focuses on
Software Engineering for "the Cloud, "
the default platform for designing and deploying
almost all modern software systems

Focus: Software engineering for the cloud!



Traditional software systems

Relied on in-house dedicated infrastructure for building and deploying software systems

Cloud software systems

Modern software systems are designed and deployed in the cloud for scalable and cost-effective infrastructure

Cloud computing

- ***Cloud computing*** *is the delivery of* ***computing services*** *over the Internet!*

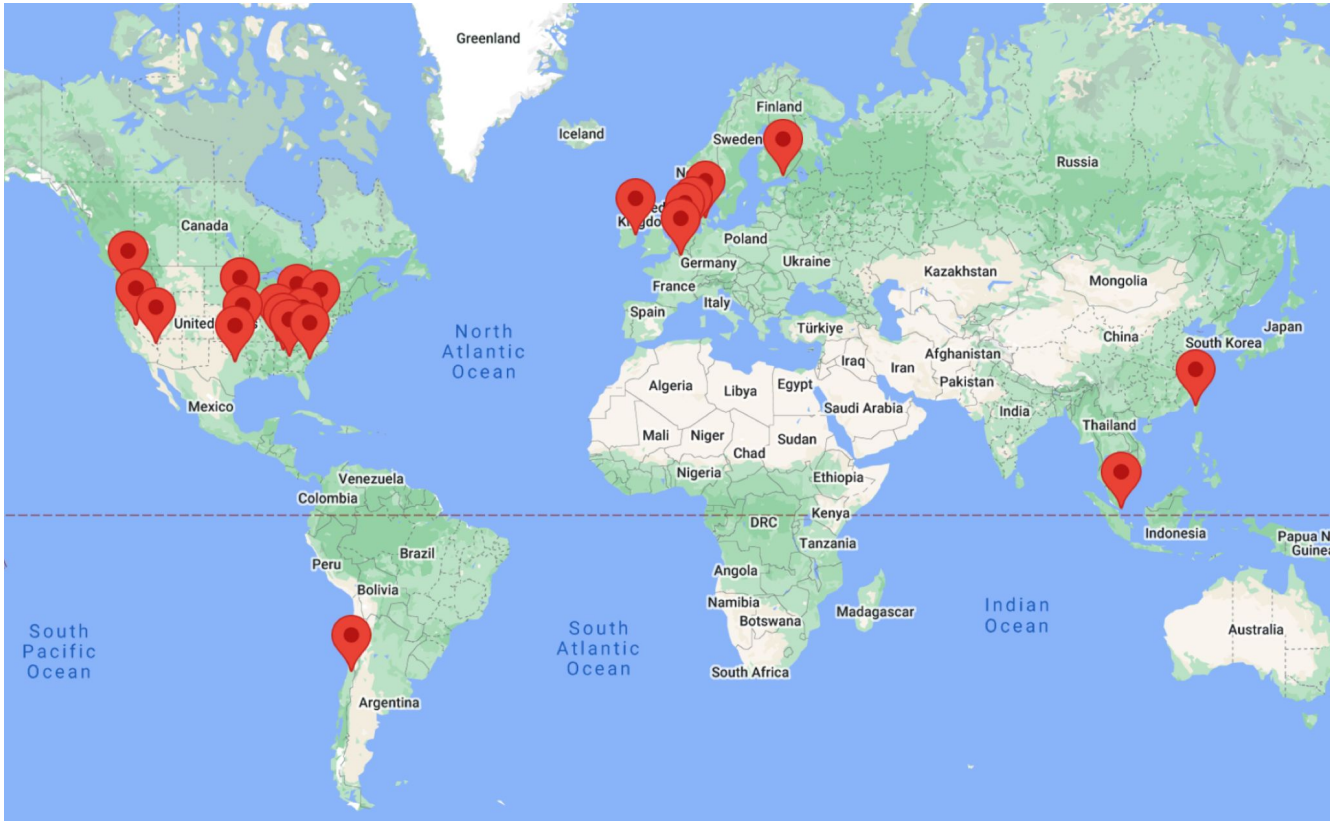
  — Microsoft Azure
  https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing

Cloud infrastructure

Large data centers hosting 100s of 1000s of machines

https://www.popsci.com/technology/article/2012-10/12-beautiful-photos-googles-problematic-data-centers/

(Google's) Geo-distributed data centers



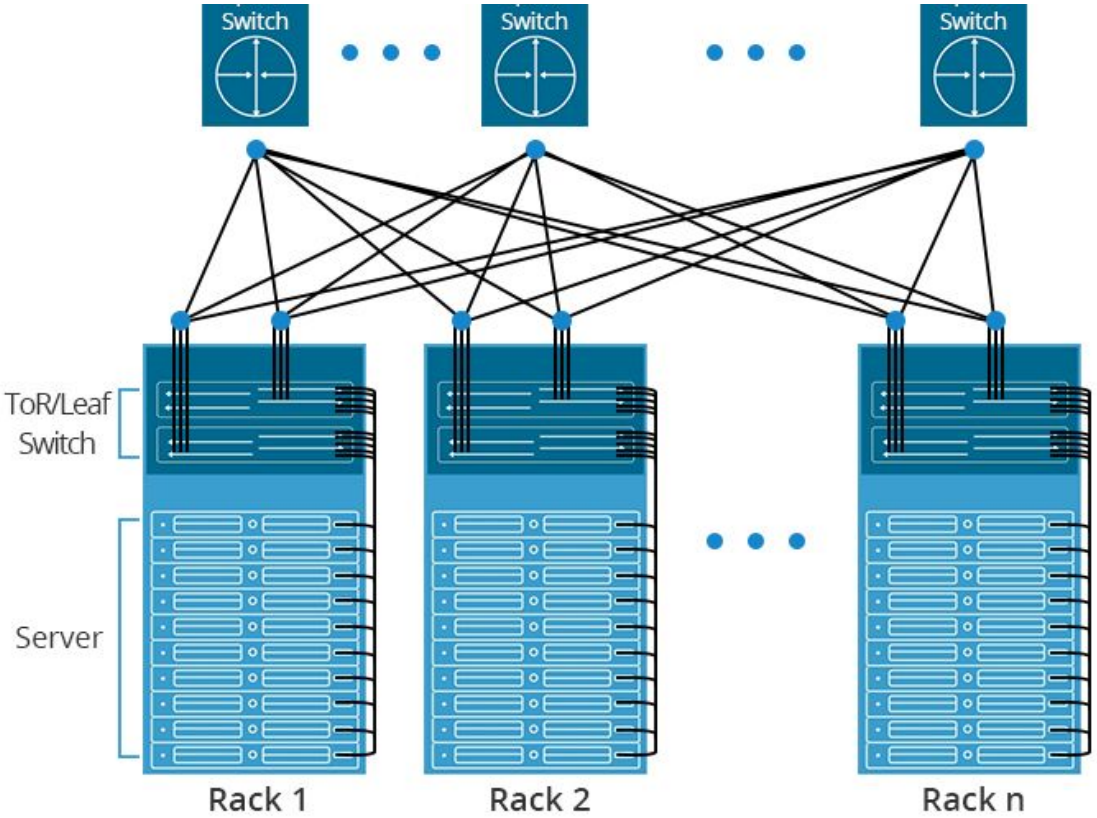
29
locations worldwide

11
countries

Google's datacenters locations: https://www.google.com/about/datacenters/locations/

Data centers

A server rack



Types of cloud computing

- Public cloud
  - Owned and operated **by a third-party cloud service providers**
  - E.g., Microsoft Azure, Amazon AWS, Google Cloud

- Private cloud
  - Cloud computing resources used exclusively **by a single business or organization**
  - E.g., LRZ @ TUM

- Hybrid cloud
  - Hybrid clouds **combine public and private clouds**
  - Allows data and applications to move between private and public clouds
  - Offers greater flexibility, more deployment options, and **security and compliance**

A traditional computing infrastructure

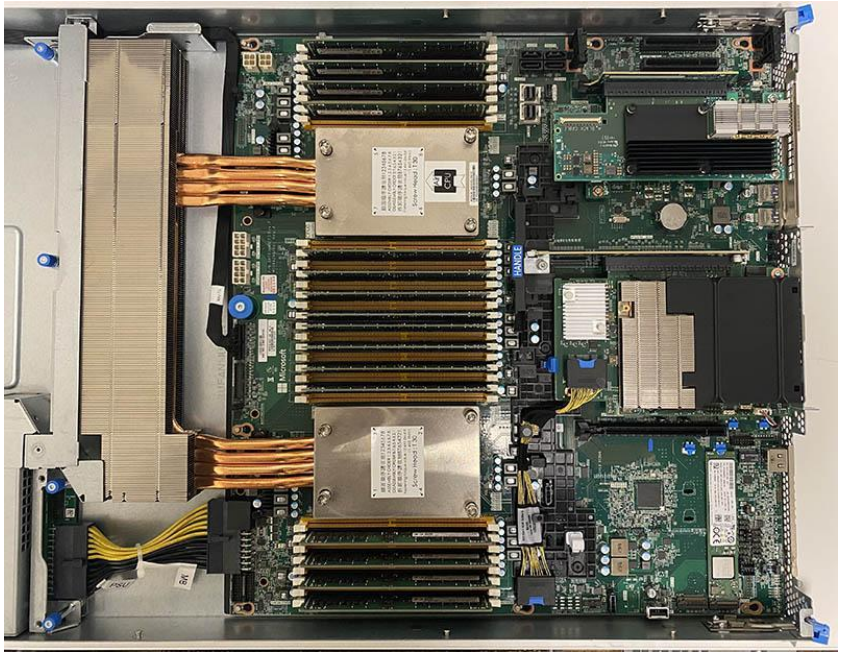
- A **typical server** in good old days
  - Compute (CPUs)
  - Storage (disk)
  - Network

- **Exclusive access** to the server
  - No multi-tenancy of resources

An example server

- **Server used by Microsoft Azure**
  - 64 cores / 128 threads (AMD Epyc, 3rd Gen)
  - Up to 1TB RAM
  - Typically deployed around 16 SSDs
  - Would have been the Top500 super computer in the 2000s and be in this list until 2007

- **Important:** Not all applications need this much compute power <-> Smaller servers might still take the same space / maintenance cost

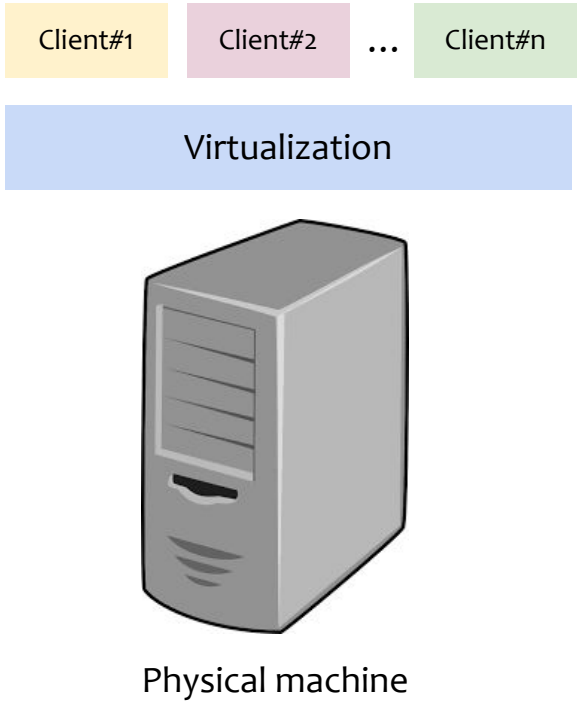


Source: https://specbranch.com/posts/one-big-server/

How can we share hardware between multiple tenants efficiently?

Virtualization

- **Virtualization** is technology that you can use to create virtual representations of servers, storage, networks, and other physical machines
  - Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine
- **Advantages:** Efficient resources of computing resources in the cloud across multiple tenants/clients



Physical machine

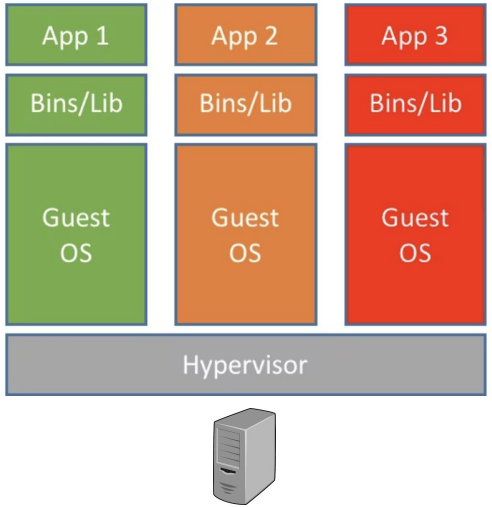
https://aws.amazon.com/what-is/virtualization/

Types of virtualization

- **Virtualization of computing resources** can be
  a. Compute
  b. Network
  c. Storage

a. Compute virtualization

- **Compute virtualization** is a process that partitions a physical server into multiple virtual servers
- Different level of abstractions:
  - Virtual machines (VMs)
  - Containers (covered in L09)

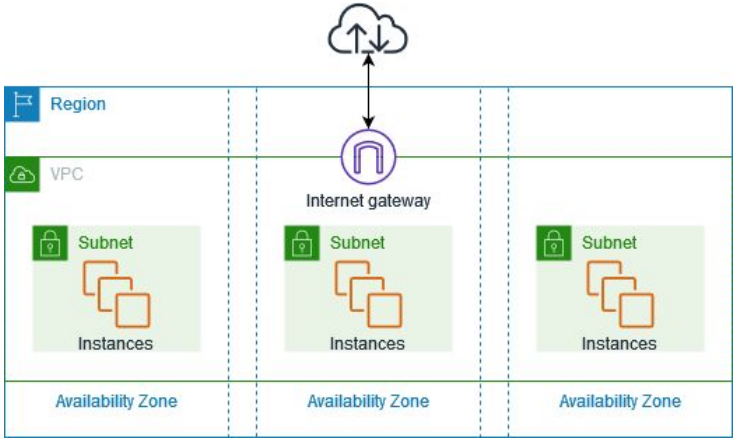


VM-based virtualization of a physical machine

https://www.ibm.com/cloud/blog/containers-vs-vms

b. Network virtualization

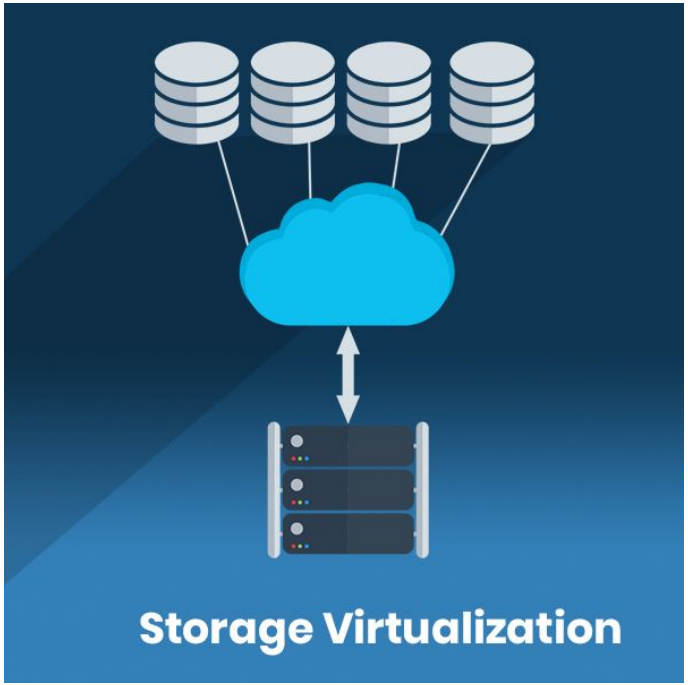
- Any computer network has hardware elements such as switches, routers, & firewalls
- **Network virtualization** is a process that combines all of these network resources to create Virtual Private Cloud (VPC)
  - Administrators can adjust and control these elements virtually without touching the physical components, which greatly simplifies network management



https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

c. Storage virtualization

- **Storage virtualization** combines the functions of physical storage devices to pool the storage hardware in data center
  - A large unit of virtual storage that can assign be assigned to multiple tenants
  - The cloud providers streamline storage activities, such as archiving, backup, and recovery



Data lakes:
https://cloud.google.com/learn/what-is-a-data-lake

Case study: Google Cloud

- Infrastructure
- Data and analytics
- Management tools
- Security and identity

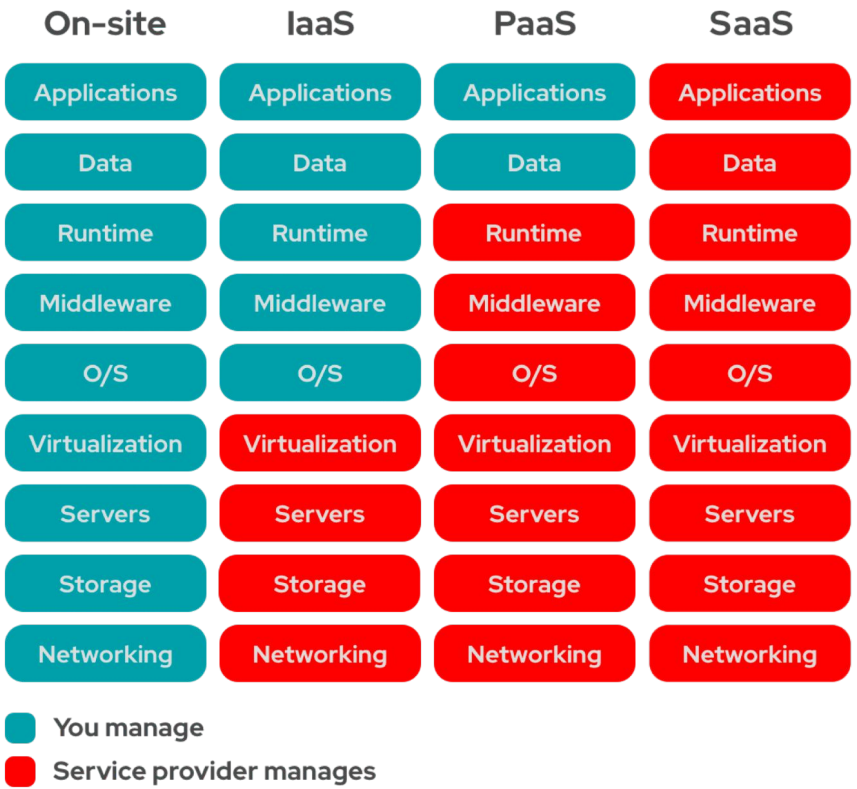
https://cloud.google.com/

Types of cloud services

- Cloud services: Each type of service provides a different level of abstraction and functionality to users, depending on their needs and requirements

- **Infrastructure as a Service (IaaS)**
  - IaaS provides virtualized computing resources
- **Platform as a Service (PaaS)**
  - PaaS provides a platform for application development and deployment
- **Software as a Service (SaaS)**
  - SaaS provides software applications over the internet

Types of cloud services



https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas

Infrastructure as a Service (IaaS)

- **IaaS** exposes virtualized computing resources, such as servers, storage, and networking

- **Ideal** when complete control over their computing environment is required

- **Examples of IaaS providers**
  - Amazon Web Services (AWS)
  - Microsoft Azure
  - Google Cloud Platform

Platform as a Service (PaaS)

- **PaaS** provides users with a platform for developing, deploying, and managing applications without having to manage the underlying infrastructure

- **Ideal** for organizations that want to focus on application development and deployment without having to manage the underlying infrastructure

- **Examples of PaaS providers**
  - Heroku
  - Google Appengine
  - Fly.io

Software as a Service (SaaS)

- **SaaS** provides users with access to software applications over the internet, without having to install or manage the software on their own devices or servers
  - Users typically pay for the service on a subscription basis

- **Ideal** for organizations that want to avoid the upfront costs and ongoing maintenance associated with traditional software applications

- **Examples of SaaS providers**
  - Salesforce, Microsoft Office 365, Google Workspace
  - **Typical applications:** Mail/Office tools, Customer relationship management (CRM), Enterprise resource planning (ERP)

A case study: A new AI-based search engine!

Build a software startup!

Let's first discuss the advantages of cloud computing

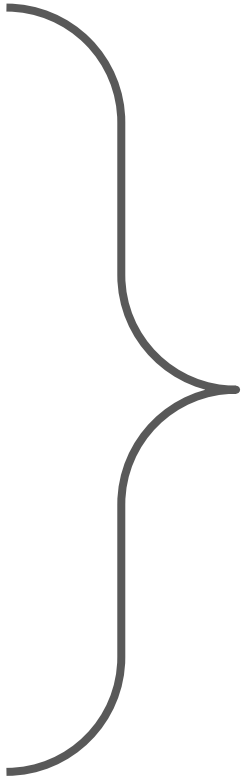
And next, the software engineering challenges of building a "successful" cloud-based software company

Benefits of cloud computing

- Scalability:
  - Easily scale your applications up or down based on changing needs
- Cost-effective:
  - Can be more cost-effective than maintaining in-house infrastructure
- Flexibility:
  - Offers software engineers flexibility in terms of choosing the services they need
- Reliability:
  - Higher levels of reliability and uptime than an in-house IT team can provide
- Collaboration:
  - Cloud computing facilitates collaboration and teamwork among software engineers

Software engineering challenges of cloud computing

- Software architectures
  - Monolithic
  - Microservices
- System challenges
  - Modularity, data management
  - Performance, scalability, concurrency
  - Security, reliability, availability
- Dependability
  - Testing, program analysis
- Software management
  - Source code management, build, DevOps
- System deployment and monitoring
- Software quality and project management



Focus of our
EIST curriculum

L01 Introduction

Prof. Pramod Bhatotia
Systems Research Group
https://dse.in.tum.de/

Welcome to EIST

- **Teaching concept:**
  - **Synchronous:** Lectures and tutorials
  - **Asynchronous:** Lecture recordings, slides, readings, and homework exercises
  - **Dedicated team:** Lecturers, Teaching Assistants (TAs), and Tutors for **~2,100 students**
  - **State-of-the-art curriculum: Cloud software systems**
- **Expectation:**
  - Be open to new technical concepts, and learning methods
  - Be active and curious, and learn to organize yourself
  - **Be kind and patient:** bear with us if something does not work out as expected
- **Main goals:**
  - You learn how to analyse, design, implement, test and deploy **cloud software systems**
  - You can achieve **the best possible grade** in the course

Today's learning goals

- **Part 0:** (*a quick!*) Introduction to cloud computing
- **Part I:** Administrative
  - Introduction to the course team
  - Understand the course organization
- **Part II:** Introduction to software engineering
  - Terminology and software engineering activities
  - Abstraction and system modeling with UML
- **Part III:** Course overview
  - **Course focus:** Cloud software engineering
  - An overview of the software engineering activities in the cloud
- **Part IV:** Software engineering models
  - Defined (e.g., Waterfall model) and empirical methods (e.g., Scrum model)
  - Agile methods in software engineering projects (Scrum)
  - A note on tactical vs strategic programming

Outline

- ~~**Part 0:** (*a quick!*) Introduction to cloud computing~~

- **Part I:** Administrative
  - **Introduction to the course team**
  - Understand the course organization

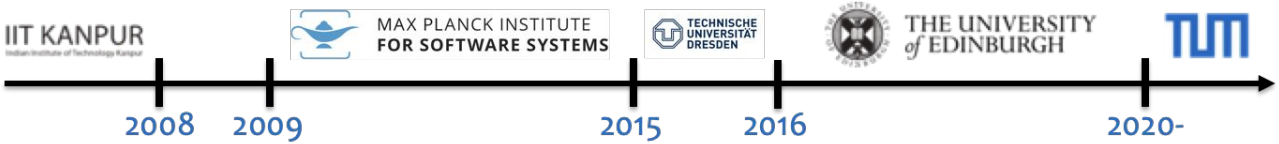
- **Part II:** Introduction to software engineering

- **Part III:** Course overview

- **Part IV:** Software engineering models

Prof. Pramod Bhatotia

- Full Professor and Chair at TUM
  - Systems of Research Group
  - Homepage: https://dse.in.tum.de/bhatotia/
- Research:
  - https://dse.in.tum.de/



Lecturers

Prof. Pramod Bhatotia

Course Lead

Dr. Marco Elver

Guest Lecturer (L07 and L08)

Dr. Jörg Thalheim

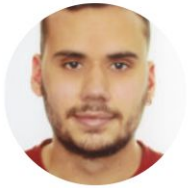
Guest Lecturer (L09)

Guest Lecturers

Teaching Assistants (TAs)

Dimitrios Stavrakakis 🇬🇷

Head TA



Manos Giortamis 🇬🇷

Head Exercise Instructor

Peter Okelmann 🇩🇪

Thore Sommer 🇩🇪

Harshavardhan Unnibhavi 🇮🇳

Exams Instructors

Martin Fink

Head Exam Instructor

Ilya Meignan--Masson

Teofil Bodea

Nicolò Carpentieri

Julian Pritzi

IT support

Ruth Demmel

Andreas Jung

and the IT Support (RBG) team

Tutors

Andrii Bochkovskyi

Amar Ribic

Ivan Lubyako

Linus Bohle

Lana Atiya

Johannes Klein

Moritz Wendel

Mykhailo Kazymyr

Özgür Deniz Türker

Abdelrahman Diab

Berat Bozkurt

Thees Haddinga

Tutors

Michael Berezovský

Alper Peker

Mürüvvet Bozkurt

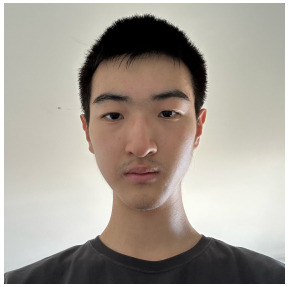
Lie Leon Alexius

Naz Filiz

Moritz Linsmaier

Alexander Reyers

Daniel Kunz



Tingsheng Chen

Maximilian Palmetshofer

Ivan Logvynenko

Gregory Wiskemann

Tutors

Aniruddh Zaveri

Alp Titiz

Serhad Çalışkan

Braun Ekaterina

Aymen Faouel

Tuan Khang Nguyen

Erik Hado

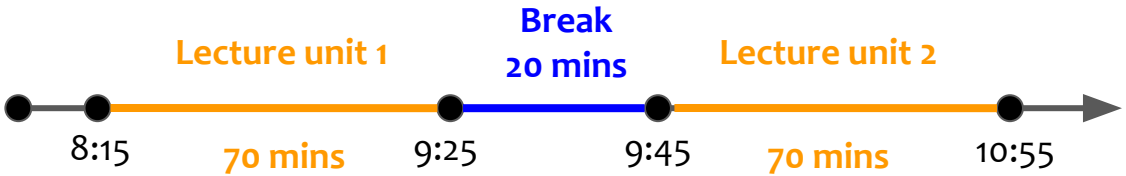
Outline

- Part I: Administrative
  - Introduction to the course team
  - **Understand the course organization**
- Part II: Introduction to software engineering
- Part III: Course overview
- Part IV: Software engineering models

Course organization

A. (10x) Lectures
B. (8x) Tutorials
C. (8x) Exercises
   - **+ (2x)** homework for practice after the first and last lectures – **NOT Graded**
D. (2x) Exams
   - Main exam (GOE)
   - Repeat exam (GORE)
   - **+ (1x)** Practice exam
E. Tools

A: Lecture structure



- **Venue:** HS 1 (FMI Building)
- **Livestream:**
  - https://live.rbg.tum.de/
- **Video recordings**
  - https://live.rbg.tum.de/

There could be technical issues with livestream and video recording!

Lectures

Course Focus:

Cloud Software Engineering

Clients

Cloud Software

L03 + L04 + L05
System Design and Implementation

L06 + L07
Software Testing and Program Analysis

L08
Software Management, Build Systems, & DevOps

L09
Software Deployment and Monitoring

L10 Software Quality and Project management

L02 System Design Requirements and Software Architectures

L01 Introduction

Lecture schedule

#	Date	Topic
1	24.04	Introduction **(NO tutorials)**
	01.05	**Holiday (no lecture, no tutorials)**
2	08.05	System Design Requirements and Software Architectures
3	15.05	System Design I: Modularity and Data Management
4	22.05	System Design II: Performance, Concurrency, and Scalability
	29.05	**Holiday (no lecture, no tutorials)**
5	05.06	System Design III: Security, Reliability, and Availability
6	12.06	Software Testing
	19.06	**Holiday (no lecture, no tutorials)**
7	26.06	Program Analysis: Static Analysis and Dynamic Analysis
8	03.07	Software Management, Build Systems, and DevOps
9	10.07	Software Deployment and Monitoring
10	17.07	Software Quality and Project Management + **Exam prep.** **(NO tutorials)**
	24.07	Practice Exam

B: Tutorials

- **(26x)** tutors groups on **Monday-Friday, except Thursday**
  - **(12x)** In-person tutorials
    - **Room number:** 01.07.014, Boltzmannstr. 3, 85748 Garching
  - **(14x)** Online tutorials via Zoom
    - **Zoom link:**
      https://tum-conf.zoom-x.de/j/62966214303?pwd=j4VQOToMxdJaRuATxqzuqKSBtit1dj.1

- Important info for tutorials:
  - **No obligation** for students to attend, but **highly recommended**
  - In case the **tutor group meeting was cancelled** (SVV, FVV, holiday), **please re-allocate** yourself to another tutor group

Tutorial slots and info

	Monday	Tuesday	Wednesday	Thursday	Friday
Session #1 (10:00 - 11:30h)	(a) In-person	(a) In-person	(a) In-person	**No tutorials (Lecture slot)**	(a) In-person
	(b) Online	(b) Online	(b) Online		(b) Online
Session #2 (13:00 - 14:30h)	(a) In-person	(a) In-person	(a) In-person	**No in-person**	(a) In-person
	(b) Online	(b) Online	(b) Online	(b) Online	(b) Online
Session #3 (15:00 - 16:30h)	(a) In-person	(a) In-person	(a) In-person	**No in-person**	(a) In-person
	(b) Online	(b) Online	(b) Online	(b) Online	(b) Online

Tutorial information:

https://collab.dvb.bayern/display/TUMeist/Tutorial+Information

Structure of a tutorial (90 minutes)

- **Quick summary** of the lecture
  - Q&A related to the lecture
- Explain the **programming basics (PB) exercises**
  - Prepare you for the homework programming exercises
- Introduce the **homework exercises** in Artemis
- **Q&A for completed homework exercises, if any!**
  - If you have questions about previous homework after the due date, ask the tutors!

Tutorial schedule

#	Week starting	Based on the lecture content of:
1	**No tutorials**	Introduction
2	12.05	System Design Requirements and Software Architectures
3	19.05	System Design I: Modularity and Data Management
4	26.05	System Design II: Performance, Concurrency, and Scalability
5	09.06	System Design III: Security, Reliability, and Availability
6	16.06	Software Testing
7	30.06	Program Analysis: Static Analysis and Dynamic Analysis
8	07.07	Software Management, Build Systems, and DevOps
9	14.07	Software Deployment and Monitoring
10	**No tutorials**	Software Quality and Project Management **+ Exam prep.**

Tutorial registration: TUMonline

- Enter your preferences on TUMonline between **Thursday (April 24)** and **Thursday (May 1)**
- Select groups on TUMonline registration procedure based on their
  - **Mode (IN-PERSON / VIRTUAL)**
  - **Time slots**
- **Please select as many as slots possible!**
  - **Otherwise, you might not get matched!**
- Receive your assignment on **Friday (May 2)** on TUMonline
- The first tutor group meetings take place on **Monday (May 12)**

C: Exercises

Programming exercises

Programming Basics (PB)

Covered in the tutorials

NOT graded

Programming (P)

Homework (individually)

Graded

Programming Extra (PE)

Homework

NOT graded

Exercises

- Released on Artemis after the lecture (Thu)
- Due **Fri (5pm)** in two weeks
  - More than two weeks to work on the exercises
- **(8x)** Exercises
  - Programming Basic (PB) exercises — **NOT graded**
    - Discussed during the tutorial
  - Programming exercises (P) — **Graded**
    - Homework exercises done individually
  - Programming Extra exercises (PE) — **NOT graded**
    - Extra exercises for practicing

- **Note: (2x) Practice exercises after the first and last lectures are also not graded**

Homework exercises timeline

Programming Basics (PB)
covered in the tutorials

Publication of exercises on Artemis
Thursday

1st week

Understand the exercises
Friday

2nd week

Submission due on Artemis
Friday, 5pm

More than two weeks to work on the homework exercises!

Submission schedule for homework exercises

#	Submission deadline (Friday, 5pm)	Based on the lecture content of:
1	**NOT Graded**	Introduction
2	23.05	System Design Requirements and Software Architectures
3	30.05	System Design I: Modularity and Data Management
4	06.06	System Design II: Performance, Concurrency, and Scalability
5	20.06	System Design III: Security, Reliability, and Availability
6	27.06	Software Testing
7	11.07	Program Analysis: Static Analysis and Dynamic Analysis
8	18.07	Software Management, Build Systems, and DevOps
9	25.07	Software Deployment and Monitoring
10	**NOT Graded**	Software Quality and Project Management **+ Exam prep.**

Bonus for homework exercises

- **(8x)** Homework exercises
- **In total, homework exercises will account 10 bonus points on top of the exam points for the overall grade!**
- **IMPORTANT:**
  - You must first **PASS the main exam (at least 4.0)** to get the bonus points
  - The bonus is **only applicable** for the **main exam (GOE)**
  - Bonus **does NOT apply** for the **repeat exam (GORE)**

D: Exams

- First Exam (GOE)
  - Tentative: July / August
- Repeat Exam (GORE)
  - Tentative: Sept / October

Note: The exam dates are set by the central admin. We don't have any control/influence over them.

IMPORTANT: The exam format will be explained in the last lecture (L10)!

“Early info” for the exams

- **Exams will be in-person using Artemis**
- We **evaluate your problem-solving skills** rather than memorizing ability
  - The focus will be on evaluating your **skills to understand the given problem, and solve it by system design and implementation**
  - The exam will require **programming**
  - You will need a **computer with Internet connection** for the **exam**
    - If you don’t have a computer with internet connection, we will provide you an option to take the exam from Rechnerhalle *(More information later!)*
- **Exercises** will (indirectly) help you prepare for the exam
- We also plan to **give you a practice exam** after the last lecture
  - This will also help you to prepare for the exam

Exam format

- **(3-4x) System design and implementation questions**
  - Each question will have **2-3 tasks**
  - You need to **write code** to solve the problem
  - **A working solution** is must to evaluate the task
  - Exam will be **auto-graded** using the test-cases
- **Grading key:** The exact grading key will be based on the overall exam performance
  - **If you score 50% points in the exam, you will receive at least 4.0 (pass grade)**
- More details in the last lecture!

Important: All exam information is tentative and subject to change

An appeal

- Focus on **learning and mastering the topics**
  - Please go beyond the topics covered in the lecture
  - We are just scratching the surface as part of the lectures
  - **There is so much to learn!**
- We are offering a state-of-the-art curriculum
  - Covering the best practices for software and systems engineering in the cloud!
- Overall, recommendation:
  - **Please don’t optimize your learning experience just for the grades!**

E: Tools (Students)

- **Zulip** (for online Q&As)
  - **EIST - general:** General questions
  - **EIST - homework:** Homework related questions

- **Artemis** (for course material): https://artemis.in.tum.de/
  - Lecture and tutorial slides
  - Programming exercises: PB, P, PE

- **Live streaming/video recording** of the lectures
  - https://live.rbg.tum.de/

- **Wiki** (for tutorial info and forms): https://collab.dvb.bayern/display/TUMeist/EIST

- **Website** (for general info): https://dse.in.tum.de/teaching/

Important: Please **NO** E-mail communication

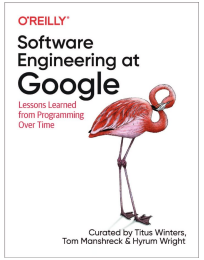
E: Tools (Zulip for Tutors only)

- **Zulip workspace**
  - Join with your TUM IDs

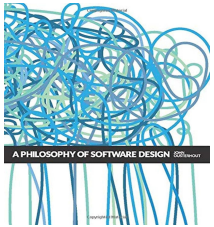
- **Zulip channels:**
  - **All tutors:** #EIST - all-tutors
  - **Tutorials:** #EIST - tutorials
  - **Exercises:** #EIST - exercises
  - **Exams:** #EIST - exams

- **Important:** No E-mail communication (also for tutors)!

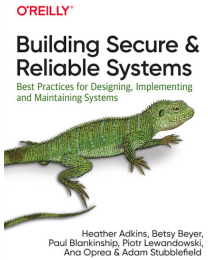
F: Text books



"Software Engineering at Google"
— By: Hyrum Wright, Titus Winters, and Tom Manshreck
— Available in the library and also online: https://abseil.io/resources/swe-book



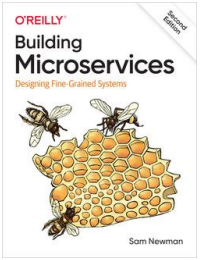
"A Philosophy of Software Design"
— By: John Ousterhout
— Available in the library



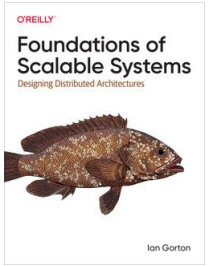
"Building Secure and Reliable Systems"
— By: Heather Adkins, Betsy Beyer, Paul Blankinship, Ana Oprea, Piotr Lewandowski, Adam Stubblefield
— Available in the library and also online:
https://sre.google/books/building-secure-reliable-systems/

Text books

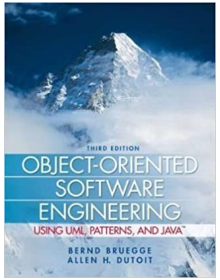
“Building Microservices: Designing Fine-Grained Systems”
— By: Sam Newman
— Available in the library and also online (Check Univ’s digital library)
https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/



“Foundations of Scalable Systems”
— By: Ian Gorton
— Available in the library and also online (Check Univ’s digital library)
https://www.oreilly.com/library/view/foundations-of-scalable/9781098106058/



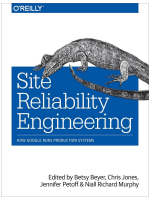
“Object-Oriented Software Engineering Using UML, Patterns, and Java”
— By: Bernd Bruegge, Allen H. Dutoit
— Available in the library



Text books

“Designing Data-Intensive Applications”
— By: Martin Kleppmann
— Available in the library and online:
https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/

“Site Reliability Engineering”
— By: Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy
— Available in the library and also online: https://sre.google/books/



“Effective DevOps”
— By Jennifer Davis, Ryn Daniels
— Available in the library and also online:
https://www.oreilly.com/library/view/effective-devops/9781491926291/

Basic tools

- Learn basic tooling for your homework workflow
  - A skill broadly required for almost all tasks in this course, and all practical courses in your study program!
- Swiss-knife for your CS education
  - https://missing.csail.mit.edu/
    - Shell scripting
      - https://missing.csail.mit.edu/2020/shell-tools/
    - Command line
      - https://missing.csail.mit.edu/2020/command-line/
    - Debugging/profiling
      - https://missing.csail.mit.edu/2020/debugging-profiling/
    - Version control (Git) – we will also cover in L08
      - https://missing.csail.mit.edu/2020/version-control/

Additional references

- We will also provide additional references
  - Online blogs
  - Lecture material from other courses
  - Selected chapters from books

Code of conduct

- University plagiarism policy
  - https://www.in.tum.de/en/current-students/administrative-matters/student-code-of-conduct/

- Decorum
  - Promote freedom of thoughts and open exchange of ideas
  - Cultivate dignity, understanding and mutual respect, and embrace diversity
  - Racism and bullying will not be tolerated

IMPORTANT: Action items

- **Exchange students**
- **Frühstudium**
- **Nachteilsausgleich**
- **Rechnerhalle-exam**

IMPORTANT: We will release these forms at the appropriate time via Email.

- Please fill these forms, if required, by the **specified due dates**
- **NO EXCEPTION** will entertained after the due date!
- Submissions **by email will NOT be entertained!**

Outline

- ~~**Part 0:** (*a quick!*) Introduction to cloud computing~~

- ~~**Part I:** Administrative~~

- **Part II:** Introduction to software engineering
  - **Terminology and software engineering activities**
  - Abstraction and system modeling with UML

- **Part III:** Course overview

- **Part IV:** Software engineering models

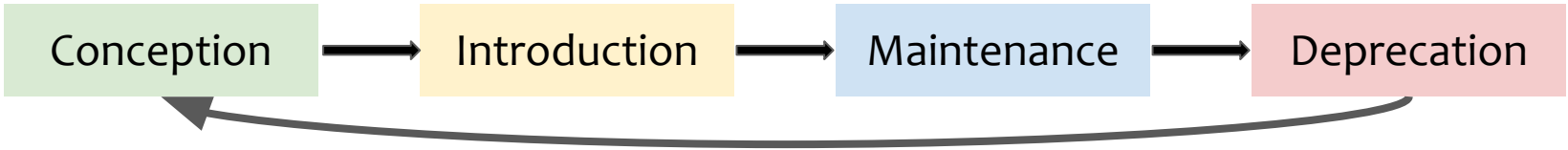
Programming vs Software engineering

- Have you done programming? **I believe yes!**
  - Writing code and maintaining your own projects is fun!
  - *You should continue to have your own pet projects – Please maintain a GitHub Profile!*

- What distinguishes **“software engineering”** from **“programming”**?
  - Software Engineering → **Programming Over Time**

Software engineering

- **"Software engineering"** is not just the act of writing code, but all of the **tools** and **processes** an organization uses to build and maintain that **code over time**
  - **Engineering best practices** for keeping the software sustainable over its **entire life cycle**, from **conception** to **introduction** to **maintenance** to **deprecation**



Software life cycle

Software engineering is more than programming

- **Problem solving**
  - Understand the application domain and problem
  - Propose a solution and a plan (often called prototype)
- **Dealing with complexity**
  - Creating abstractions and models
  - Engineer a system based on the proposed solution using a good design
- **Dealing with change**
  - Adapting to the changing requirements
- **Employing the best practices**
  - Engineering the system based on good practices to meet the (non-)functional requirements

This course

- Software engineering: **a problem solving activity**
  - **Requirement elicitation and analysis:** Understand the nature of the problem
  - **Design and implement:** How to build “elegant yet simple” systems?
  - **Validation of the system:** How to validate and verify the system design?
  - **Build, deployment, and maintenance:** How to build, deploy, and maintain systems?
  - **Process management:** How to manage the overall project?

As a software engineer in an organization

- **Time and change**
  - How code will need to adapt over the length of its life?
- **Scale and growth**
  - How an organization will need to adapt as it evolves?
- **Trade-offs and costs**
  - How an organization makes decisions, based on the lessons of **time and change** and **scale and growth**?

Outline

- ~~**Part 0:** (*a quick!*) Introduction to cloud computing~~

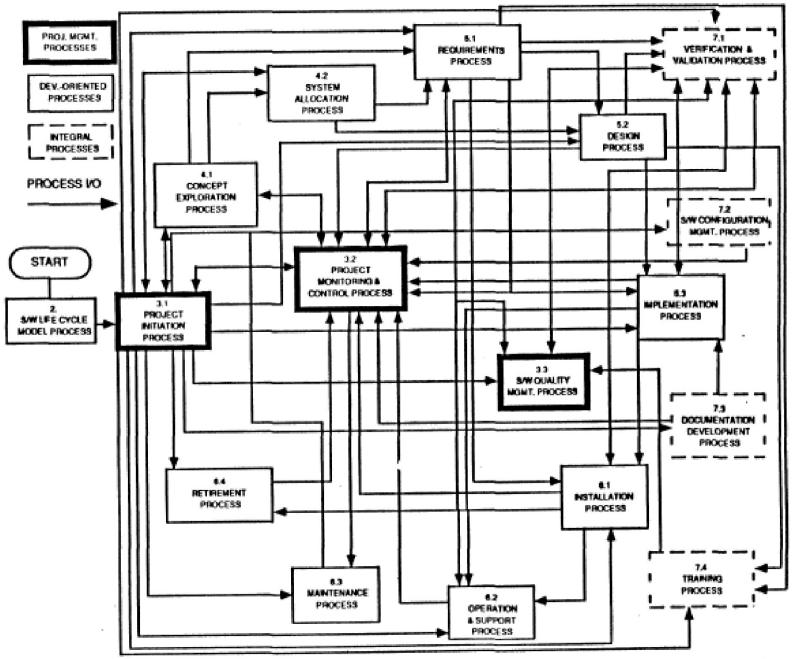
- ~~**Part I:** Administrative~~

- **Part II:** Introduction to software engineering
  - Terminology and software engineering activities
  - **Abstraction and system modeling with UML**

- **Part III:** Course overview

- **Part IV:** Software engineering models

What is the problem with this diagram?

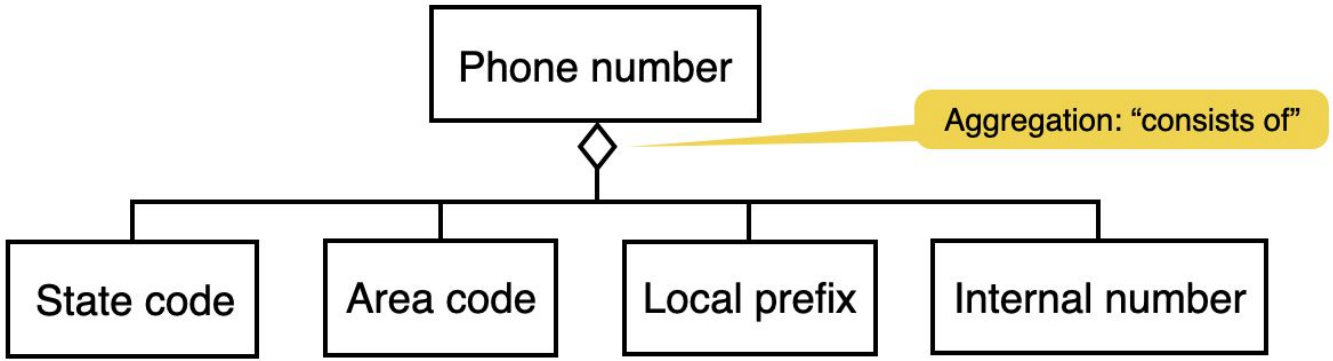


It's too complex!

Solution: Abstraction!

Abstraction

- Complex systems are hard to understand end-to-end
- Abstractions allow us to ignore unnecessary details
- **Example:**
  - Phone numbers with more than 9 digits are complex!
  - Chunking Group collection of objects to reduce complexity
  - A phone number consists of a state code, area code, local prefix, internal number



Abstractions in computer systems

Abstraction in computer systems simplifies complexity

Manages complexity

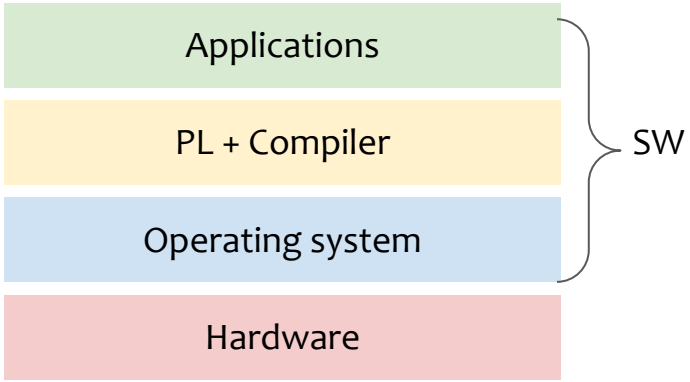
- **Divides complex systems** into smaller manageable parts
- Hides **implementation details** focusing on “what” not “how”

Improves modularity & reusability

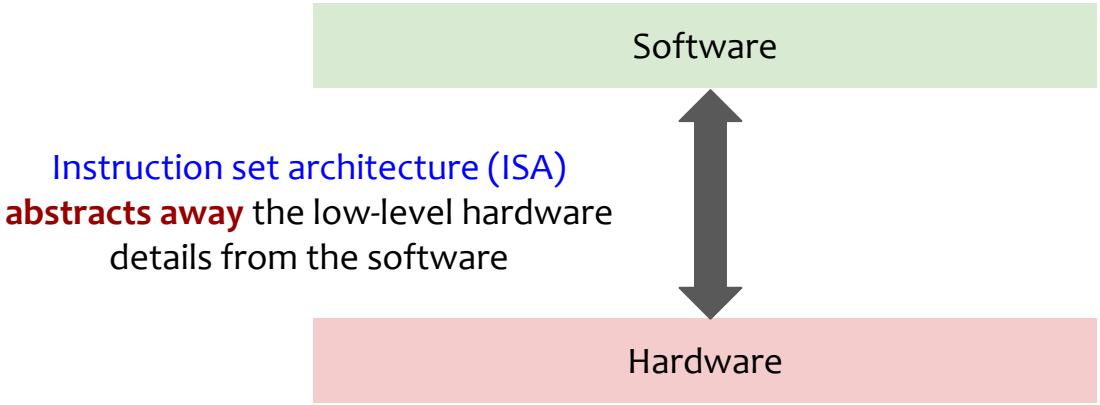
- Creates **independent reusable modules** with clear interfaces

Enhances maintainability & Scalability

- **Reduces the impact of changes** through hidden implementation

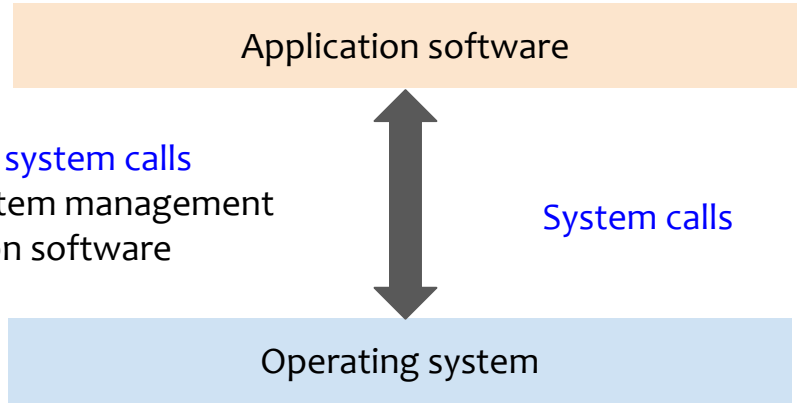


Example: Hardware-Software interface



Example: OS-application interface

Operating systems (OS) via system calls **abstracts away** the low-level system management details from the application software



Example: Application programming interface (APIs)

Application programming interface (APIs) **abstract away** the internal details of a system component by exposing a well-defined interface

Abstraction

- Abstraction allow us to implement the **information hiding principle**
  - Abstraction focuses on **"what"** an object or module does
  - We can abstract out **"how"** an object or module achieves its functionality

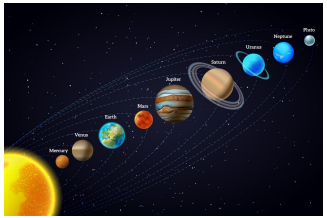
- Abstractions can be expressed with **a model**

Model

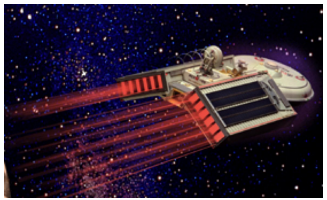
Abstraction of a system

- A system that no longer exists

- An existing system



- A future system to be built



Advantages of system modeling

- **Simplifying abstraction:** It reduces complexity by focusing on abstractions

- **Modular source development:** It can be seen as a high level programming language enabling the generation of source code – The grand promise!

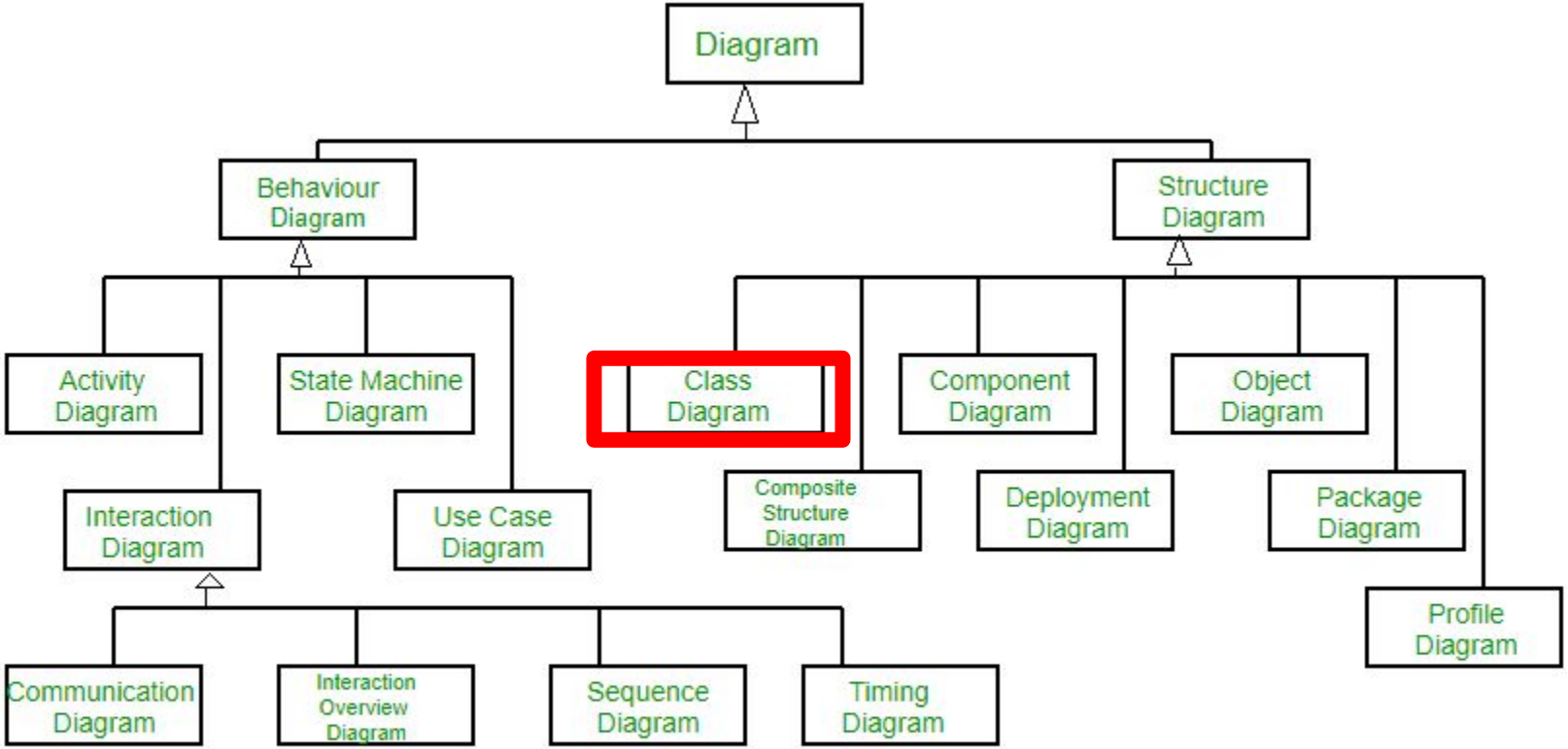
- **Communication:** It is a means of communication between people involved in a software project

- **Documentation:** It can act as a documentation for requirement analysis, system design, and implementation

UML

- **UML (Unified Modeling Language)** A standard for modeling software systems by the Object Management Group: http://www.uml.org
  - **Communication:** provides a common vocabulary for communication (“informal model”, “conceptual model”)
    - **Target:** human (developer, end user)
  - **Analysis and design:** models enable developers to specify a future system
    - **Target:** tool (code generation tool, compiler)
  - **Archival:** models provide a way for storing the design and rationale of an existing system
    - **Target:** human (analyst, project manager)

Types of UML diagrams



These UML diagrams capture a different level of abstraction

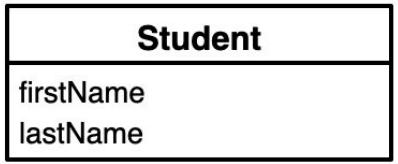
UML class diagrams

- Show the **structure of classes** and **their dependencies** to communicate the main concepts of a software system
- **Classes:** representation / blueprint of an object which defines its **structure (attributes)** and its **functionality (methods)**
- **Associations:** define the **relationships between classes** and their corresponding dependencies / hierarchies
- Class diagrams can be used in different phases
  - **During requirements analysis** to model application domain objects
  - **During system design** to model solution domain objects
  - **During system implementation** to specify the detailed behavior and the types of attributes of classes

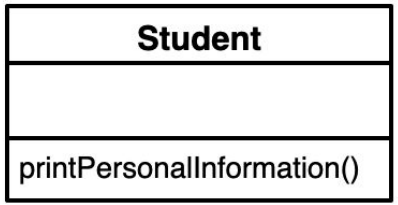
UML class diagrams

Class

Attributes



Methods



Associations

Inheritance

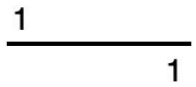
Aggregation

Composition

Reference

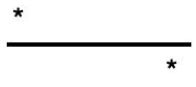
Multiplicities

one to one

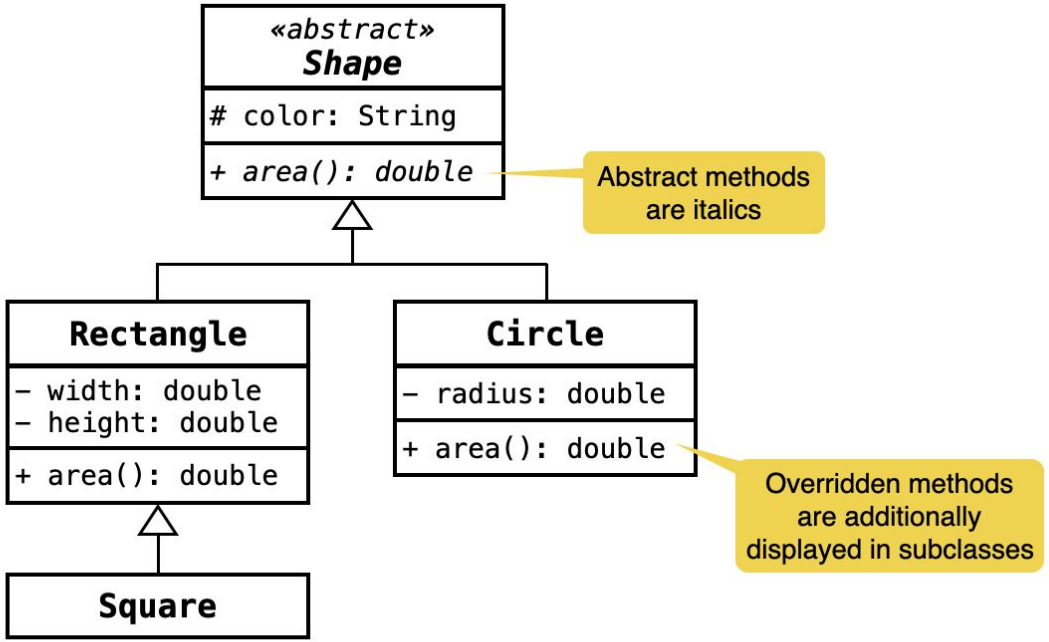


one to many

many to many



An example class diagram



More readings on UML

- **Class diagrams**
  - https://www.tutorialspoint.com/uml/uml_class_diagram.htm
- **Other relevant UML diagrams** (for self-study)
  - Use-case diagrams: https://www.tutorialspoint.com/uml/uml_use_case_diagram.htm
  - Component diagrams: https://www.tutorialspoint.com/uml/uml_component_diagram.htm
  - Deployment diagrams: https://www.tutorialspoint.com/uml/uml_deployment_diagram.htm
  - Activity diagrams: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm

Outline

- ~~**Part 0:** (*a quick!*) Introduction to cloud computing~~

- ~~**Part I:** Administrative~~

- ~~**Part II:** Introduction to software engineering~~

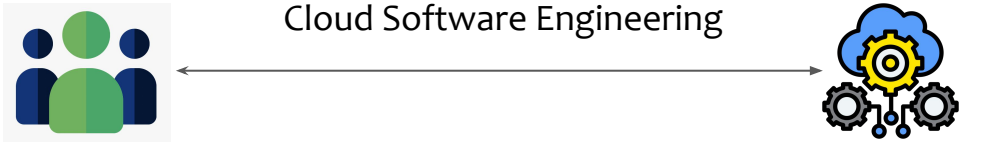
- **Part III:** Course overview
  - **Course focus:** Cloud software engineering
  - An overview of the software engineering activities in the cloud

- **Part IV:** Software engineering models

Course roadmap

Course Focus:

Cloud Software Engineering



L03 + L04 + L05
System Design and Implementation

L06 + L07
Software Testing and Program Analysis

L08
Software Management, Build Systems, & DevOps

L09
Software Deployment and Monitoring

L10 Software Quality and Project management

L02 System Design Requirements and Software Architectures

L01 Introduction

Course website

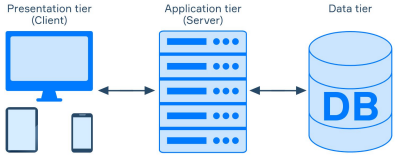
Detailed course content is available:

https://dse.in.tum.de/teaching/eist-25/

L02: System Design Requirements & Software Architectures

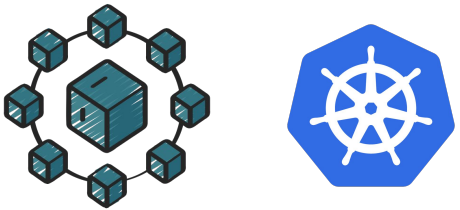
- System design requirements
  - Requirements engineering
  - Functional vs non-functional requirements

- Client-server architecture
  - RPC
  - REST-based application



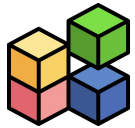
- Monolithic architecture
  - A simple three-tier architecture
  - Limitations of monolithic architectures

- Microservices architecture
  - Advantages
  - Google's service weaver



L03: System Design I

- System design challenges
  - The quest for simplicity!



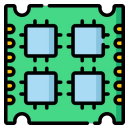
- System decomposition
  - Modularity
  - Interface design
  - Layered architectures

- Data management
  - KV stores, databases, shared logs, filesystems

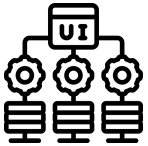
L04: System Design II

- Performance
  - How to design high-performance systems?

- Concurrency (Scale-up!)
  - Concurrent/parallel systems



- Scalability (Scale-out!)
  - Distributed systems



L05: System Design III

- Security
  - How to design secure systems?

- Reliability
  - How to design fault-tolerant systems?

- Availability
  - How to design highly available systems?

L06: Software Testing

- Software testing
  - Unit, integration, and system testing
  - Unit testing tool
- Mock testing
  - Testing doubles
- Large-scale automated testing
  - Fuzzing
  - Crash Monkey

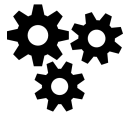
L07: Program Analysis

- Program analysis for software quality
  - **Static analysis**
    - Methods and tools
  - **Dynamic analysis**
    - Methods and tools

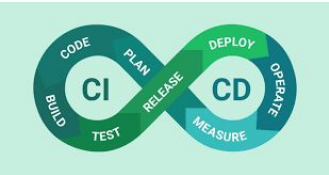
L08: Software Management, Build Systems, & DevOps

- Source code management
  - How to **manage source code** using a version control?
  - How to incorporate **code review**?

- Build and release management
  - How to **build and release large software systems**?

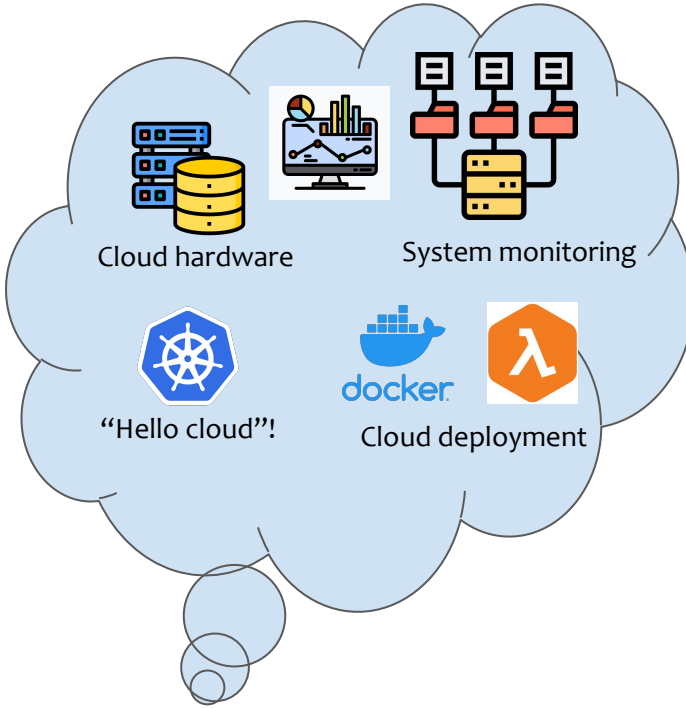


- Continuous-*
  - Integration
  - Testing
  - Deployment



L09: Software Deployment and Monitoring

- Cloud deployment models
  - Bare metal, Virtual machines, containers, serverless
- Hello cloud in the cloud
  - Designing and deploying a simple Docker app
- System monitoring
  - Monitoring, visualizing, and altering systems



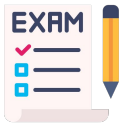
Cloud computing!

L10: Software quality and project management

- Software quality
  - How to **improve software quality?**
  - How to build **trustworthy software systems**?

- Project management
  - How to **manage software engineering teams**?
  - What are the different **modes of communication**?

- Exams!
  - Exam format and information

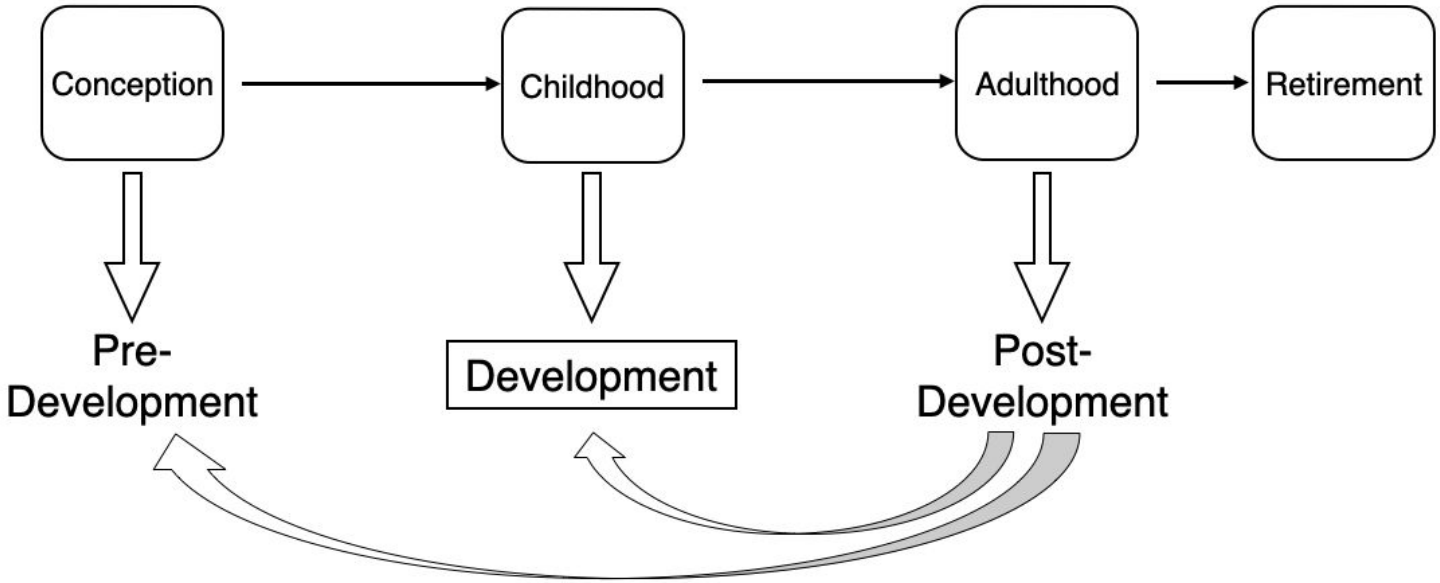


Outline

- ~~**Part 0:** (*a quick!*) Introduction to cloud computing~~
- ~~**Part I:** Administrative~~
- ~~**Part II:** Introduction to software engineering~~
- ~~**Part III:** Course overview~~
- **Part IV:** Software engineering models
  - **Defined (e.g., Waterfall model) and empirical methods (e.g., Scrum)**
  - **Agile methods in software engineering projects (Scrum)**
  - A note on tactical vs strategic programming

Software development lifecycle

- Based on the metaphor of the life of a person



Software development lifecycle

- **Software development lifecycle**
  - Set of **activities** and their **relationships** to each other to support the development of a software system
  - **Examples of activities:** requirements elicitation, analysis, system design, implementation, testing, software configuration management, delivery
  - **Examples of relationships:** implementation must be done before testing, analysis, and system design can be done in parallel

Software engineering models

- How do we manage software development?

- Two opinions:
  - **Defined processes:**
    - **Through organizational maturity** (Humphrey 1989)
    - Repeatable process, Capability Maturity Model Integration (CMMI)
  - **Empirical processes:**
    - **Through agility** (Schwaber 2001)
    - Large parts of software development are empirical in nature
    - They cannot be modeled with a defined process

Defined vs empirical models

Defined models

Fully planned ahead
Follow strict rules
Avoid deviations

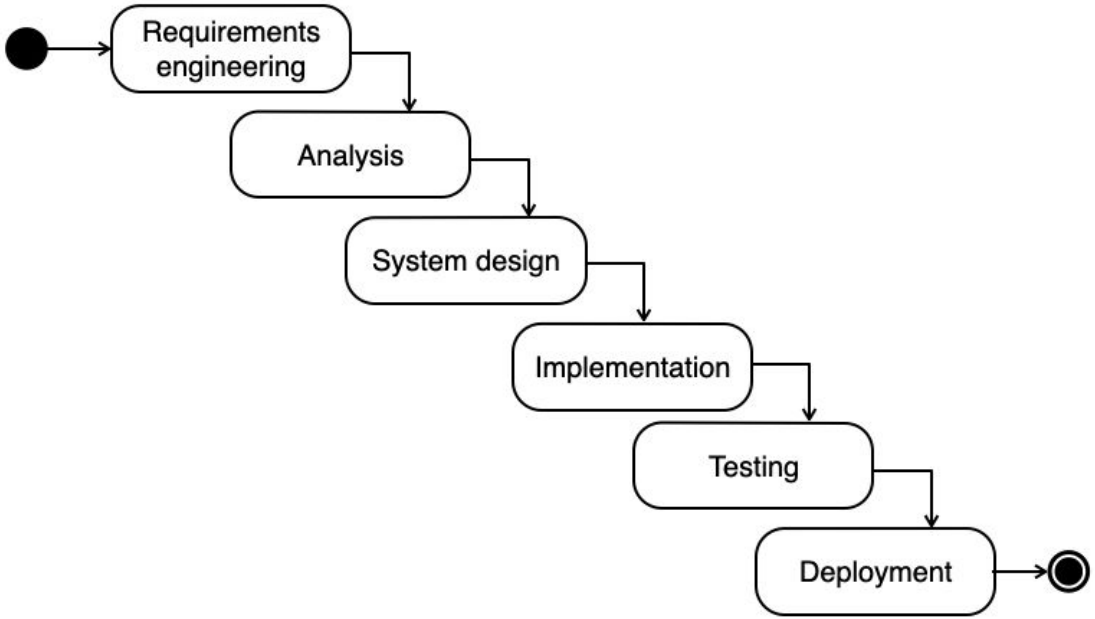
Empirical models

Not fully planned
Inspect the progress/requirements
Adapt to changes

Defined process example: The waterfall model

- **The waterfall model**
  - First described in 1970 by Royce
- **Sequential execution** of software activities
  - Assumes that software development can be scheduled as a step-by-step process that transforms user needs into code
  - You cannot turn back once an activity is completed
- **Simplistic view of software development**
  - Follows the defined plan at the start of project
  - Cannot adapt to the changing requirements

Waterfall model



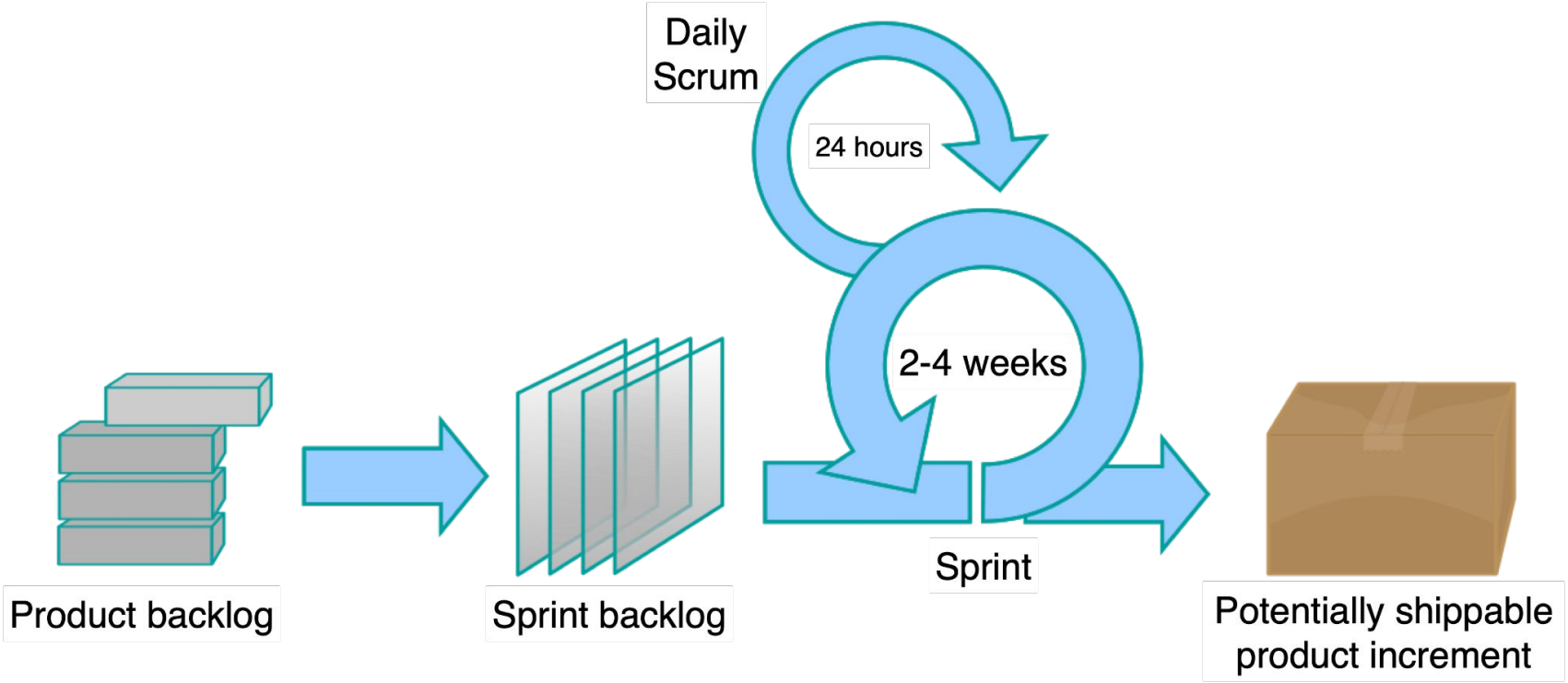
The waterfall model is obsolete and it is usually not employed in practice!

Scrum: An agile software development model

- Scrum is based on **empirical process control**
  - An **iterative** and **incremental** approach for software development

- **Main principles**
  - **Transparency:** process and work must be visible to those performing the work and those receiving the work
  - **Inspection:** artifacts and progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems
  - **Adaptation:** if any aspects of a process deviate outside acceptable limits, the process must be adjusted

Scrum life cycle



Source: https://commons.wikimedia.org/wiki/File:Scrum_process.svg

Scrum artifacts

- **Product backlog**
  - List of requirements for the whole product
- **Sprint backlog**
  - List of requirements and tasks for one iteration (“sprint”)
- **Potentially shippable product increment**
  - Release to the product owner that contains all results of the current sprint

A case study: Let’s design Google Maps

Product vision:

Our overarching vision is to create a user-friendly, reliable, and feature-rich mapping and navigation application that helps users explore the world around them. This includes core functionalities like **viewing maps**, searching for locations, **getting directions**, and potentially incorporating features like **real-time traffic** updates, points of interest, and offline map capabilities.

Product backlogs

I. Viewing Maps

1. As a user, I want to see **a basic map of Garching** displayed so that I can get a general overview.
2. As a user, I want to **zoom in and out** smoothly so that I can see more or less detail.
3. As a user, I want to **see my current location** marked so that I can orient myself.

II. Getting Directions

4. As a user, I want to **enter start and end points** and see a driving route appear so that I know the way.
5. As a user **getting driving/walking directions,** I want to see the estimated time and distance so that I can plan.

III. Real-time Traffic

6. As a user viewing the map, I want to see **traffic on Garching roads** visually so that I understand congestion.
7. As a user getting driving directions, I want **the ETA to adjust for current traffic** so that I have an accurate prediction.
8. As a user getting driving directions, I want to see **faster alternative routes** if traffic is heavy so that I can save time.

Sprint: Viewing map

Sprint Goal: Show a basic, interactive map of Garching with the user's location.

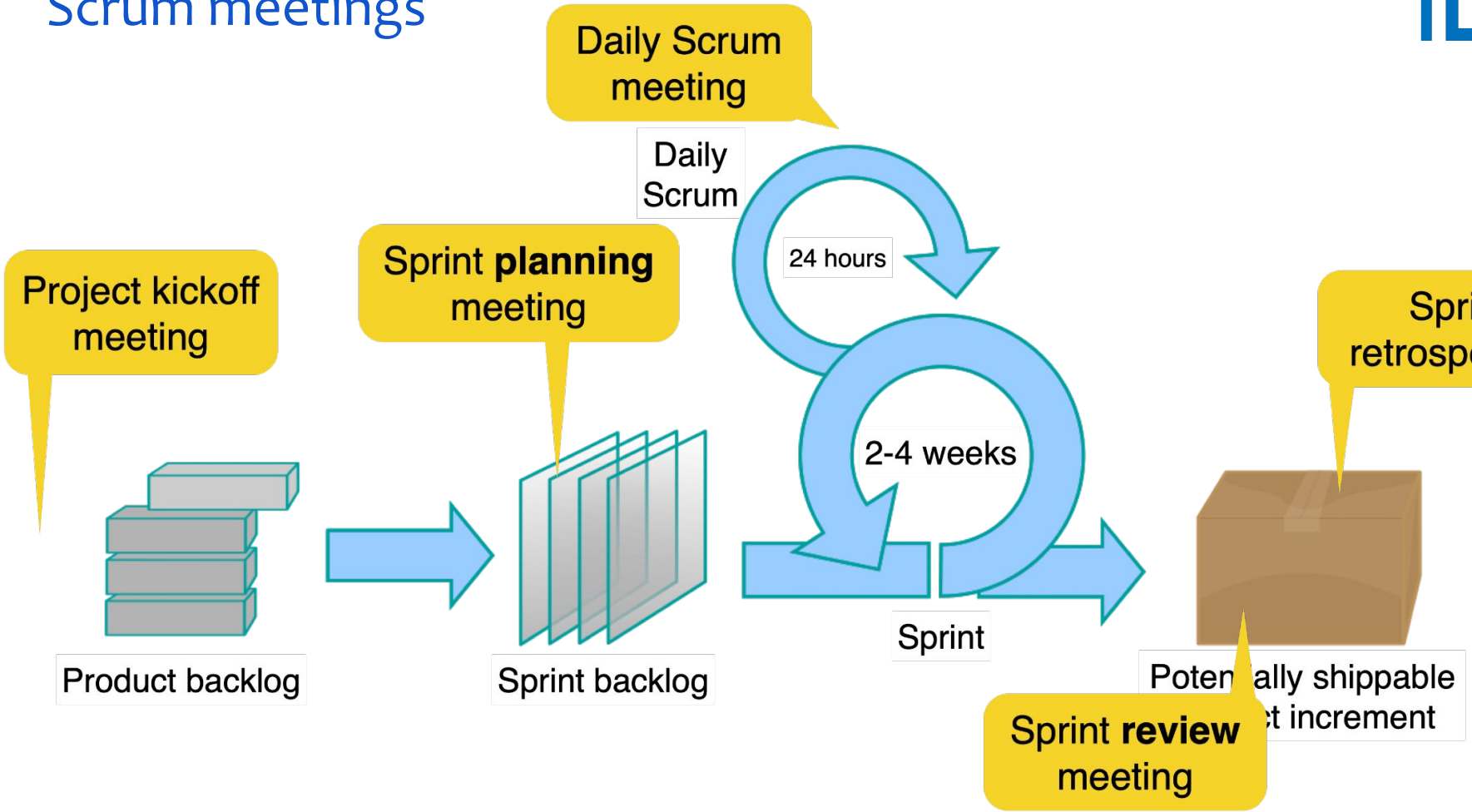
Sprint Backlog Items:

1. **Show basic Garching map**
   - **Task 1.1:** Set up the map SDK and center on Garching.
   - **Task 1.2:** Display the initial map within the application UI.
2. **Implement smooth zoom**
   - **Task 2.1:** Enable map zooming functionality using the SDK.
   - **Task 2.3:** Ensure smooth transitions and detail adjustment during zoom.
3. **Mark current location**
   - **Task 3.1:** Get user permission and retrieve current device coordinates.
   - **Task 3.2:** Display a clear marker at the user's location on the map.

Potentially shippable products

- **Sprint 1** Increment:
  - Basic Garching Map Viewer
- **Sprint 2** Increment:
  - Garching Directions (Driving & Walking)
- **Sprint 3** Increment:
  - Garching Traffic-Aware Navigation

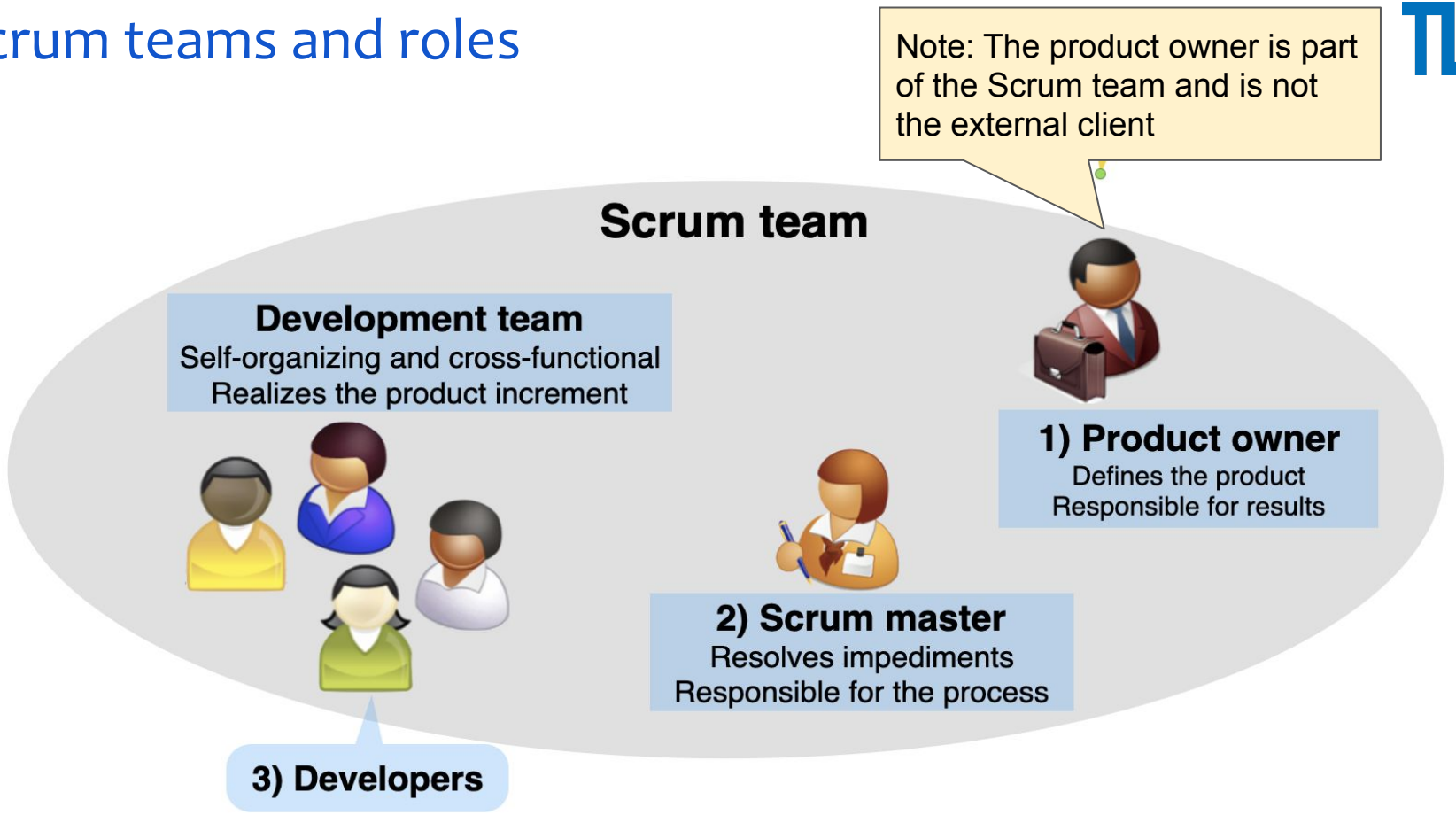
Scrum meetings



Scrum meetings

- Project kickoff meeting (start of the project)
  - Create and prioritize the product backlog
- Sprint planning meeting (start of each sprint)
  - Create the sprint backlog
- Daily Scrum meeting (every day, 15min)
  - Share status, impediments and promises (in a standup meeting)
- Sprint review meeting (end of each sprint)
  - Demonstrate the realized backlog items to the product owner (and other stakeholders)
- Sprint retrospective (after the sprint)
  - Inspect the previous sprint and create a plan for improvements to be enacted during the next sprint

Scrum teams and roles



Scrum teams

- **Product Owner (PO):** Responsible for defining the product backlog, prioritizing features based on user needs and business value, and ensuring the team understands the "what" and "why" of the product

- **Scrum Master:** Facilitates the Scrum process, removes impediments, and ensures the team adheres to Scrum practices

- **Development Team:** A self-organizing group of **developers, designers, testers,** and **other specialists** needed to build and deliver the product

References for Scrum

- Read the following articles to find out more about agile methods and Scrum
  - The Scrum guide: https://scrumguides.org/scrum-guide.html
- Scrum methodology
  - https://www.wrike.com/scrum-guide/scrum-methodology

Outline

- ~~**Part 0:** (*a quick!*) Introduction to cloud computing~~
- ~~**Part I:** Administrative~~
- ~~**Part II:** Introduction to software engineering~~
- ~~**Part III:** Course overview~~
- **Part IV:** Software engineering models
  - Defined (e.g., Waterfall model) and empirical methods (e.g., Scrum)
  - Agile methods in software engineering projects (Scrum)
  - **A note on tactical vs strategic programming**

A note on tactical vs strategic programming

- Agile methodology encourages a **tactical mindset**
  - **Hack it up!:** Implement the features as quickly as possible
  - Working code is better than a good system design

- **Strategic approach** encourages a good system design
  - **Long-term mindset:** Better maintainability, reliable and secure design

Tactical programming

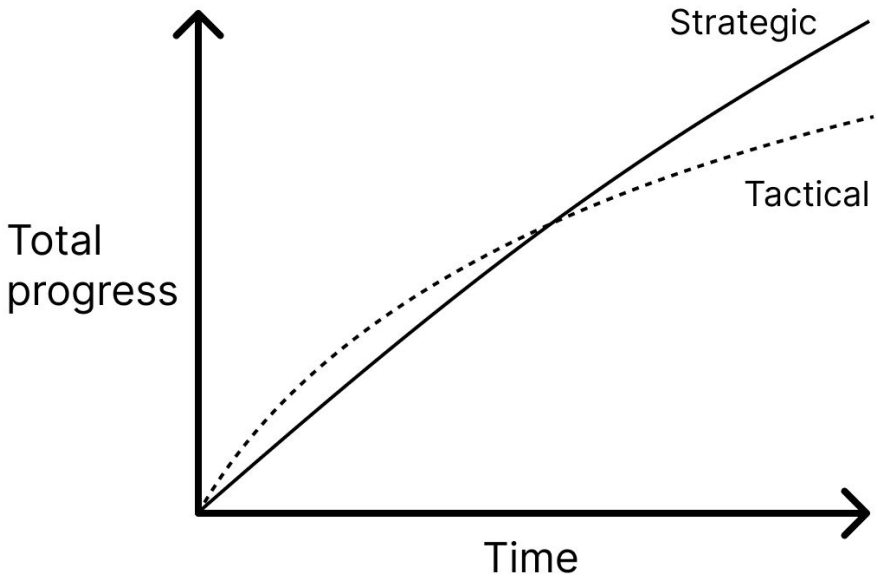
- Main focus is to get the code working
  - A new feature or fix a bug

- **Short-sighted design**
  - Leads to system complexity (and bad system design)
  - In the long term, negatively affects the software quality

Strategic programming

- **Long-term investment mindset:** Working code isn't enough!
  - Minimize complexity,
  - Better software quality: reliability and security
- **For example:**
  - Simple and clean interfaces
  - Discussion and trade-off analysis of alternative design choices
  - Writing good documentation
  - How to improve system maintainability, extensibility, supporting new features
  - Focus on improving collaborative software development

How much to invest?



A tactical approach to programming will make faster progress in the beginning, but the progress will degrade over time as the complexity of the system increases. Over a long time, the strategic approach results in greater progress as past investments constantly yield faster development times for new features.

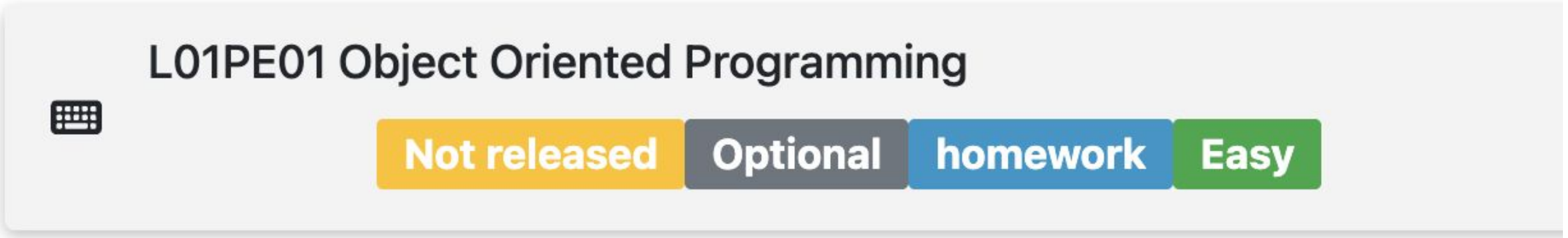
References

- Book: A philosophy of software design by John Ousterhout
  - Chapter 3: Working Code Isn't enough (Strategic vs tactical programming)
- Online blogs on the same topic:
  - https://levelup.gitconnected.com/strategic-vs-tactical-programming-and-which-is-better-b89934504822
  - https://arunas.dev/write-better-laravel-why-working-code-is-not-enough/

Summary

- **Part 0:** (*a quick!*) Introduction to cloud computing
- **Part I:** Administrative
  - Introduction to the course team
  - Understand the course organization
- **Part II:** Introduction to software engineering
  - Terminology and software engineering activities
  - Abstraction and system modeling with UML
- **Part III:** Course overview
  - **Course focus:** Cloud software engineering
  - An overview of the software engineering activities in the cloud
- **Part IV:** Software engineering models
  - Defined (e.g., Waterfall model) and empirical methods (e.g., Scrum model)
  - Agile methods in software engineering projects (Scrum)
  - A note on tactical vs strategic programming

Homework (NOT graded)



https://artemis.tum.de/