Technology
Arts Sciences
TH Köln

Lehrveranstaltungshandbuch SEA

Software Engineering für die Automatisierungstechnik

Version: undefined | Letzte Änderung: - | Entwurf: undefined | Status: undefined

Allgemeine Informationen

Langname	Software Engineering für die Automatisierungstechnik
Anerkennende LModule	SE_BaET
Verantwortlich	Prof. Dr. Stefan Kreiser Professor Fakultät IME
Gültig ab	Wintersemester 2020/21
Hinweis zum Zeitraum des Lehrangebots	Vorlesung / Übung wöchentlich (Flipped Classroom), Projektarbeit
Niveau	Bachelor
Semester im Jahr	Sommersemester
Dauer	Semester
Stunden im Selbststudium	90
ECTS	5
Dozenten	Prof. Dr. Stefan Kreiser Professor Fakultät IME

Literatur

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

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

Oestereich, Bernd et. al.: Analyse und Design mit der UML 2.5: Objektorientierte Softwareentwicklung, Oldenbourg Wissenschaftsverlag

Litke, H.D.: Projektmanagement - Handbuch für die Praxis: Konzepte - Instrumente - Umsetzung, Carl Hanser Verlag

Abschlussprüfung

Details	Natürlichsprachliche Beschreibung eines realitätsnahen Automatisierungssystems (Textaufgabe): Modellierung in UML, begründete Umsetzung in C++ 20min schriftliche Vorbereitung, 20min mündliche Befragung zur vorbereiteten Lösung mit Möglichkeit zur Optimierung.

Voraussetzungen	- grundlegende Kenntnisse zur Verhaltensmodellierung (z.B. PAP, Automaten, State Charts, Petrinetze) - grundlegende Programmierkenntnisse in C/C++ - grundlegende Kenntnisse in Objektorientierung (Klassen, Objekte)
Unterrichtssprache	deutsch und englisch
separate Abschlussprüfung	Ja

Mindeststandard	Anhand einer natürlichsprachlichen Beschreibung eines realitätsnahen Automatisierungssystems angemessener Komplexität modellieren die Studierenden das Systemmodell eines zur Lösung der Automatisierungsaufgabe geeigneten Softwaresystems und begründen und bewerten die wesentlichen Eigenschaften ihres Entwurfs. Zur Begründung und Bewertung nehmen die Studierenden Bezug auf die spezifischen Anforderungen an das Automatisierungssystem sowie auf grundlegende Qualitätskriterien für automatisierungstechnische Softwaresysteme (System-, Entwicklungs-, Betriebs-, Service- und Wartungsanforderungen) und zeigen dabei an ausgewählten Modellartefakten insbesondere, dass sich und wie sich das Systemmodell in ein Softwaremodell und anschließend in ein Implementierungsmodell transformieren lässt und welche Konsequenzen ihr Entwurf für die Modelle der nachfolgenden Entwurfsphasen hat.
Prüfungstyp	mündliche Prüfung, strukturierte Befragung

− Vorlesung / Übungen

Lernziele

Zieltyp	Beschreibung
Kenntnisse	Begriffe - Softwaresystem, Softwareprodukt - Softwarequalität - Komplexität Objektorientiertes Modellieren mit UML - Domänenmodell (Struktur, Verhalten, Systemgrenzen / Schnittstellen) - Softwarearchitekturmodell - Implementierungsmodell - Modelltransformationen - Modellierungswerkzeuge Vorgehensmodelle - lineare (Phasenmodell, V-Modell) - evolutionäre (eXtreme Programming, Scrum, Timebox) Qualitätsmanagement (SOPs) Anforderungsanalyse - Requirements Engineering - Design-Input-Requirements (Lastenheft) - Gesetze, Normen und organisatorische Vorgaben Produktrisikoanalyse (FMEA, FTA) Entwurf - Designprinzipien - Machbarkeitsstudien - Systemspezifikation (Pflichtenheft) - Softwarespezifikationen Implementierung - Wahl der Programmiersprachen, Programmierrichtlinien - Entwicklung in verteilten Teams, Entwicklertest - Systemintegration - Inbetriebnahme Verifikation & Validierung - Formalisierte Softwaretests (dynamische, statische) - Feldevaluation - Betriebsbegleitung Managementaufgaben - Dokumentmanagement - Konfigurationsmanagement (Versionsmanagement, Buildmanagement) - Testmanagement - Änderungsmanagement

Besondere Literatur

keine

Besondere Voraussetzungen

keine

Begleitmaterial	elektronische Folien zur Vorlesung, elektronische Übungsaufgabensammlung, professionelles Entwicklungswerkzeug für Unified Modeling Language (UML2) , Vorlesungsvideos
Separate Prüfung	Nein

Fertigkeiten	Technische Softwaresysteme analysieren - Systemanforderungen methodisch ermitteln, konsolidieren und priorisieren - formalisierte Anforderungsspezifikation entwerfen Technische Softwaresysteme modellieren - Notationen der Unified Modeling Language zur Modellierung einfacher Softwaresysteme nutzen - Strukturnotationen (Klassendiagramm, Paketdiagramm, Komponentendiagramm, Verteilungsdiagramm) - Verhaltens- und Schnittstellennotationen (Anwendungsfalldiagramm, Aktivitätsdiagramm und Aktionskonzept Zustandsautomat und Protokollautomat, Sequenzdiagramm) Modellierungsebenen benennen und abgrenzen - Systemmodell (Kundensicht): Entitätenmodell, Schnittstellenmodell, Verhaltensmodell - Softwaremodell (Entwicklersicht): Technische Klassenmodelle, detaillierte Verhaltensmodelle, Designprinzipien, grundlegende Softwarearchitekturen Kontext, Grenzen, Aufgaben, Verhalten und Strukturen einfacher Softwaresysteme aus Texten ableiten - technische Textabschnitte vollständig erfassen - implizite Angaben erkennen und verstehen - Inkonsistenzen erkennen und auflösen - fehlende Angaben erkennen und ableiten bzw. erfragen Softwaresysteme mit UML2-Notationen modellieren - einfache Systemmodelle iterativ entwerfen (Entitätenmodell entwerfen, Kontext- und Anwendungsfallmodell aus Kundensicht entwerfen, Anwendungsfälle detaillieren, Standardszenario und wesentliche Alternativszenarien beschreiben und als Aktivitätsdiagramm verfeinern) - einfache Softwaremodelle iterativ entwerfen (Refactoring und

Detaillierung des Entitätenmodells aus Entwicklersicht, Verhaltensmodelle aus Entwicklersicht detaillieren, strukturbasiertes Verhalten als State Chart modellieren, Aktivitäten bis zur Aktionsebene verfeinern, Zusammenhang zwischen Aktionen und Klassenmethoden herstellen)
Professionelles UML2-Entwurfswerkzeug bedienen
Modelle verifizieren
- Bewertungskriterien definieren
- Einhalten von Modellierungsvorgaben und Designprinzipien
- Vollständigkeit bzw. unnötige Komplexität
- Qualität im Hinblick auf spezifische Kundenvorgaben bewerten (Testfälle definieren, Modellreviews durchführen und dokumentieren, Modellfehler entdecken und benennen, Modelle anhand der Bewertungen korrigieren und optimieren)
Technische Softwaresysteme entwerfen
- Produktrisiken ermitteln, Milderungsmaßnahmen definieren und im Entwurf berücksichtigen
- Designprinzipien zum Erreichen definierter Qualitätsziele benennen, erläutern und anwenden
- problemgerechte System- und Softwarearchitektur auswählen und anwenden
- Methoden zur Softwareentwicklung in verteilten Teams erläutern und exemplarisch anwenden
- Methoden zur Softwareprüfung in verteilten Teams erläutern und exemplarisch anwenden
Technische Softwaresysteme qualitätsgesteuert entwickeln
- Vorgehensmodelle exemplarisch anwenden)
- Informationen aus internationalen Standards zur Softwareentwicklung gewinnen (Deutsch/Englisch)

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)

Vorlesung	1
Übungen (ganzer Kurs)	2

− Projekt

Lernziele

Zieltyp	Beschreibung

Besondere Literatur

keine

Besondere Voraussetzungen

- grundlegende objektorientierte Programmierkenntnisse (C/C++)
- grundlegende Fertigkeiten im Umgang mit einer IDE / Debugging

Begleitmaterial	Design Input Requirements für Projektaufgabe, Roundtrip Engineering Aufgabe zur Einarbeitung in UML, UML-Modellierungswerkzeug und IDE (Integrierte Entwicklungsumgebung)
Separate Prüfung	Ja

Separate Prüfung

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

Fertigkeiten	Größere technische Softwaresysteme analysieren - umfangreiche technische Texte erfassen und verstehen, insbesondere englischsprachige Texte - umfangreiche Systemanforderungen auswerten und anordnen Größere technische Softwaresysteme modellieren - Modellierungsebenen abgrenzen: Systemmodell (Kundensicht), Softwaremodell (Entwicklersicht) - Modellnotationen systematisch zur Systembeschreibung nutzen - Schnittstellen-, Verhaltens- und Strukturmodelle in UML2-Notationen iterativ herleiten - Professionelles UML2-Entwurfswerkzeug zielgerichtet einsetzen - Modelle verifizieren und bewerten, Modellfehler korrigieren und Modelle optimieren Größere technische Softwaresysteme entwerfen - Designprinzipien zum Erreichen definierter Qualitätsziele auswählen und anwenden - problemgerechte System- und Softwarearchitektur auswählen und anwenden - Softwareentwicklung und Softwareprüfung in verteilten Teams durchführen Quellcode erstellen und prüfen - gegebenen Quellcode analysieren und zielgerichtet erweitern - objektorientierte Programmiersprache (C++) nutzen Größere technische Softwaresysteme qualitätsgesteuert entwickeln - evolutionäres Vorgehensmodell anwenden - Informationen aus internationalen Standards zur Softwareentwicklung gewinnen (Deutsch/Englisch) Arbeitsergebnisse des Teams in englischer Sprache kompakt und zielgruppengerecht präsentieren Handlungskompetenzen zeigen: Realweltsysteme modellieren - Dekomposition (Systemgrenzen erkennen bzw. definieren und korrekt nutzen, Systemschnittstellen erkennen bzw. definieren und korrekt

Details	Teil 1 (Round-Trip Engineering): nachvollziehbare, auf Basis der Anforderungen und Rahmenbedingungen begründete Transformationen zw. Systemmodell, Softwaremodell, Implementierungsmodell und Quellcode durchführen. Teil 2 (Projektaufgabe): Anhand einer natürlichsprachlichen Beschreibung (Englisch) eines realitätsnahen Automatisierungssystems angemessener Komplexität modellieren die Studierenden das Systemmodell eines zur Lösung der Automatisierungsaufgabe geeigneten Softwaresystems und begründen und bewerten die wesentlichen Eigenschaften ihres Entwurfs. Zur Begründung und Bewertung nehmen die Studierenden Bezug auf die spezifischen Anforderungen an das Automatisierungssystem sowie auf grundlegende Qualitätskriterien für automatisierungstechnische Softwaresysteme (System-, Entwicklungs-, Betriebs-, Service- und Wartungsanforderungen) und zeigen dabei an ausgewählten Modellartefakten insbesondere, dass sich und wie sich das Systemmodell in ein Softwaremodell und anschließend in ein Implementierungsmodell transformieren lässt und welche Konsequenzen ihr Entwurf für die Modelle der nachfolgenden Entwurfsphasen hat.

nutzen, Systemstrukturen erkennen bzw. definieren und korrekt darstellen, Systemfunktionen erkennen bzw. definieren und korrekt darstellen)
- Komposition (Struktur- und Verhaltensmodelle erstellen, Modelle integrieren, Teilmodelle und Gesamtmodell verifizieren und bewerten)
- komplexe Aufgaben arbeitsteilig im Team bewältigen (einfache Projekte planen und steuern, Absprachen und Termine einhalten, Reviews planen und durchführen)
- Modelltransformationen anwenden (Modellelemente aus gegebenem C++ Quellcode zurückführen, Modelle durch manuelle Quellcodeanalyse vervollständigen und verifizieren, Systemerweiterungen und Lösungsmodifikationen auf Basis einer aktuellen Spezifikation modellieren, Quellcode aus neuem Modell generieren und generierten Quellcode manuell vervollständigen, Implementierung im Debugger und durch systematische Tests auf dem Zielsystem verifizieren)

Mindeststandard	Teil 1: Lauffähiges Softwaresystem, das nachweislich die geforderten Eigenschaften besitzt und konsistent zum UML-Modell ist. Teil 2: Systemmodell, das nachweislich die wesentlichen Anforderungen aus den DIRs erfüllt, Softwarearchitekturkonzept, das das zur Umsetzung des Systemmodells unter Berücksichtigung aller DIRs begründet geeignet ist.

Aufwand Präsenzlehre

Typ	Präsenzzeit (h/Wo.)
Projekt	1