UNIVERSITY OF COLOGNE | Software & System Engineering / Software Technology
Abteilung Informatik, Department Mathematik / Informatik

Object-Oriented Software Engineering

Organization & Introduction

Adrian Bajraktari | bajraktari@cs.uni-koeln.de | SoSe 2024 | 08.04.2024

Welcome to the Lecture!

The Software & System Engineering Chair

Andreas Vogelsang

Requirements Engineering

- Classification
- Trace Links
- Test Case Derivation

Mersedeh Sadeghi

Explainable Intelligent Systems

- Explainable System Requirements
- SE for Explainable Systems
- SmartLab "SHINE"

Topics on Hold

- Model-based Systems Engineering
- Data-Driven Systems Engineering

Adrian Bajraktari

Research Software Engineering

- Code Quality
- Methodologies
- Reproducibility
- Team and Process

Alexander Korn

TBD

- TBD

Student Assistants

- Michelle Binder
- Simon Scholz
- Anna Trapp
- Max Unterbusch
- Annika Vogt

The Software & System Engineering Chair: Teaching

Winter Term	Summer Term
Lecture Software Engineering (9 ECTS, Bachelor)	Lecture Requirements Engineering (9 ECTS, Master)
Capstone Project (15 ECTS, Bachelor)	Lecture Empirical Software Engineering (6 ECTS, Master)
Advanced SE Projects (9 ECTS, Master)	Lecture Object-Oriented Software Engineering (6 ECTS, Master)
Bachelor’s and Master's Thesis Topics: https://cs.uni-koeln.de/sse/theses	

The Team: You

- Which programming languages have you learned?
- Who has taken a Software Engineering course?
- Who is from the computer science master?
    - Who are the others?
- Who has experience in a real project? (uni or work)

Communication

Discord will be our central instrument for announcements, questions, discussions, communication.

- You are more than welcome to
  - Comment on and add to anything
  - Raise organizational questions in channel **#orga-questions**
  - Raise questions on lecture content in channel **#lecture-questions**
- Use of real name is mandatory

Please do not send any messages or mails to Prof. Dr. Felderer!

Who is this lecture for?

Not a *pure* programming course.

Not *not* a programming course.

Important Information

Lecture and Exercise

- Tuesdays (lecture), 14:00-15:30
- Fridays (lecture or exercise), 14:00-15:30
- Always in Building 321 Lecture Hall III

Effort and Prerequisites

- 6 ECTS → 4 SWS presence + 4 SWS self-study.
- No formal prerequisites, but
  - Programming Course,
  - Programming Lab and
  - Software Engineering are recommended.

Written Exam, 60 Min, in English

- 31.07.2024, 13:00 – 14:00, Physik Hörsaal 2.
  - Inspection: 02.08.2024, 10-11, Sibille-Hartmann Str. 2-8 1.421
- 23.09.2024, 09:30 – 10:30, Physik Hörsaal 2.
  - Inspection: 25.09.2024, 10-11, Sibille-Hartmann Str. 2-8 1.421

Exam Preparation

- One rehearsal exam in July, after all exam relevant topics.
- One hand-written cheat sheet (both sided) allowed.
- Q&A Session in last lecture.

Exam Admission and Bonus

- No exam admission criteria.
- Homework handout bi-weekly on Friday evening, hand-in two weeks later Friday, 14:00, groups up to 5 people.
- You need >=50% points in the exam to pass. The bonus will only be counted if you pass the exam.
- 5% bonus in the exam if you have >= 80% of homework points.
- 10% bonus in the exam if you have >= 95% of homework points.

Components of the Course

Conveying Knowledge

- Lectures
- Script with additional material
- Case studies, real world examples

Exercise Knowledge

- Integrated "global" exercises with on-site exercises (with example solutions)
- Homework sheets (graded)
- Self-tests in Ilias (with direct feedback)
- Additional exercises (with example solutions)

Exercise

- "Global"
- Content:
  - Discussion of homework.
  - Extensive examples.
  - On-site exercises.
  - Case-studies.
  - ...
  - Outlook on next homework.
- You will need a laptop or any device you can develop software on.

Material

All material will be shared in Sciebo. The link is in ILIAS.
Preliminary lecture slides will be published before the lecture (usually, on Mondays, but no promises).
Slides may be updated and extended after lectures, so be aware. Updates will be announced.

A Word on Example Solutions

Disclaimer: Sample Solutions

Previously, we never handed out any "sample solutions" in any of our lectures. Reasons:

- In SE, there is never "the one" solution. A sample solution would only discourage alternate, but correct solutions.
- Sample solutions are always seen as correct, leading to discussions during exam inspection.
- Sample solutions tempt students to not do the exercises on their own.
- In follow-up semesters, it would be unfair if people hand-in sample solutions and get extra points in the exam.

This time, we will give you sample solutions for everything, except graded homework (where you will get direct feedback).

Disclaimer: These solutions are **never**

- 100% checked for completeness
- 100% checked for consistency
- 100% checked for errors
- the *only* possible solution.

Take these solutions as a direction, but not as the final word. We do not give warranty. We will not accept any discussions like "but in the solutions...".

Programming Languages in this Lecture

Java

Enterprise Information Systems

Main Language

Embedded Systems Gaming

Data Science and Machine Learning

Language of the Web

With a bit of

Scala

SWI Prolog

Infrastructure

We recommend to use JetBrains Toolbox IDEs for this lecture:

- IntelliJ for Java and Kotlin
- PyCharm for Python
- CLion for C++
- WebStorm for JavaScript

All exercises have been tested and designed for these IDEs.
JetBrains products are free to use in Community Edition and in Ultimate Edition for students.

If you do not want to use JetBrains products, you can use any IDE you like but we can not guarantee a flawless experience.

What's next?

Follow-Up Module: Advanced Software Engineering

Concept

Development of a real project.

- Your ideas, or
- One of our projects, or
- Some other organization's project.

Can be from scratch or continuation of existing ones.

Format

- 9 ECTS, project-based, agile.
- No lectures, no exercises, but a meeting at the end of each sprint.
- No examination; instead, project and final presentation.

Who can participate?

The module will have a maximum capacity of 1-2 teams, 7 people each.

We are especially interested in students that actively contact us. If there are more applicants than places, results of the OOSE, SE and RE lectures will be used to decide who can participate.

Follow-Up Module: Advanced Software Engineering

Content

- Development in an agile team via git.
- Infrastructure: GitLab, Teamscale, other…
- You learn not only agile SE, but also something about the application domain.

How to pass

- Deliverables:
  - Hand in of final project.
  - Presentation.
- Expectations: Code Quality, process quality, documentation.

Time Schedule

- 3 weeks ideation,
- 5x2 weeks sprints,
- 2 weeks preparation for presentation.

Roles

- Prof. Vogelsang and I are Scrum masters and partly product owner.
- You are the dev team and the other part of the product owner.

During Sprints

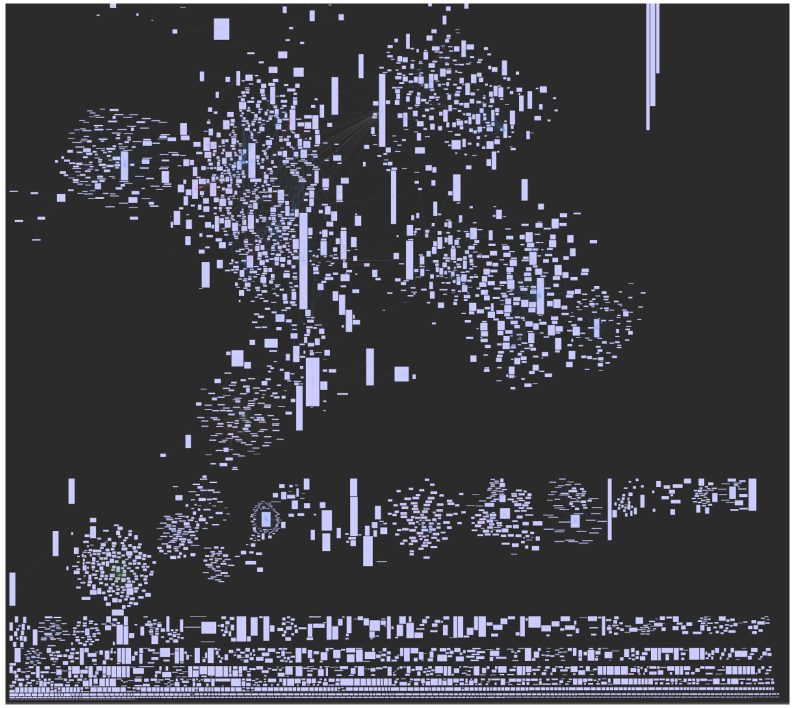
- Document and manage requirements,
- Design, develop, test and deploy solutions,
- Present sprint results, retrospective and planning.

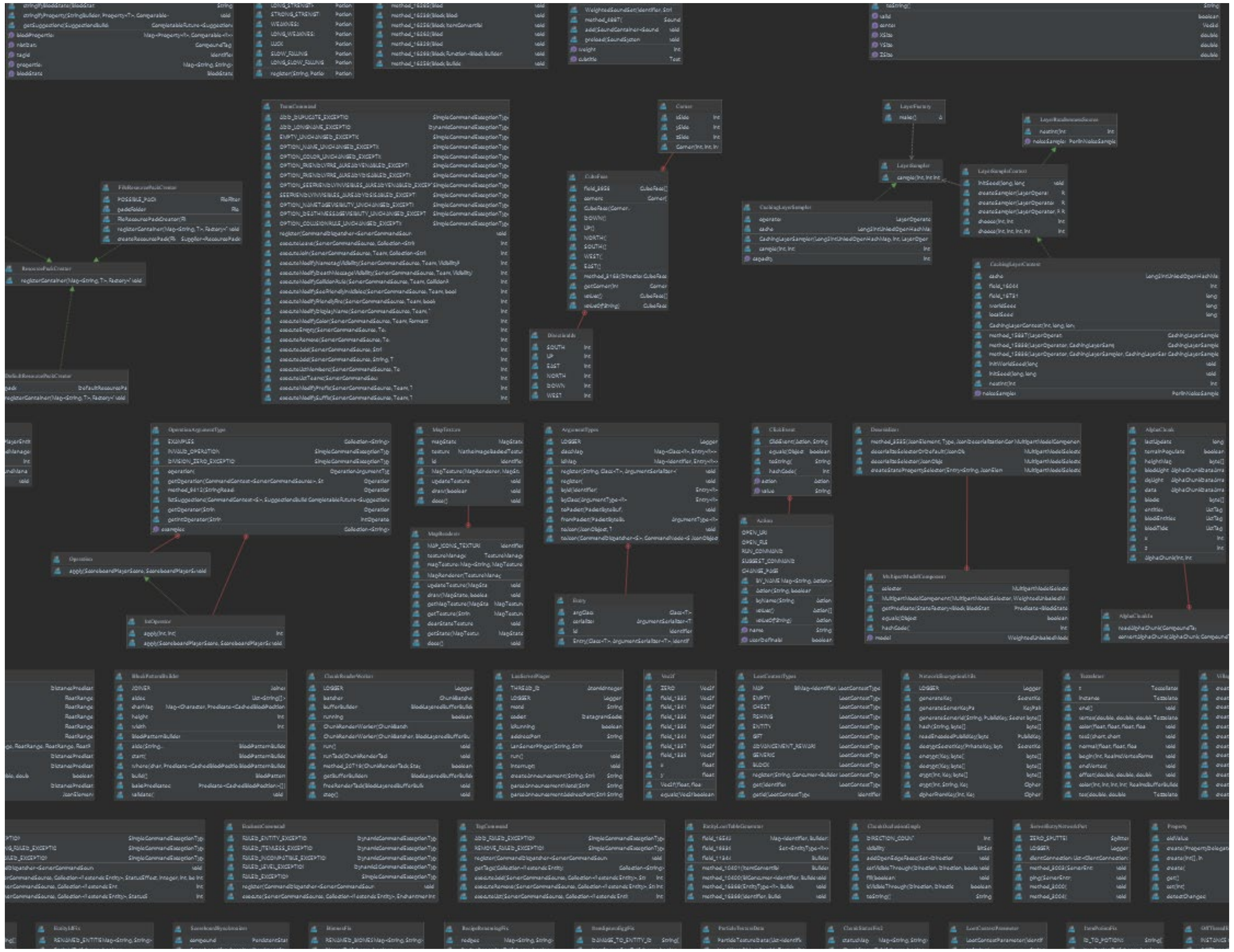
Why Software Engineering?

Software Engineering = "Programming in the Large"?

Software Products by SLOC

- Google: ~2 Billion
- Car software: ~100 Million
- Mac OS X: ~85 Million
- Windows XP: ~45 Million
- PHP: ~693.000
- Moodle: ~396.000





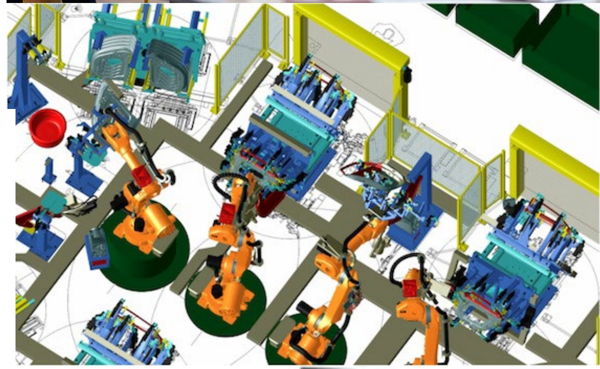
Software

Software

Software stands for one or several computer programs and all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful.
[adapted from Sommerville]

Software-intensive Systems (aka Cyber-Physical Systems)

A system (i.e., mechanics, electronics, software) where software influences to a large extent the design, construction, deployment, and evolution of the system as a whole.



Software Characteristics

Other Engineering Subjects (e.g. Hardware)	Software
• Clearly defined problem. • Tangible: You can touch the material. • Bound to physical laws. • Easy to measure, e.g., 4.5 GHz clock. • Ages: Parts break or become slow. • Hard to adapt / no adaption planned: changing things is difficult and costly. • No or seldom adaption. • Sparse parts available. • Costly to replicate: Each exemplary must be produced again.	• Often unclear or incomplete big picture. • Intangible: You can not touch or see it. • Abstract. • Hard to measure. • Does not age as such. Solutions appear "old" when looked upon 10 years later. • Easy to adapt. • Adaption on regular basis, even multiple times a day. • There are no sparse parts as such. • Easy and cheap to replicate: Just copy it.

Software Cost

Software Cost

Software cost often surpasses system cost by magnitudes.

Software maintenance costs more than its initial construction, usually 80% of the total cost.

Low-quality software damages more than it helps.

Software Engineering aims to provide means to construct high-quality software cost-efficiently.

Software Development Goals

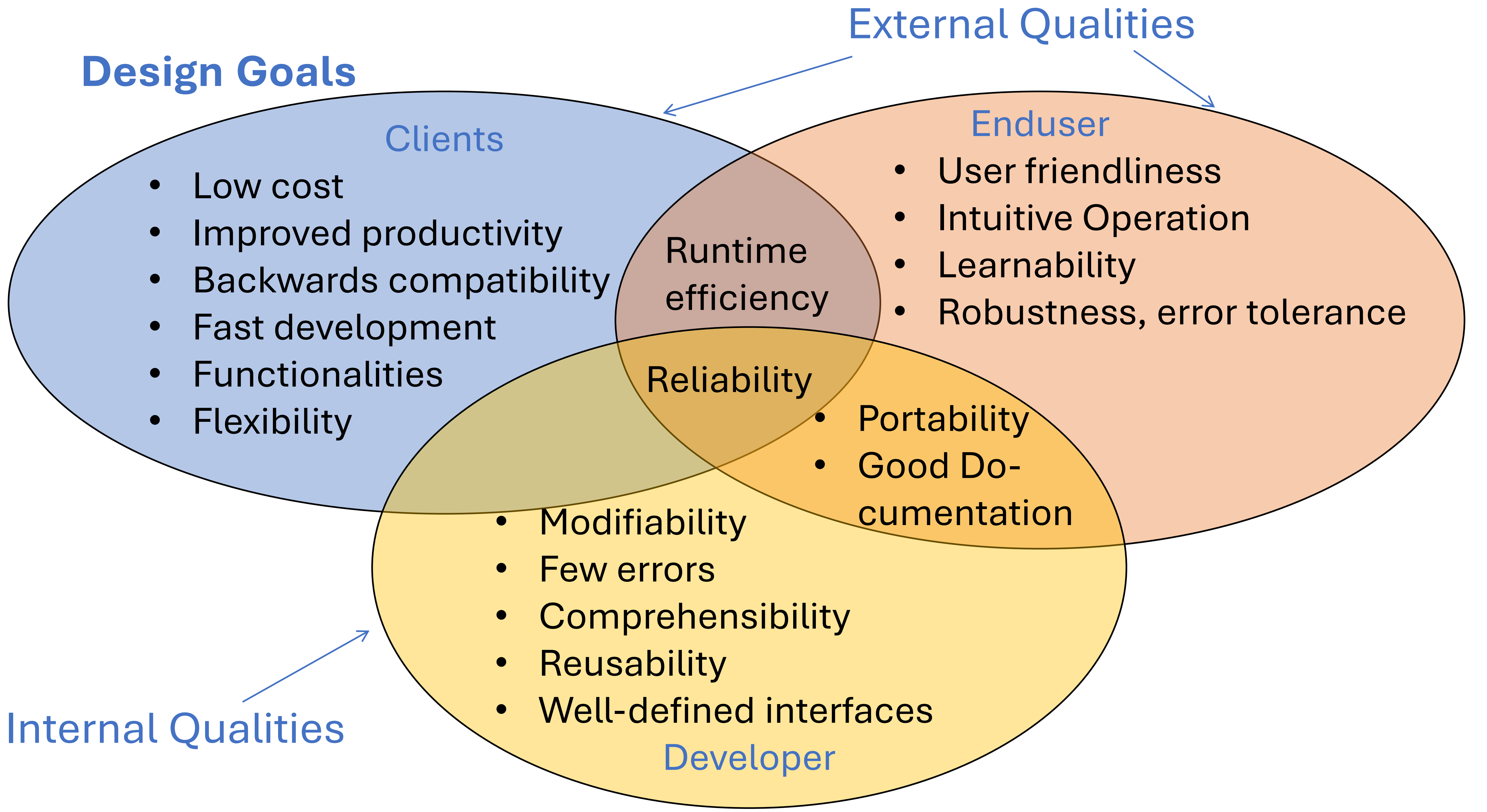
Usually, only three of the following goals are freely adjustable. The fourth one will be determined by the other three.

- Cost
- Quality
- Time
- Functionality Scope

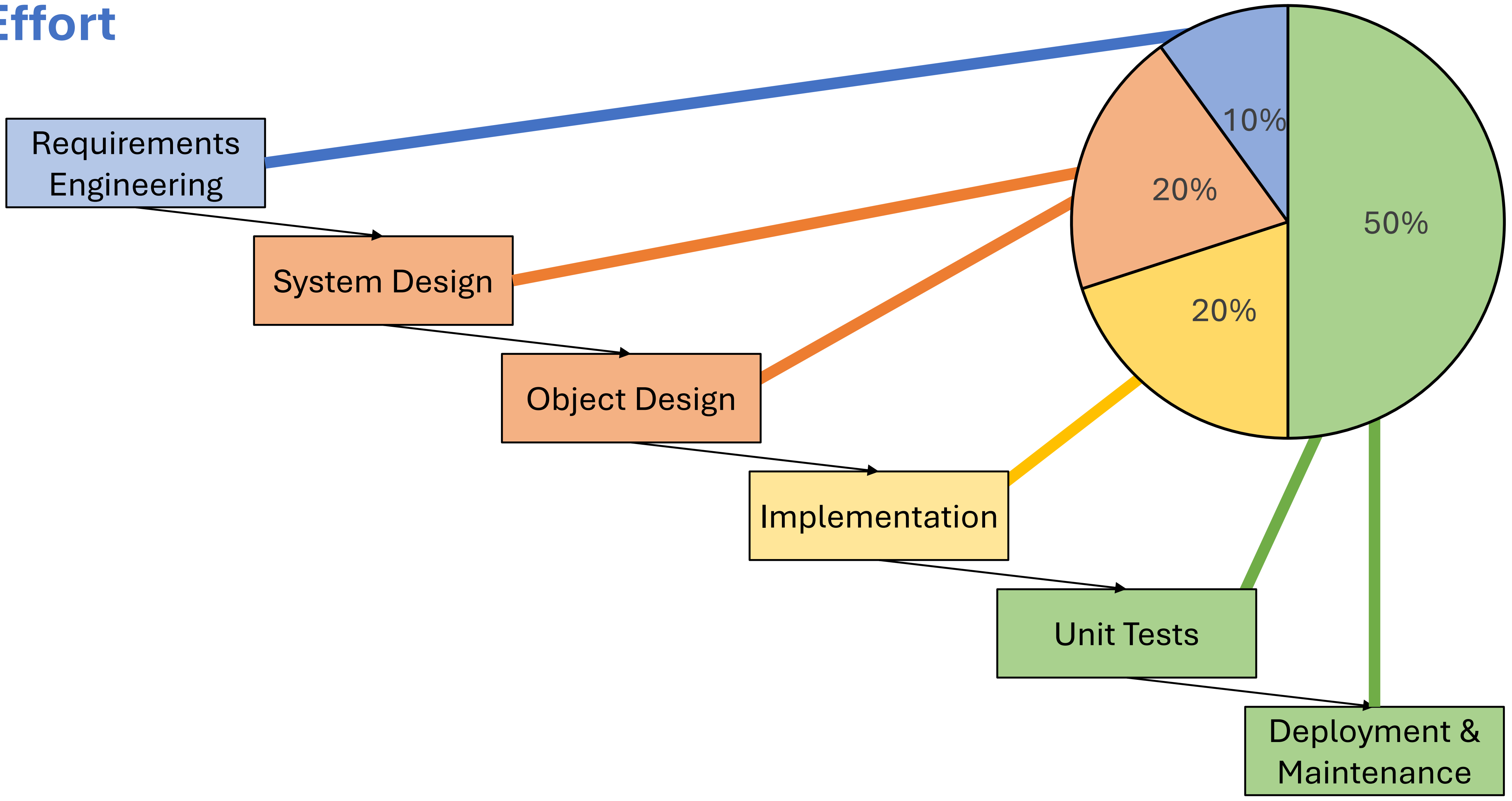
Examples

- High quality, fast, many features → High cost.
- Low cost, many features, high quality → Slow.
- Low cost, fast, many features → Low quality.

Design Goals



Effort


Requirements Engineering
System Design
Object Design
Implementation
Unit Tests
Deployment & Maintenance
10%
20%
20%
50%

(Software) Engineering

Engineering

Engineering is about getting results of the required **quality** within **schedule** and **budget**. [...] Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them **selectively** and always try to **discover solutions** to problems even when there are no applicable theories and methods. Engineers also recognize that they must work within **organizational** and **financial constraints**, and they must look for solutions within these constraints.

[Sommerville]

Software Engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from **initial conception** to **operation** and **maintenance**. [...] Software engineering is not just concerned with the **technical processes** of software development. It also includes activities such as **software project management** and the **development of tools, methods, and theories** to support software development.

[Sommerville]

Object-Oriented Software Engineering

Object-Oriented Software Engineering

Object-Oriented Software Engineering is a variant of Software Engineering that relies on object-oriented principles throughout the software's lifecycle. It often involves object-oriented analysis (OOA), design (OOD) and modeling (OOM).

OOSE in this Lecture

Before the *agile uprising*, OOSE was the de facto standard for commercial software development. While object-oriented programming is still at the core of most commercial software development, extensive in-depth OOA and OOD, however, are not on pace with today's agile software development world.

Thus, we will not approach software engineering as strictly model-based, planned OO-everything but rather OOP and object design (more "on the code" topics) will be our major focus point.

What is your Purpose as a Software Engineer?

You are not employed because you do something or know something or solve some problems.

There are developers in india, russia, china, poland, … that can do the same for significantly less salary.

You are employed to solve problems such that the value of the solution is higher than the cost of the solution – much higher at best.

Key things to learn in software engineering lectures:

1. Understand problems and their severity.
2. Solve problems efficiently.
3. Evaluate cost vs. value of methodologies and techniques.

Next steps

- Join the Discord

- Check course schedule and Ilias course.

- Today (Tuesday): Homework 0 – Technical Setup and Git