Technology
Arts Sciences
TH Köln

Course
SEA - Software engineering for automation technology

Version: undefined | Last Change: - | Draft: undefined | Status: undefined

General information

Long name	Software engineering for automation technology
Approving CModule	SE_BaET
Responsible	Prof. Dr. Stefan Kreiser Professor Fakultät IME
Remarks	Lecture / Exercise weekly (Flipped Classroom), Project work
Level	Bachelor
Semester in the year	summer semester
Duration	Semester
Hours in self-study	90
ECTS	5
Professors	Prof. Dr. Stefan Kreiser Professor Fakultät IME
Requirements	- basic knowledge of behavioral modeling (e.g. PAP, automata, state charts, Petri Nets) - basic programming knowledge in C/C++ - basic knowledge of object orientation (classes, objects)
Language	German and English
Separate final exam	Yes

Final exam

Details

Natural language description of a realistic automation system (textbook):
Modeling in UML, sound implementation in C++.

20min written preparation,

20min oral questioning on the prepared solution with possibility for optimization.

Minimum standard

Using a natural language description of a realistic automation system of appropriate complexity, students model the system model of a software system suitable for solving the given automation task and justify and evaluate the essential properties of their design. To justify and evaluate, students refer to the specific requirements of the automation system as well as to fundamental quality criteria for automation software systems (system, development, operation, service, and maintenance requirements), demonstrating on selected model artifacts in particular that and how the system model can be transformed into a software model and subsequently into an implementation model, and what consequences their design has for the models of the subsequent design phases.

Exam Type

Natural language description of a realistic automation system (textbook):

Modeling in UML, sound implementation in C++.

20min written preparation,

20min oral questioning on the prepared solution with possibility for optimization.

Lecture / Exercises

Learning goals

Knowledge

Terms

- software system, software product
- software quality
- software Complexity

Object-oriented modeling with UML

- domain model (structure, behavior, system boundaries / interfaces)
- software architecture model
- implementation model
- model transformations
- modeling tools

Process models

- linear (phase model, V-model)
- evolutionary (eXtreme Programming, Scrum, Timebox)

Quality management (SOPs)

Requirements analysis

- requirements engineering
- design-input-requirements (requirement specification)
- Laws, standards and organizational requirements

Product risk analysis (FMEA, FTA)

Design

- design principles
- feasibility studies
- system specification (functional specification)
- software specifications

Implementation

- choice of programming languages, programming guidelines
- development in distributed teams, developer test

- system integration
- commissioning

Verification & Validation
- formalized software testing (dynamic, static)
- field evaluation
- operational support

Management tasks
- document management
- configuration management (version management, build management)
- test management
- change management

Skills

Analyze technical software systems
- methodically elicit, consolidate and prioritize system requirements
- design formalized requirements specification

Model technical software systems
- use Unified Modeling Language notations to model simple software systems
- use notations for structural modeling (class diagram, package diagram, component diagram, distribution diagram)
- use notations for behavioral and interface modeling (use-case diagram, activity diagram and action concept state machine and protocol machine, sequence diagram)

Name and delimit modeling levels
- system model (customer view): Entity model, interface model, behavior model
- software model (developer view): Technical class models, detailed behavioral models, design principles, basic software architectures.

Derive context, boundaries, tasks, behavior, and structures of simple software systems from texts
- comprehend technical text sections completely
- recognize and understand implicit statements
- recognize and resolve inconsistencies
- recognize and derive missing information or ask for it

Model software systems with UML2 notations
- design simple system models iteratively (derive entity model, context and use-case model from customer's point of view, detail use cases, describe standard scenario and essential alternative scenarios and refine as activity diagram)
- design simple software models iteratively (refactor and detail entity model from developer's perspective, detail behavioral models from developer's perspective, model structure-based behavior as state chart, refine activities to action level, establish relationship between actions and class methods)

Operate professional UML2 design tool

Verify models
- define evaluation criteria
- adhere to modeling guidelines and design principles
- evaluate completeness or unnecessary complexity
- evaluate quality with respect to specific customer specifications (define test cases, perform and document model reviews,
detect and name model errors, correct and optimize models based on assessments)

Design technical software systems
- identify product risks, define mitigation measures and consider them in the design
- name, explain and apply design principles to achieve defined quality goals
- select and apply problem-oriented system and software architecture
- explain and exemplarily apply methods for software development in distributed teams
- explain methods for software testing in distributed teams and apply them exemplarily

Develop technical software systems quality-controlled
- apply process models by way of example
- Obtain information from international standards for software development (German/English)

Expenditure classroom teaching

Type	Attendance (h/Wk.)
Lecture	1
Exercises (whole course)	2

Separate exam

none

Project

Learning goals

Skills

Analyze larger technical software systems
- comprehend and understand extensive technical texts, especially English-language texts
- evaluate and arrange extensive system requirements
Model larger technical software systems
- delimit modeling levels: system model (customer view), software model (developer view)
- use model notations systematically to describe systems
- iteratively derive interface, behavior and structure models in UML2 notations
- use professional UML2 design tools purposefully
- verify and evaluate models, correct model errors and optimize models
Design larger technical software systems
- select and apply design principles to achieve defined quality goals
- select and apply problem-oriented system and software architecture
- perform software development and software testing in distributed teams
Create and review source code
- analyze given source code and extend it purposefully
- use object-oriented programming language (C++)
Develop larger technical software systems in a quality-controlled manner
- apply evolutionary procedure model
- gain information from international standards for software development (German/English)
Present the team's work results in English in a compact and target group-oriented manner

Demonstrate action competencies:
Model real-world systems
- Decomposition (recognize or define system boundaries and use them correctly, recognize or define system interfaces and use them correctly, recognize or define system structures and represent them correctly, recognize or define system functions and represent them correctly)
- Composition (creating structural and behavioral models, integrating models, verifying and evaluating partial models and overall models)
- master complex tasks in a team based on division of labor (plan and control simple projects, comply with agreements and deadlines, plan and conduct reviews)
- apply model transformations (revert model elements from given C++ source code, complete and verify models by manual source code analysis,

model system extensions and solution modifications based on a current specification, generate source code from new model and complete generated source code manually, verify implementation in the debugger and by systematic tests on the target system)

Expenditure classroom teaching

Type	Attendance (h/Wk.)
Project	1

Separate exam

Exam Type

working on projects assignment with your team e.g. in a lab)

Details

Part 1 (round-trip engineering): perform comprehensible transformations between system model, software model, implementation model and source code based on the requirements and framework conditions.

Part 2 (Project Task): Based on a natural language description (English) of a realistic automation system of appropriate complexity, students model the system model of a software system suitable for solving the automation task and justify and evaluate the key features of their design. To justify and evaluate, students refer to the specific requirements of the automation system as well as to basic quality criteria for automation software systems (system, development, operation, service, and maintenance requirements), demonstrating on selected model artifacts in particular that and how the system model can be transformed into a software model and subsequently into an implementation model, and what consequences their design has for the models of the subsequent design phases.

Minimum standard

Part 1: Executable software system that can be shown to have the required properties and to be consistent with the UML model.

Part 2: System model that demonstrably fulfills the essential requirements from the DIRs, software architecture concept that is justifiably suitable for implementing the system model taking into account all DIRs.