Karlsruhe Institute of Technology

# Software Engineering II

**Prof. Dr. Ralf H. Reussner**

## Wrap-Up

DSIS – DEPENDABILITY OF SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

dsis.kastel.kit.edu

# Exam Organization

**First Exam for SS25 and Second Exam for WS2425** (7500207)

- Date and time: 09/10/2025 at 08:00
- Registration period: As stated below
- Registration start: 02/19/2025 at 12:00 AM
- Registration deadline: 09/02/2025 at 11:59 PM
- Deregistration period: As stated below
- Deregistration start: 02/19/2025 at 12:00 AM
- Deregistration deadline: 09/10/2025 at 07:55 AM

**Second Exam for WS2526** (7500054)

- Date and time: 03/10/2026 at 11:00
- Registration period: As stated below
- Registration start: 10/27/2025 at 12:00 AM
- Registration deadline: 03/02/2026 at 11:59 PM
- Deregistration period: As stated below
- Deregistration start: 10/27/2025 at 12:00 AM
- Deregistration deadline: 03/10/2026 at 10:55 AM

If you have any issues with the exam registration, please contact:

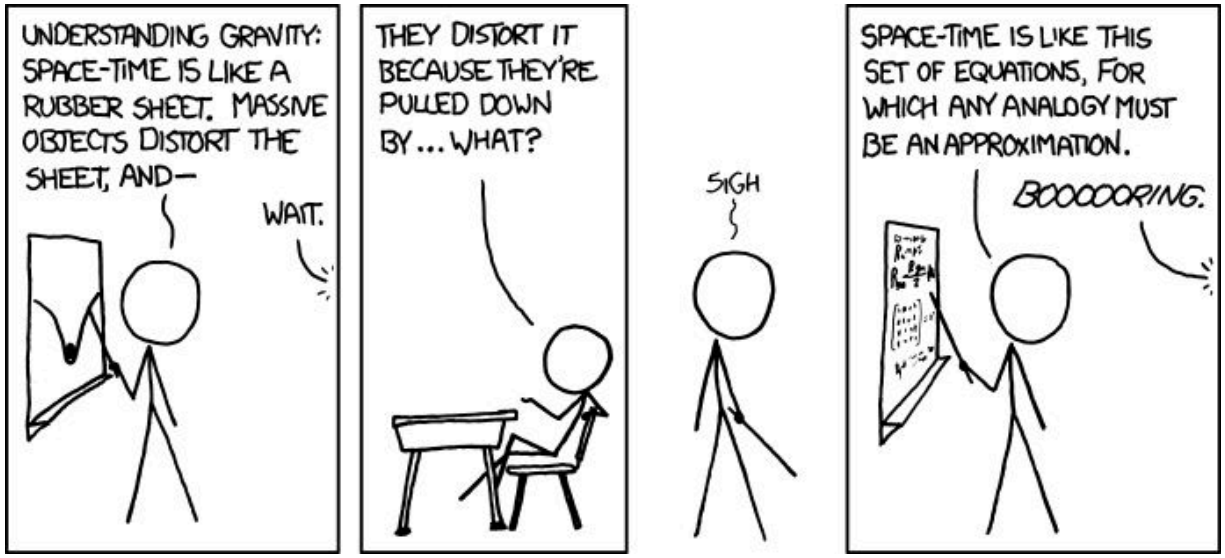
→ *Raziyeh Dehghani (raziyeh.dehghani@kit.edu)*

# Wrapping Up

## Summary of learning goals

- Not exhaustive
- Everything discussed in the lecture and exercise is relevant, except mentioned or marked otherwise

## Additional hints




[Teaching Physics by Randall Munroe https://xkcd.com/895/]

# Course Topics

## Designing High Quality Systems

- Clean Coding
- Software Architecture
  - Clean Architecture
  - Enterprise Application Architecture
  - Components
- Domain-driven design
- Microservices
- Cloud Architecture

## Processes and Requirements

- Development processes
- Agile development
- Requirements engineering
- Use cases

## Ensuring Software Quality

- Software Reliabilty
  - Real-time Systems
  - Real-time design patterns

# DESIGNING HIGH QUALITY SYSTEMS

# Learning Goals

- Clean Code: Be able to –
  - Reiterate best practices for coding (including SOLID principles)
  - Identify violations to the best practices for coding
  - Refactor code wrt. best practices for clean code
  - Apply template method and strategy design pattern

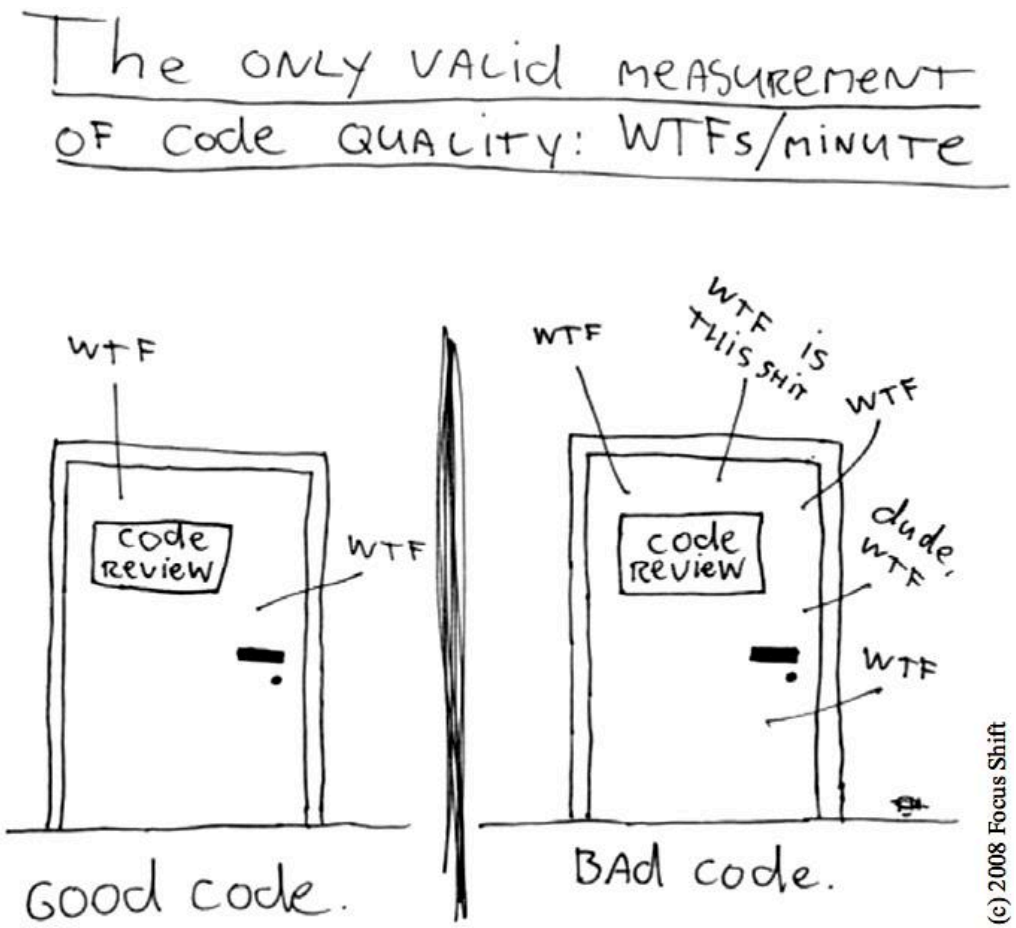



Image source: [Martin2008, p. xxv]

# Learning Goals

- Software Architecture: Be able to –
  - reproduce and describe the definitions of software architecture
  - explain Conway‘s Law
  - explain the difference between software architecture and software architecture documentation
  - describe the advantages of explicit architecture and the influences on architecture decisions
  - assign design decisions and elements to architectural layers
  - explain the architectural style *layered architecture*
  - apply the patterns *Model View Controller* and *Observer* to examples
- Clean Architecture: Be able to –
  - know underlying principles of clean architecture styles
  - know *”circles”* of the clean architecture style and how they are applied
  - apply *dependency inversion* to resolve violations to the clean architecture

# Learning Goals

- Enterprise Application Architecture: Be able to –
  - characterize enterprise systems and decide which characteristics a given application has
  - describe patterns of structuring domain logic, data source architectural patterns, and object-relational structural patterns.
  - select an appropriate pattern for a given design problem and justify its selection with respect to its advantages and disadvantages
  - apply a selected enterprise application pattern for a concrete design
- Software Components: Be able to –
  - understand and define component models
  - Read and “write” component diagrams using the UML notation
  - describe the elements of the Palladio component model (PCM)
  - explain some of the design decisions made in the PCM
  - know the influence factors for the performance of software components

# Learning Goals

- Domain-Driven Design: Be able to –
  - know ideas and patterns underlying Domain-Driven Design
  - Identify bounded context, context map and interaction pattern for a given use case and scenario description
  - distinguish and assign different building blocks to domain model
  - specify additional support patterns to an existing domain model
  - Distinguish different strategic designs
- Microservices: Be able to –
  - explain why Microservice is an architectural style.
  - know the characteristics of Microservices
  - explain the typical elements of a Microservice

# Learning Goals

- Cloud Computing and Cloud Architecture: Be able to –
  - know the differences between cloud computing and hosted resources regarding used technologies, software delivery model
  - explain the economics of cloud computing
  - know and apply the map-reduce programming model
  - know and apply architectural principles of cloud software
  - distinguish SaaS, PaaS, and IaaS and give examples for each

# PROCESSES AND REQUIREMENTS

# Learning Goals

- Software Development Processes: Be able to –
  - recognize and distinguish different types of process models
  - understand the evolutionary and incremental development
  - describe the advantage of incremental over a sequential process
  - describe the phases and disciplines of the Unified Process
- Agile Development: Be able to –
  - know agile development ideas and principles
  - understand how to perform a project following an eXtreme Programming or Scrum process
  - distinguish Unified Process, eXtreme Programming, and Scrum process
  - discuss advantages and disadvantages of agile practices

# Learning Goals

- Requirements Engineering: Be able to –
  - describe the terms and activities of Requirements Engineering
  - classify and assess requirements according to the facets *kind* and *representation*.
  - apply fundamental guidelines on specifying natural language requirements
- Use Cases: Be able to –
  - describe the purpose and the elements of use case diagrams
  - classify use cases according to their level and goal
  - create use case diagrams and *”fully-dressed”* use cases
  - describe their role in the software development process

# ENSURING SOFTWARE QUALITY

# Learning Goals

- Software Quality – Reliability: Be able to –
  - know and distinguish different kinds of software qualities
  - explain the concept of a real-time system
  - distinguish between different types of real-time systems
    - Hard and soft real-time systems, vs. monitoring and control systems
  - select a suitable safety and reliability pattern for a given system

# Course Topics

## Designing High Quality Systems

- Clean Coding
- Software Architecture & Components
  - Clean Architecture
  - Enterprise Application Architecture
  - Microservices
  - Cloud Architecture
- Domain-driven design

## Processes and Requirements

- Development processes
- Agile development
- Requirements engineering
- Use cases

## Ensuring Software Quality

- Continuous Integration
- Software Reliabilty
  - Real-time Systems
  - Real-time design patterns

**Do you want to know more?**

# KASTEL Course Offerings

**Research Groups (Leadership):**

Modeling for Continuous Software Engineering (Prof. Dr. Anne Koziolek)

Self-adaptive Software-Intensive Systems (Prof. Dr. Raffaela Mrandola)

Reliability of Software-Intensive Systems (Prof. Dr. Ralf Reussner)

Testing, Validation, and Analysis of Software-Intensive Systems (Prof. Dr. Ina Schaefer)

# Courses in the Winter Semester 2025/26

Karlsruhe Institute of Technology

**Lecture:**

- Software Architecture and Quality (Prof. Reussner)
- Engineering Self-Adaptive Systems (Prof. Mirandola)
- Software Testing and Quality Management (SQM) (Prof. Schaefer, Dr. Heydari Tabar, Mr. Demmler, Mr. Kodetzki)
- Natural Language Processing and Software Engineering (Dr. Hey, Dr. Keim)
- Model-driven Software Development (Dr. Burger)
- Software Evolution (Dr. Lange)
- Empirical Software Engineering (Dr. Dehghani)
- Introduction to Quantum Computing (Mr. Ammermann)

**Seminar:**

- Data in Software-Intensive Technical Systems – Modeling – Analysis – Protection (Prof. Russner)
- Continuous Software Engineering (Prof. Koziolek)

**Practical Course:**

- Advanced Software Development Tools (Prof. Ina Schaefer)
- Automotive Software Engineering (Prof. Schaefer)
- Engineering Software Development (Dr. Burger)
- Tools for Agile Modeling (Prof. Koziolek)

Test, Validation, Analysis

Software Design and Quality

# Courses in the Summer Semester 2026

**Lecture:**

- **Software Engineering II (Prof. Reussner)**
- **Software Security Engineering (Dr. Gerking)**
- **Ethics of IT (Dr. Heinrich, Dr. Bagattini)**
- **Software Product Line Engineering (Dr. Feichtinger)**

**Seminar:**

- **Software Quality Assurance and Software Testing (Dr. Heydari Tabar)**

**Practical Course:**

- **Current Topics in Quantum Computing (Prof. Schaefer)**
- **Engineering Software Development (Prof. Reussner)**
- **Model-Driven Software Development (Dr. Burger)**
- **Tools for Agile Modeling (Prof. Koziolek)**

**Overview:**

**https://sdq.kastel.kit.edu/wiki/Sommersemester_2026**

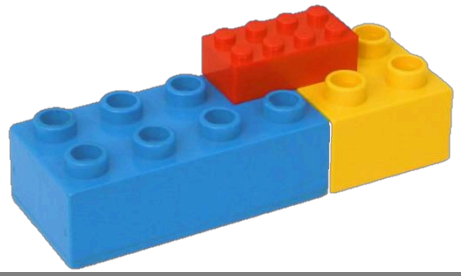
**https://tva.kastel.kit.edu/lehrveranstaltungen/kurse.php**

Test, Validation, Analysis

SDQ Software Design and Quality

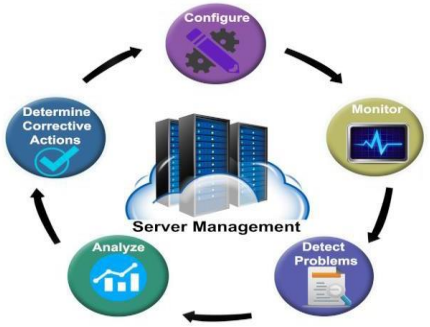
# Software Architecture and Quality (Winter)

- Engineering Software Development with Predictable Quality Characteristics
- Documenting, Evaluating Architectures
- Modeling with UML2
- Component Design, Interface Design, Patterns, Styles
- Middleware Platforms and Component Technologies: EJB, COM, SCA, …
- Reuse of Architectures
  - Architecture Patterns
  - Software Product Lines
- Interoperability of Components



# Course: Engineering Self-Adaptive Systems (Winter)

- Lecturer: Prof. Raffaela Mirandola
- Period: WS 24/25
- Lecture: Thursday, 9:45-11:15, room 131, building 50.34
- **Learning objectives**




- Understand the motivation for self-adaptation
- Understand how to engineer self-adaptive software systems from a software engineering perspective
- Understand the decision-making process using formal analysis at runtime for quality assurance
- Understand the notion of uncertainty in self-adaptive systems and how to tame it with formal verification at runtime
- Understand the level of adoption of self-adaptive systems in the industry

# Software Testing and Quality Management (Winter)

- Focusses on software testing in the software development process
- Explore the different aspects of testing in the life cycle of software
- Current research results in the area of testing of variant-rich systems (SPLs)
- Option to take the *ISTQB Certified Tester Foundation Level* certification at KIT after the regular exam

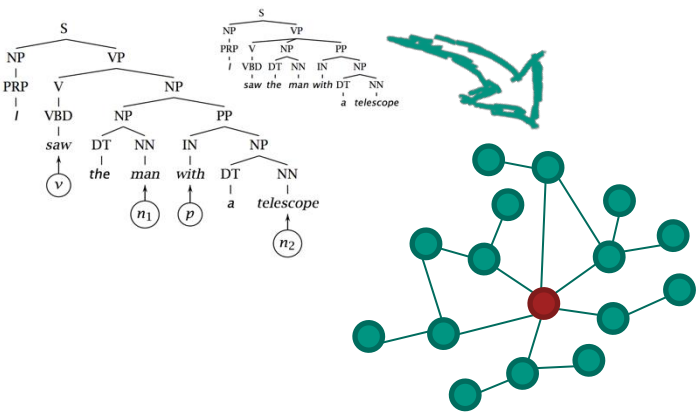
- 2+2 SWS (Lecture + Exercise), given in English
- Written exam (90min)
- 5 LP

# Introduction to Quantum Computing (Winter)

- Focusses on quantum computing foundations and gives an overview of current topics
- Lecture series consisting of:
  - Foundations of quantum computing
    (about half of the lectures, including excercises)
  - Talks by invited experts on current topics, e.g.,
    - Quantum Hardware
    - Quantum Programming using Qiskit
    - Quantum Machine Learning
- 2 SWS (Lecture)
- Written exam (90min)
- 3 LP

# Natural Language Processing and Software Engineering (Winter)

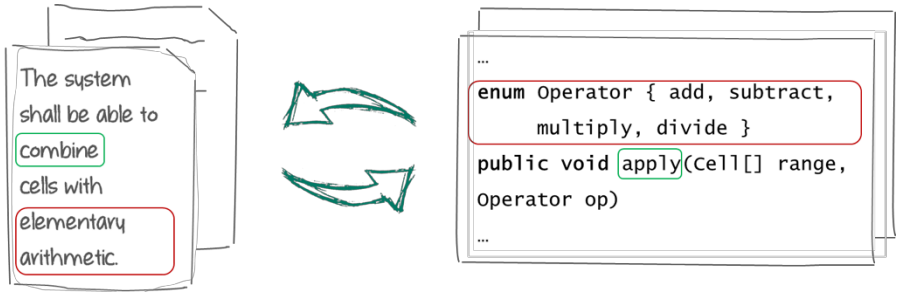
- Applications of computational linguistics and artificial intelligence to support and improve the software development process, e.g. through
  - Automatic model generation
  - Requirements classification
  - Traceability analysis
  - ...




**Contact:** Dr.-Ing. Tobias Hey, Dr.-Ing. Jan Keim

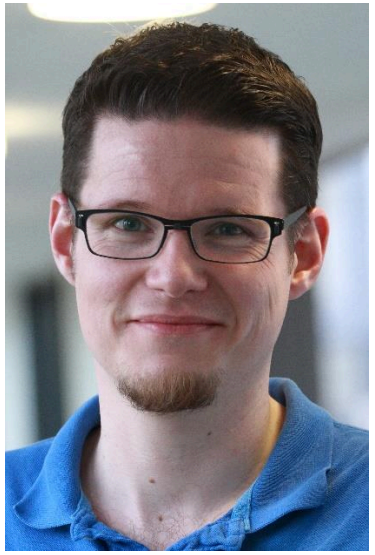
**LP:** 3

Thursday 14:00-15:30, R -118



# Model-Driven Software Development (Winter)

- LP (ECTS): 3
- SWS: 2
- Lecturer: Dr.-Ing. Erik Burger



Location and Time of the Course

- Friday, 09:45–11:15
- Building 50.34, Room –102

`https://sdq.kastel.kit.edu/wiki/MDSD`

# Software Evolution (Winter)

- Course Content:
  - Software Development Processes
  - Peculiarities of Long-Lasting Software Systems
  - Evolution Scenarios for Software Systems
  - Software Architecture Development
  - Software Renovation
  - Implementation Techniques
  - Architecture Patterns
  - Traceability
  - Software Evaluation Methods
  - Durability Analysis and Tools (to support software evolution)

# Empirical Software Engineering (Winter)

- **Hands-On Research Experience**: Design and Carry out Empirical Studies
- **Practical Statistical Skills**: Use Statistical Methods for Data Analysis
- **Critical Thinking Development**: Assess and Interpret the Results of Empirical Studies
- **Design Your Own Experiments**: Get Practical Experience in Planning and Conducting Engineering Experiments
- **Collaborative Learning**: Participation in Group Projects and Discussions to Refine Understandings

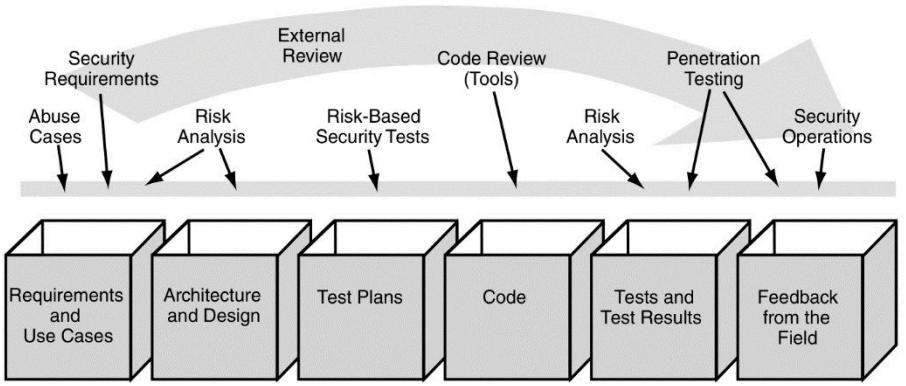
# Practical Labs at TVA (Winter)

- Students learn to use and extend software from research and practice
- Adding new functionality to tools and improving non-functional requirements
- Close mentoring (individual or in groups)
- High practical effort

- Topics & registration via Wiwi-Portal (see website)
- 4 SWS
- 6 LP

# Software Security Engineering (Summer)

- **Learning Objective:** Engineering-based assurance of cybersecurity throughout the software development lifecycle.
- **Course Content**
  - Constructive and analytical measures to achieve security objectives through prevention or detection of vulnerabilities
  - Implementation and execution of security measures in various development phases
  - Relevant fundamentals of formal security models
- **Advanced Subjects**
  - Software Engineering and Compiler Construction
  - Cryptography and Security
- **ECTS:** 3
- **Contact:** Dr. Christopher Gerking, christopher.gerking@kit.edu



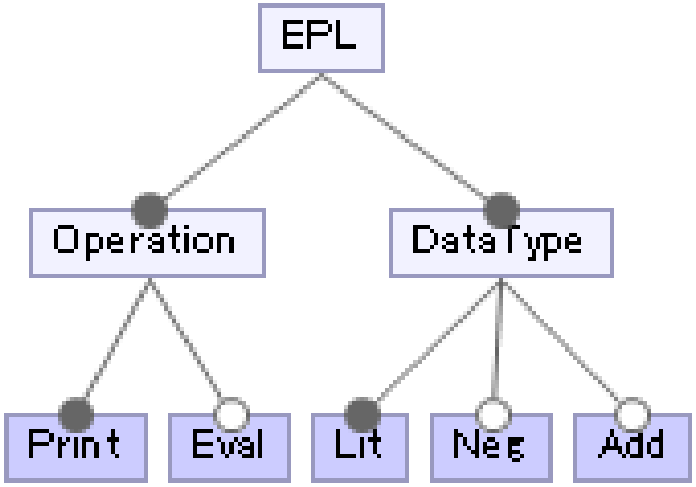
# Ethics of IT (Summer)

- **Emerging IT Technologies**: Self-driving cars, care robots, recruitment software, and complex medical diagnostic tools offer both benefits and risks.
- **Everyday Impacts**: Phishing emails, spam, and the mental health effects of social media, like loneliness, affect almost everyone today.
- **Global Consequences**: Technologies like surveillance systems, facial recognition, and big data can undermine political systems, as seen in the Cambridge Analytica scandal.
- **Ethical Responsibility**: While responsibility is shared across society, computer scientists are often the first to face ethical challenges.
- **Navigating Ethical Gray Areas**: The course helps students confidently address ethical dilemmas where clear right or wrong answers may not exist.
- **Open, Expert-led Discussions**: The lecture fosters open discourse on fundamental and applied ethical issues in IT, welcoming students from all faculties.

# Software Product Line Engineering (Summer)

- Development of variant-rich software-intensive Systems
- Product line scoping
- Variability implementation and management
- Variability model analysis
- Product line testing
- Current research advances

- Written exam
- 3 LP




```
class Op {
  #ifdef Eval
  int res;
  #endif
  #ifdef Neg
  public neg() {
  #endif
  …
  #ifdef Add
  }
  #endif
}
```

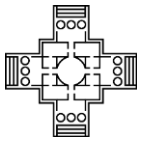
# Seminar: Software Quality Assurance and Software Testing (Summer)

- Exploring current research in the field of software quality assurance
- Scientific analysis and comparison of at least two approaches to a specific topic
- Preliminary pool of topics for SS25: Software testing in general, software testing using AI, software verification using AI

- 2 SWS, given in English
- Main tasks: writing a scientific paper (6-8p), giving a presentation about the topic, and reviewing papers of two fellow students
- 4 LP
- Registration via ILIAS in April (see website for updates)

# Practical Course in Engineering-Based Software Development (Summer)

- Extending the Eclipse development environment or platform and related tools with plug-ins (additional topics depending on the semester, e.g., MDSD, .NET, etc.)
- Research-oriented and practical at the same time.
- Focus Areas
  - Palladio Software Architecture Simulator
  - Vitruvius for consistency-preserving, view-based software development
- Individual support in a pleasant atmosphere
  - Generally, one supervisor per student
  - Regular meetings
  - Coffee machine, pool room
  - Creativity and initiative are encouraged

VITRUVIUS

# Practical Course in Model-Driven Software Development (Summer)

- In this practical course, you can expect:
  - Software development using current standards and tools (EMF, QVT, ATL, Xtext, …)
  - Graphical editors (Sirius, Graphiti)
  - Development of domain-specific languages (DSL)
  - Collaboration on the current Vitruvius research project




**We recommend this practical course for students who have already attended the Model-Driven Software Development lecture.**

# Practical Course: Tools for Agile Modeling (Summer)

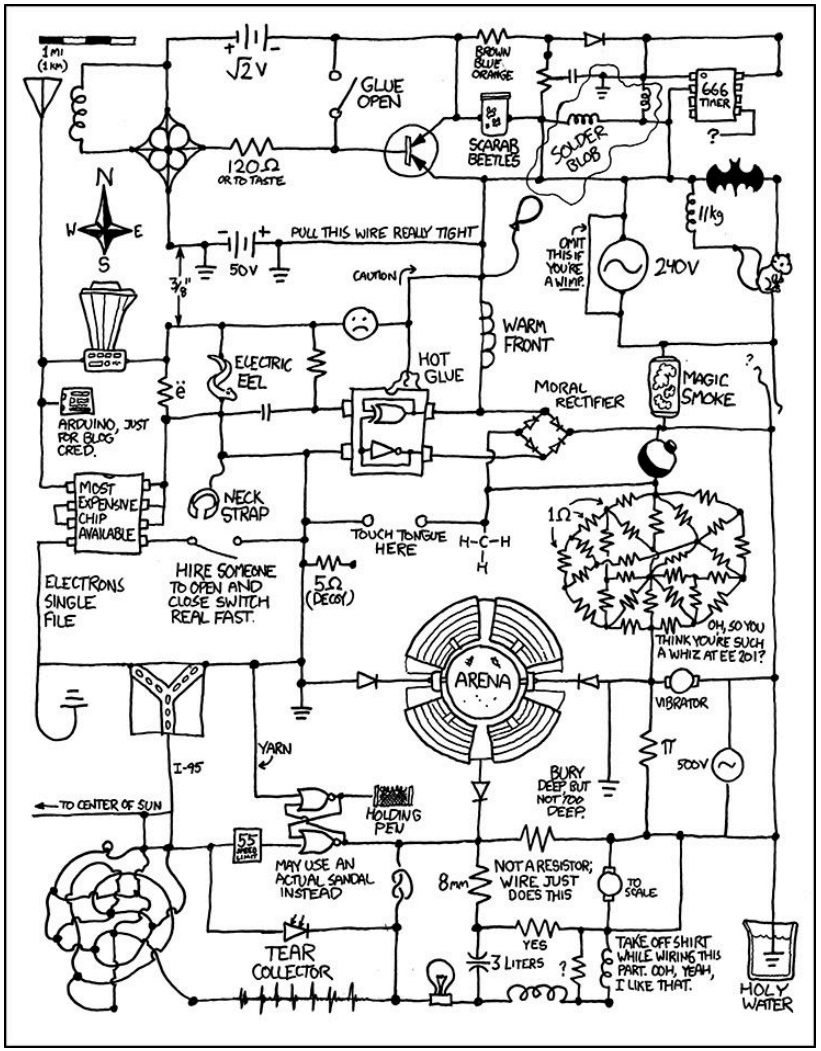
- In this practical course, participants will use and extend current tools from both industry and research to support agile modeling.
- The following types of tools will be covered:
  - Tools for analyzing natural language and/or model sketches to derive semi-formal models and statements about software systems
  - Tools for partially automating the derivation of models from source code and runtime information, particularly performance data
  - Tools for simplified, domain-specific modeling
- Depending on the topic being worked on, different technologies will be used, including:
  - Eclipse platform, EMF (Eclipse Modeling Framework), and additional Eclipse plugins
  - Palladio simulator
  - Protégé ontology editor
  - Natural Language Processing Toolkit (Stanford CoreNLP)

# ADDITIONAL HINTS

# How much Theory, how much Practice?

- Be able to redo what was done in the exercise
  - For another task, of course
- Know and understand what has been presented in the lectures
  - Most importantly content of slides
  - “Understanding” means being able to explain the rationale of it. Hence, not just knowing by heart. Ask yourself: “why is it like this?”
  - Referenced books help in understanding such backgrounds, but also recorded lecturer
  - conclude new insights through transfer of knowledge
- Be able to apply as stated in learning goals
- Be able to read and draw some types of diagrams *(next slide)*

# Which Diagrams Should I Know?




- Use Case diagrams
- Class diagrams
- Sequence Diagrams
- Deployment Diagrams
- Activity Diagrams
- Component diagrams

- Know the basic constructs and notation used in the slides (UML 2.5 Notation)

[Circuite Diagram by Randall Munroe https://xkcd.com/730/]

# More Useful Hints

- Take obtainable amount of points per task into account
  - 2 point task does not require a two page explanation
- Skip tasks, which you cannot immediately solve
  - Don‘t get wound up in a difficult task, before you have at least read and attempted all task
- Read the task description **very** thoroughly
  - If, for instance, the Task asks for *“Verstöße gegen Richtlinien des Formulierens natürlichsprachlicher Anforderungen”*
  - Do not name other issues with the requirements you might have found
- If you want to fail the exam, strike out every attempted tasks before handing in your exam, with a pen and a ruler
  - To pass the exam you must receive around 50% percent of the points

# Language of the Exam?

- The tasks will be provided in English and German.
- You can use dictionary during the exam. Please write teaching assistants about this, and deliver the dictionary at least one week before the exam.
- Answers can be written in German or English:
  - English is recommended and preferred.
  - Please write consistent within one task and avoid mixing German-English sentences.
  - English specific terms can also be used in German answers.

# EVENTS IN THE NEXT SEMESTER

# Assistant Positions

- Tutors for the lecture "Programming" (every semester)

  Programming/Exercise Guide

- Further development and maintenance of the Traceability Link Recovery approach ArDoCo (incl. CI/CD)

  Jan Keim, Dominik Fuchß, Sophie Corallo

- Preparation and documentation of project knowledge for teaching

  Robert Heinrich, Nicolas Boltz

- Support in the CI/CD build process of large research projects

  Larissa Schmid, Nicolas Boltz

# INFORMATION SCIENCE AND RESPONSIBILITY

# Information Science and Responsibility – Current Ethical Challenges in IT

- New technologies and applications in the field of IT (e.g., self-driving cars, caregiver robots, apps, software for recruitment processes or for use in complex medical diagnostic procedures, modern surveillance systems, facial recognition technologies, and big data algorithms in socio-global contexts) bring both advantages and risks.
- This ambivalence of new technologies leads to the question of responsibility for the consequences associated with them.
- Computer scientists are often the ones who are first confronted with ethical problems.

# The „two“ things you should know on SE


- Professor Dr. Harro Heuser (Analysis I, 1993)
  - „if you know these two thing from this lecture, I will be happy.“
    1. Convergence formular of geometric series
    2. Cauchy convergence criterium
- So what are the „two“ things for SE II?
  - actually more than two…
- Number ONE: un-documented dependencies are THE poison for software (projects)
  - make it as easy for you and for others to change something ***consistently***.
  - Traceability for free is the best you can have in you documents (incl. code)
    - use use cases
      - 1:1 mapped to sequence diagrams
      - 1:1 mapped to sets of acceptance test cases
  - A well managed (kept consistent) software architecture documentation is the central document of all (it can link all other artifacts)

# The „two“ things you should know on SE

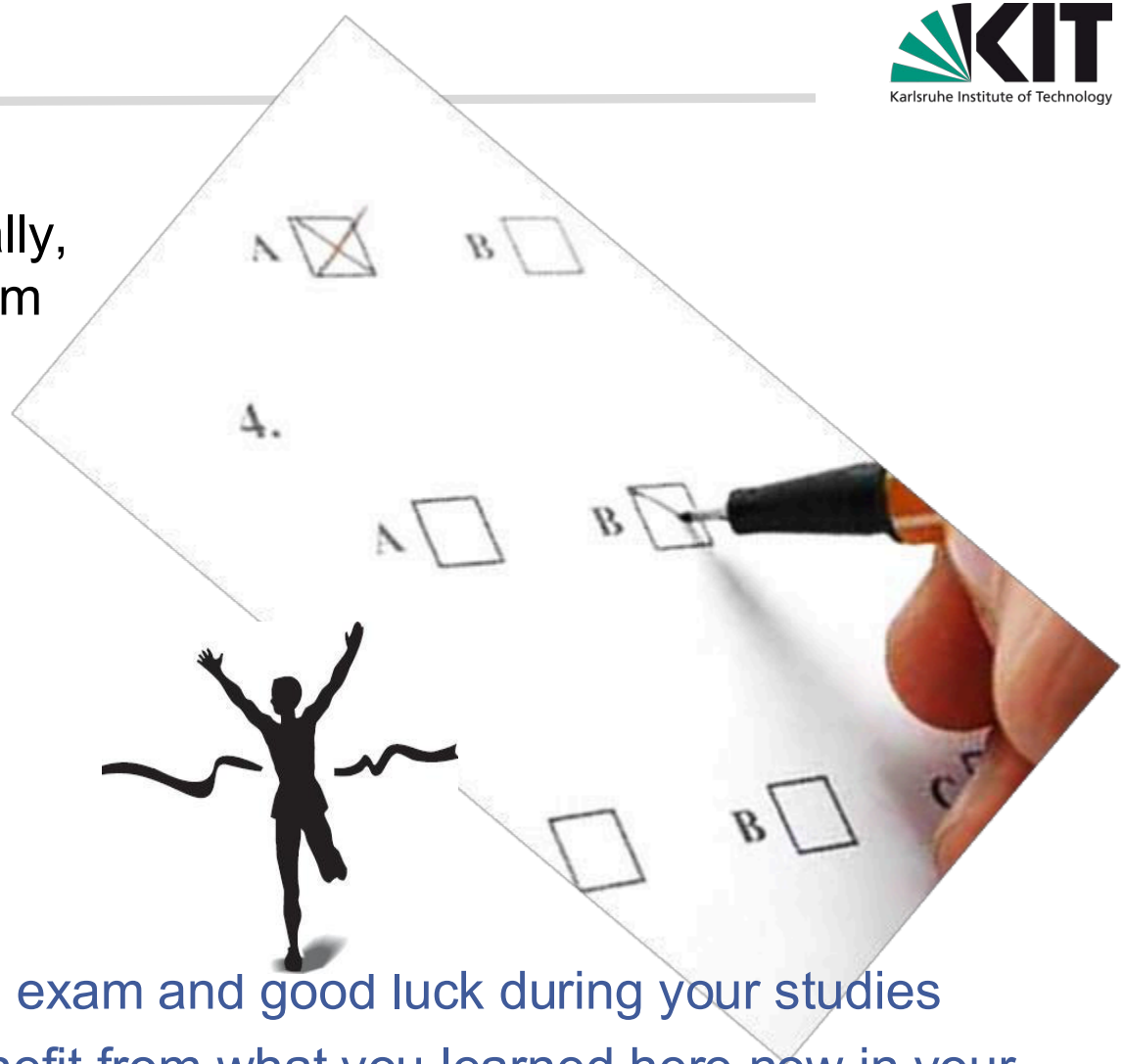
- Number TWO: **All working complex systems come from a working simple system!** (Herbert Simon, „The Sciences of the Artificial“)
  - enable fast feedback cycles
    - Plan ahead to reduce waste (unnecessary refactorings, make central architectural decisions). If planning creates more waste, stop planning.
    - But be aware of changes of requirements, teams, technolgy, during the course of your project. These changes must be reflected for a successful project and a product being usabe when it is delivered!
    - **use executable, deployable increments, usable by customer**
    - have a serious „definition of done“ (do not kid yourself (nicht in die eigene Tasche lügen)) → Test-driven development, monitored high unit case coverage, rigorous unit tests, reviews! (reviews!) ((Did I mention reviews?→ Reviews!!!)), automated integration and system tests
  - try to improve yourself, your process. (self-reflection, restrospectives)
  - move the the employer supporting this!

# The „two“ things you should know on SE

- Number THREE: Not for all societal „problems“ a technical (software) solution can help, but for all technical solutions reflect ethically whether it supports the values of our society. (If you do not know them, have a look in our constitution.)
- Consider legal regulations (if they exist for your domain).
- Be aware, that in any case software acts as an institution:
  - it creates „laws“
  - guides people what they consider allowed and possible.
- Software shapes our living together and our society.
- Hence, if no legal regulations apply, reflect ethically on potential effects (their likelyhood) and whether it complies with our values.
- Move to the employer fitting to this understanding.

# Feedback?

- Give feedback personally, via e-mail or ILIAS forum
- We are happy to get your feedback




→ I wish you a successful exam and good luck during your studies

→ I hope that you can benefit from what you learned here now in your professional life later!