

Lehrveranstaltung

SEKM - Software Engineering mit Komponenten und Mustern

Version: 1 | Letzte Änderung: 06.10.2019 18:52 | Entwurf: 0 | Status: vom verantwortlichen Dozent freigegeben

^ Allgemeine Informationen

Langname	Software Engineering mit Komponenten und Mustern
Anerkennende LModule	QEKS MaET , QEKS MaTIN
Verantwortlich	Prof. Dr. Stefan Kreiser Professor Fakultät IME
Niveau	Master
Semester im Jahr	Wintersemester
Dauer	Semester
Stunden im Selbststudium	78
ECTS	5
Dozenten	Prof. Dr. Stefan Kreiser Professor Fakultät IME
Voraussetzungen	<ul style="list-style-type: none">- Programmierkenntnisse in einer objektorientierten Programmiersprache, bevorzugt C++- Kenntnisse in Software-Modellierung mit Hilfe der Unified Modeling Language (UML) oder anderen (formalen) Sprachen, die das Modellieren von Schnittstellen, Verhalten und Strukturen unterstützen- grundlegende Kenntnisse in (agilem) Projektmanagement- grundlegende Softwarearchitekturmodelle- Kommunikationsmodelle in Softwaresystemen (OSI, TCP/IP, Messaging)
Unterrichtssprache	deutsch und englisch
separate Abschlussprüfung	Ja

Abschlussprüfung

Details

Mündliche Prüfung nach schriftlicher Vorbereitung.

Anhand einer realitätsnahen Aufgabenstellung angemessener Komplexität entwickeln und modellieren die Studierenden eine geeignete Softwarearchitektur für ein verteiltes Automatisierungssystem unter angemessener Anwendung von Strategien zur Wiederverwendung von Modell- und/oder Softwareartefakten. Sie begründen die essenziellen Strukturen ihrer Architektur unter Bezugnahme auf die spezifische Zielsetzung und die spezifischen Umgebungsbedingungen für den Einsatz des jeweiligen Automatisierungssystems sowie unter Bezugnahme auf grundlegende Qualitätskriterien für automatisierungstechnische Softwaresysteme (System-, Entwicklungs-, Betriebs-, Service- und Wartungsanforderungen). Sie erläutern, welche besonderen organisatorischen Rahmenbedingungen sich für die Entwicklung aus der Softwarearchitektur ergeben, und bewerten die Qualität der Architektur aus technischer und betriebswirtschaftlicher Sicht.

Mindeststandard

- Studierende extrahieren die wesentlichen relevanten Informationen, Rahmenbedingungen und Lösungseinschränkungen aus der Aufgabenspezifikation und entwerfen ein Modell der Softwarearchitektur unter Berücksichtigung grundlegende Qualitätskriterien für automatisierungstechnische Softwaresysteme. Dazu wählen Sie eine sinnvolle Strategie zur Wiederverwendung von Modell- und/oder Softwareartefakten aus und begründen das Vorgehen.

- Studierende erläutern die wesentlichen Strukturen ihrer Softwarearchitektur im Hinblick auf die Einpassung wiederverwendeter Modelle und Artefakte und begründen sie im Hinblick auf die gegebenen Systemanforderungen, also technische Systemanforderungen sowie weitere, ggfs. relevante, Entwicklungs-, Betriebs-, Service- und Wartungsanforderungen.

Prüfungstyp

Mündliche Prüfung nach schriftlicher Vorbereitung.

Anhand einer realitätsnahen Aufgabenstellung angemessener Komplexität entwickeln und modellieren die Studierenden eine geeignete Softwarearchitektur für ein verteiltes Automatisierungssystem unter angemessener Anwendung von Strategien zur Wiederverwendung von Modell- und/oder Softwareartefakten. Sie begründen die essenziellen Strukturen ihrer Architektur unter Bezugnahme auf die spezifische Zielsetzung und die spezifischen Umgebungsbedingungen für den Einsatz des jeweiligen Automatisierungssystems sowie unter Bezugnahme auf grundlegende Qualitätskriterien für automatisierungstechnische Softwaresysteme (System-, Entwicklungs-, Betriebs-, Service- und Wartungsanforderungen). Sie erläutern, welche besonderen organisatorischen Rahmenbedingungen sich für die Entwicklung aus der Softwarearchitektur ergeben, und bewerten die Qualität der Architektur aus technischer und betriebswirtschaftlicher Sicht.

^ Vorlesung / Übungen

Lernziele

Kenntnisse

Begriffe

Wert einer technischen Software

verteiltes Softwaresystem, Nebenläufigkeit

Softwarequalität, Dienstgüte, Refactoring

Komplexität (algorithmische, strukturelle), Emergenz

Wiederverwendung (Re-Use), Symmetrie und Symmetrioperationen, Abstraktion, Invarianten

Methodische Ansätze zur qualitätsgesteuerten Wiederverwendung

Varianten für White Box Reuse

Black Box Reuse

Grey Box Reuse (Wiederverwendungshierarchie)

Re-Use in automatisierungstechnischen Softwaresystemen

Determinismus

Vorteile und Herausforderungen

angepasste Vorgehensmodelle und Personalstrukturen

vorhersagbare Zielerreichung in Entwicklungsprojekten (Produktqualität, Kosten, Zeit)

arbeitsteilige Entwicklung, Wartung und Pflege von Softwaresystemen

Muster (Pattern)

Musterbeschreibung mit UML

grundlegende Architekturmuster

Erzeugungsmuster

Strukturmuster

Verhaltensmuster

klassenbasierte (statische) vs. objektbasierte (dynamische) Muster

grundlegende Muster für nebenläufige und vernetzte Echtzeitsysteme

Muster zur Kapselung und zur rollenbasierten Erweiterung von Layerarchitekturen

Muster für Nebenläufigkeitsstrukturen zur Durchsatzoptimierung und Latenzzeitminimierung

Muster zur verteilten Ereignisprozessierung

Muster zur Prozesssynchronisation

Aufbau und Nutzung von Musterkatalogen, Mustersprachen

musterbasierter Entwurf komplexer Softwaresysteme

Komponenten und Frameworks

Designprinzipien

Schnittstellenarchitektur

aktive und passive Systemelemente

Entwurf, Programmierung und Test

Qualität

Konfiguration und Nutzung

Middlewaresysteme in Architekturen technischer Softwaresysteme

ORB-Architekturen am Beispiel CORBA und TAO

integrierte Systemplattformen am Beispiel MS .NET

Multiagentensysteme (MAS)

Architekturmodelle für Agenten

Kollaboration zwischen Agenten

Agentensprachen

Einsatzabwägung

Fertigkeiten

Muster zur Gestaltung komplexer Softwaresysteme einsetzen

Verwendungszweck, Einsatzgrenzen, invariante und parametrierbare Anteile von Mustern aus Literaturquellen in englischer und deutscher Sprache ableiten und diskutieren

Implementierungsskelette von Mustern nachvollziehen und auf Aufgabenstellungen mit eingeschränktem inhaltlichen Fokus transferieren

Vorteile objektorientierter Programmiersprachen diskutieren

wiederkehrende Aufgabenstellungen beim Entwurf komplexer SW-Systeme ableiten

Muster beispielhaft implementieren und Beispielimplementierungen prüfen

Muster sinnvoll kombinieren, um wiederkehrende Aufgabenstellungen mit verbreiterem inhaltlichen Fokus zu lösen

UML2-Notationen nutzen

Professionelles UML2-Entwurfswerkzeug für Round-Trip-Engineering nutzen

Integration anhand der Beispielimplementierungen der zu kombinierenden Muster durchführen

Integrationstest durchführen, Lösung bewerten und optimieren

Black-Box-Komponenten musterbasiert konstruieren

Komponentenbasierte Softwarearchitekturen analysieren

sinnvolle Anwendungsbereiche aus den Architekturvorgaben ableiten

Vorgehen zur Konstruktion von Anwendungen diskutieren (Anwendungsebene erkennen)

aktive und passive Systemelemente erkennen und Laufzeitverhalten ableiten

abstrakte Umgebungsschnittstellen zur Vernetzung, Konfiguration und Aktivierung von Komponenten erkennen

abstrakte Anwendungsschnittstellen zum Datenaustausch erkennen

Systemerweiterungspunkte finden (funktionale und strukturelle Parametrierungsebene erkennen)

Verteilungsarchitekturen analysieren

Essenzielle Systemdienste erkennen, beschreiben, einordnen und begründen

strukturgebenden Architekturartefakten sinnvolle Lösungsmuster zuordnen

sinnvolle Anwendungsbereiche aus den Architekturvorgaben ableiten

Vorgehen zur Konstruktion von Anwendungen diskutieren (Anwendungsebene erkennen)

Eigenschaften und Einsatzgrenzen von Kommunikationsprotokollen diskutieren

vorgesehene Systemerweiterungspunkte finden

Multiagentensysteme mit konventionellen Verteilungsarchitekturen vergleichen

Agent vs. Komponente

Architekturmodelle

Aktivierungsmechanismen

Verteilungsmechanismen

Kommunikationsprotokolle und Kollaborationsmechanismen

Einsatzgebiete und Einsatzgrenzen

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)
Vorlesung	1
Übungen (ganzer Kurs)	1
Übungen (geteilter Kurs)	0
Tutorium (freiwillig)	0

Separate Prüfung

keine

^ Seminar

Lernziele

Kenntnisse

anspruchsvolle Seminarthemen können z. B. aus den folgenden oder fachlich angrenzenden Themengebieten definiert werden:

- wiederverwendbare Artefakte zum Aufbau der Architektur verteilter Softwaresysteme,
 - professionelle Verteilungsarchitekturen,
 - Multiagentensysteme,
 - besondere betriebswirtschaftliche, haftungsrechtliche und ethische Anforderungen bei Softwaresystemen mit (verteilter) künstlicher Intelligenz und deren Auswirkungen auf die Gestaltung von Softwarearchitekturen
-

Fertigkeiten

eigene Arbeitsergebnisse und Arbeitsergebnisse des Teams schriftlich und mündlich kompakt und zielgruppengerecht präsentieren

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)
Seminar	1
Tutorium (freiwillig)	0

Separate Prüfung

Prüfungstyp

Fachgespräch (Interview) zu besonderen Fragestellungen in Szenario, Projektaufgabe oder Literaturrecherche

Details

Auswertung wissenschaftlicher Literatur und Bewertung der Qualität von Softwarearchitekturen für verteilte Automatisierungssysteme im Hinblick auf vorgegebene automatisierungstechnische Fragestellungen.

Ergebnisvorstellung und wissenschaftlicher Diskurs in der Gesamtgruppe.

Mindeststandard

Studierende recherchieren mindestens zwei relevante, voneinander unabhängige, wissenschaftlich seriöse Quellen (Whitepaper, Normen, Fachartikel, Fachbücher, ...) zum gewählten Seminarthema, werten diese Quellen wissenschaftlich aus und bewerten die recherchierten Aussagen und Ergebnisse im Hinblick auf die vorgegebenen automatisierungstechnischen und softwaretechnischen Fragestellungen. Sie berichten über die recherchierten Aussagen und Ergebnisse sowie die persönliche Bewertung und Einordnung im Hinblick auf die vorgegebenen Fragestellungen und sind im Rahmen einer Fachdiskussion in der Lage, die Ergebnisse zu erläutern und die persönliche Bewertung zu begründen.

^ Projekt

Lernziele

Fertigkeiten

Softwareartefakt einer Verteilungsarchitektur für komplexe Softwaresysteme entwickeln

Projektierung in verteilten Teams mit agilem Vorgehensmodell durchführen

umfangreiche Systemanalyse hinsichtlich der Rolle des Artefakts in der Verteilungsarchitektur durchführen

Anforderungen an das Softwareartefakt ermitteln

Softwareartefakt basierend auf den Anforderungen spezifizieren und modellieren

Designprinzipien und Muster zum Erreichen definierter Qualitätsziele auswählen und begründen

Schnittstellen-, Verhaltens- und Strukturmodelle musterbasiert in UML2-Notationen iterativ herleiten

Professionelles UML2-Entwurfswerkzeug zielgerichtet einsetzen

Modelle verifizieren und bewerten, Modellfehler korrigieren und Modelle optimieren

Softwareartefakt in C++ programmieren

sinnvolle Prüfzenarien definieren und Softwareartefakt verifizieren

Qualität des Softwareartefakts bewerten

Arbeitsergebnisse des Teams kompakt und zielgruppengerecht präsentieren

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)
Projekt	1
Tutorium (freiwillig)	0

Separate Prüfung

Prüfungstyp

Projektaufgabe im Team bearbeiten (z.B. im Praktikum)

Details

3 Präsenztermine je 4h je Projektgruppe, Abschlusspräsentation mit Diskussion

Mindeststandard

- Begründeter Entwurf einer angemessenen Softwarearchitektur unter umfassendem Einsatz von Wiederverwendungsstrategien und Nachweis, dass die wesentlichen Designanforderungen erfüllt sind oder durch Erweiterung der Architektur erfüllt werden können
- Begründeter Nachweis der Implementierbarkeit der Softwarearchitektur (Machbarkeitsstudie, C++ Prototyp)
- Bewertung der Qualität der Softwarearchitektur