Technology
Arts Sciences
TH Köln

Course Manual SEKM

Software Engineering by Components and Pattern

Version: 1 | Last Change: 06.10.2019 18:52 | Draft: 0 | Status: vom verantwortlichen Dozent freigegeben

− General information

Long name	Software Engineering by Components and Pattern
Approving CModule	QEKS_MaET, QEKS_MaTIN
Responsible	Prof. Dr. Stefan Kreiser Professor Fakultät IME
Valid from	winter semester 2020/21
Level	Master
Semester in the year	winter semester
Duration	Semester
Hours in self-study	78
ECTS	5
Professors	Prof. Dr. Stefan Kreiser Professor Fakultät IME

Literature

D. Schmidt et.al.: Pattern-Oriented Software Architecture. Patterns for Concurrent and Networked Objects (Wiley)

Gamma et.al.: Design Patterns, (Addison-Wesley)

Martin Fowler: Refactoring, Engl. ed. (Addison-Wesley Professional)

U. Hammerschall: Verteilte Systeme und Anwendungen (Pearson Studium)

Andreas Andresen: Komponentenbasierte Softwareentwicklung m. MDA, UML2, XML (Hanser Verlag)

T. Ritter et. al.: CORBA Komponenten. Effektives Software-Design u. Progr. (Springer)

Bernd Oestereich: Analyse und Design mit UML 2.5 (Oldenbourg)

OMG Unified Modeling Language Spec., www.omg.org/um

I. Sommerville: Software Engineering (Addison-Wesley / Pearson Studium)

K. Beck: eXtreme Programming (Addison-Wesley Professional)

Ken Schwaber: Agiles Projektmanagement mit Scrum (Microsoft Press)

Final exam

Requirements	- programming skills in an object-oriented programming language, preferably C++ - knowledge of software modeling using Unified Modeling Language (UML) or other (formal) languages that support modeling of interfaces, behavior and structures - basic knowledge in (agile) project management, SCRUM oder XP - basic knowledge of essential softare architectural models - basic knowledge of interconnection models in software systems (OSI, TCPIP, Messaging)
Language	German and English
Separate final exam	Yes

Details	Oral examination after written preparation. Based on a realistic task of appropriate complexity, the students develop and model a suitable software architecture for a distributed automation system using strategies for the reuse of model and/or software artifacts. They justify the essential structures of their architecture with reference to the specific objectives and the specific environmental conditions for the use of the respective automation system as well as with reference to basic quality criteria for automation software systems (system, development, operation, service and maintenance requirements). They explain which special organizational conditions with respect to the development result from their software architecture and evaluate the quality of the architecture from a technical and business point of view.

Minimum standard	- Students extract the essential relevant information, basic conditions and solution limitations from the task specification and design a model of the software architecture considering basic quality criteria for automation software systems. To do this, they select a sound strategy for the reuse of model and/or software artifacts and justify their approach. - Students explain the essential structures of their software architecture with regard to the adaptation of the reused models and artifacts and justify them with regard to the given system requirements, i.e. technical system requirements as well as further, possibly relevant, development, operation, service and maintenance requirements.
Exam Type	EN mündliche Prüfung, strukturierte Befragung

Lecture / Exercises

Learning goals

Goal type	Description
Knowledge	Terminology value vs. cost of a technical software distributed software system, concurrency software quality, quality of service, refactoring complexity (algorithmic, structural), emergence re-use, symmetry and symmetry operations, abstraction, invariants quality controlled re-use, methodical approaches variants of white box re-use black box re-use grey box re-use (hierarchical approach to re-use) re-use in automation control software systems determinism benefits and challenges tailoring process models and personnel structures in projects meet requirements in development projects predictably (product quality, cost, deadlines) distributed development, maintenance and support of software systems software pattern pattern description using UML essential architectural pattern construction pattern structural pattern behavioural pattern class based (static) vs. object based (dynamic) pattern essential pattern for concurrent and networked real time systems encapsulation and role based extension of layered architectures concurrency structures to optimize throughput and system response latency distributed event processing process synchronisation construction and use of pattern catalogues, pattern languages pattern based design of complex software systems components and frameworks design principles interface architectur

Special requirements

none

Accompanying material	lecture slides, digital exercise collection, digital professional development tool for Unified Modeling Language (UML2) professional integrated software development tool for C++
Separate exam	No

	active and passive system elements design, programming and test quality configuration and use using middleware systems to develop architectures of technical software systems ORB architectures, e.g. CORBA and TAO integrated system plattforms, e.g. MS .NET multi agent systems (MAS) agent architectural models collaboration between agents agent languages considering cases for MAS application
Skills	use pattern to design complex software systems extract and discuss purpose, limitation of use, invariant and configurable parts of pattern from english and german literature sources understand implementation skeletons of pattern and map them to problem settings with limited technical focus discuss benefits of using object oriented programming languages derive recurrent settings in the development of complex software systems implement pattern on exemplary settings and test resulting implementations reasonably combine pattern to solve recurring problem settings with a broader technical focus use UML2 notations use professional UML2 IDE for round-trip-engineering integrate software system based on exemplary implementations of the pattern to combine conduct integration test, assess software quality and optimize software system construct black-box-components based on pattern analyse component based software architectures derive suitable scope from architectural specs understand and discuss development process to construct software systems find active and passive system elements and derive system run time behaviour understand abstract system interfaces to interconnect,

configure and activate components
understand abstract system interfaces to exchange applicational run time data
understand system extension points (functional and structural system configuration layer)
analyse distribution architectures
understand basic system services (describe and reason service usage, relate to system tasks)
relate pattern to structure making architectural software artefacts
derive suitable range of appications for a given distribution architecture
understand engineering process to construct user applications (application layer)
discuss attributes and limitation of usage of interconnection protocols
find designated system extension points
compare MAS to conventional distribution architectures
agent vs. component
architectural models
activation of agents
deployment of agents
protocols for interconnection and collaboration
range of appications and and limitation of usage

Expenditure classroom teaching

Type	Attendance (h/Wk.)
Lecture	1
Exercises (whole course)	1
Exercises (shared course)	0
Tutorial (voluntary)	0

– Lecture / Exercises

Learning goals

Goal type	Description
Knowledge	challenging seminar topics can be defined e.g. from the following or related subject areas - reusable artifacts for building the architecture of distributed software systems, - professional distribution architectures, - Multiagent systems, - special economic, liability and ethical requirements for software systems with (distributed) artificial intelligence and their effects on the design of software architectures
Skills	present personal work results and work results of the team in a compact and target-group-oriented way, both orally and in writing

Expenditure classroom teaching

Type	Attendance (h/Wk.)
Seminar	1
Tutorial (voluntary)	0

Special literature

selbst recherchierte Fachlitertur

Special requirements

none

Accompanying material	Collection of general and, if applicable, topic-related questions with respect to automation engineering and software engineering according to which the quality of software architectures for distributed automation systems is to be investigated and evaluated.
Separate exam	Yes

Separate exam

Exam Type	undefined
Details	Evaluation of scientific literature and evaluation of the quality of software architectures for distributed automation systems with regard to given automation issues. Presentation of results and scientific discourse in the whole group.

Minimum standard	Students research at least two relevant, independent, scientifically serious sources (whitepapers, standards, technical articles, reference books, ...) on the selected seminar topic, evaluate these sources scientifically and evaluate the researched statements and results with regard to given questions in the field of automation engineering and software engineering. They report on the researched statements and results as well as their personal evaluation and classification with regard to the given questions and are able to explain the results and justify the personal evaluation within the scope of a technical discussion.

− Lecture / Exercises

Learning goals

Goal type	Description
Skills	Develop software artifact of a distribution architecture for complex software systems Carry out project planning in distributed teams with an agile process model Perform extensive system analysis with respect to the role of the artifact in the distribution architecture Determine design input requirements for the development of the artifact Specify and model the software artifact based on the design input requirements Select and justify design principles and patterns to achieve defined quality objectives Derive interfaces, behavioral and structural models iterativly based on patterns in UML2 notations Use professional UML2 design tool purposefully Verify and evaluate models, correct model errors and optimize models Programming software artifacts in C++ define meaningful test scenarios and verify software artifacts Evaluate the quality of the software artifact Present the team's project results to a professional audience in a compact and target-group-oriented way

Expenditure classroom teaching

Type	Attendance (h/Wk.)
Project	1
Tutorial (voluntary)	0

Special requirements

none

Accompanying material	- Digitally specified project task (design input requirements specification) - Development tool for UML modeling - integrated development environment for programming in C++
Separate exam	Yes

Separate exam

Exam Type	EN Projektaufgabe im Team bearbeiten (z.B. im Praktikum)
Details	3 attendance appointments of 4h each per project group, final presentation and discusssion
Minimum standard	- justified design of an appropriate software architecture, making full use of reuse strategies and demonstrating that the essential design requirements are met or can be met by extending the architecture. - justified proof of implementability of the software architecture (feasibility study, C++ prototype) - evaluation of the quality of the software architecture