Modules
International Master Program
‘Software Engineering for Embedded Systems’

Mai 2019

Table of Contents

Preamble

In the context of a distance study degree program, the course of studies “Software Engineering for Embedded Systems” shall provide graduates the opportunity to obtain the qualification necessary for independent and responsible work in the area of software engineering.

The students shall acquire:

1. Basic skills in the following areas:
   - Software Engineering and the development of large and complex software systems
   - Software development for embedded systems
   - Planning and execution of software projects
2. Advanced skills in the following areas:
   - Quality assurance
   - Requirements analysis and specification in software development
   - Definition, development, and evaluation of software product lines
   - Distribution of system functionality to different software components
   - Requirements of real-time systems on software development
   - Development of safety-critical software
3. The skills acquired will be applied in a studio module spanning two semesters (second and third semester) in order to promote the acquisition of practice-oriented skills.

The following **academic objectives** are the basis of the distance study program:

1. The students have profound knowledge and a critical awareness regarding the principles, general procedures, and models in software engineering for embedded systems.
2. The students have the ability to elicit, formulate, and formalize problems on all levels of abstraction, and are able to solve these problems in an engineering-style manner by applying critical thinking and using profound judgment.
3. The students are capable of combining software and hardware knowledge, applying state-of-the-art methods and techniques, and know what the non-technical effects of their work as software engineers are on the application domains.
4. The students are capable of using the acquired knowledge and understanding for designing the different software engineering artefacts and for applying new innovative solution methods and they are capable of learning new software engineering technologies and evolving them.
5. The students are able to conduct activities in the area of software engineering for embedded systems independently, to introduce new technologies in an organization, and to lead an interdisciplinary team in the context of embedded system development.

The expected **learning outcomes** of the degree program are:

1. The graduates shall have knowledge and understanding regarding the principles of software engineering for embedded systems in general and their impact on software engineering methods, techniques, and tools in particular.
2. They shall have learned lessons regarding general software engineering procedures and models, such as lifecycle, quality, process, and product models for the software development of embedded systems.

3. They shall have acquired a critical awareness regarding current findings in software engineering for embedded systems (e.g., new requirements on software development for safety-critical realtime systems).
4. They shall be capable of understanding problems and development challenges from the application areas of embedded systems on all levels of abstraction.
5. They shall be capable of eliciting, formulating, and formalizing problems and implementing engineering-style solutions by selecting and applying suitable methods, techniques, and tools.
6. They shall be capable of critical thinking, of making judgments and decisions on the basis of complex, ambiguous, and incomplete information (e.g., incomplete customer requirements, yet unknown development/evolution in product lines).
7. They shall be capable of combining software and hardware development knowledge and of reducing and mastering complexity through the use of appropriate techniques.
8. The graduates shall be capable of selecting and applying techniques, methods, and tools in the area of software engineering for embedded systems, giving consideration to their limitations and benefits for a specific application context.
9. They shall possess professional knowledge in software engineering for embedded systems and the most up-to-date knowledge regarding state-of-the-art processes, methods, techniques, and tools.
10. They shall possess knowledge about the non-technical effects of their practical work as computer scientists.
11. They shall be capable of using their knowledge and understanding to design and improve models, products, and processes in the area of embedded system development in a goal-oriented manner.
12. They shall be capable of applying innovative methods for solving problems in the area of embedded system development.
13. They shall be capable of understanding, evaluating, and evolving new methods, techniques, and tools in the area of software engineering for embedded systems.
14. The graduates shall be capable of working independently in the development of embedded systems and shall be able to introduce new technologies in an organization and assume the role of an “Agent of Change“.

A. Basic Modules

B-M.1 – Software Engineering Basics

Module name:	Software Engineering Basics
Abbreviation:	Module No. B-M.1
Semester:	1st (winter semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	Prof. Dr. Dieter Rombach, Prof. Dr. Norbert Wehn, Prof. Jesse Poore (†) and Dr. Stacy Prowell (authors of written lectures)
Language:	English
Classification within the curriculum:	In its first part this module supplies basic principles and important fundamentals of software engineering in general. The techniques described in this part apply for all types of software systems and provide the foundation for more specialized modules. The second part focuses on development fundamentals, techniques, and approaches specific to embedded systems development.
Teaching format / class hours per week during the semester[1] (SWS):	Textbook, self-control assignments, online tutorial (17 weeks), on-campus weekend / 8
Workload:	Contact study workload: 45 hrs per semester[2] Self-study workload: 130 hrs per semester Overall workload: 175 hrs per semester
Credit points:	7
Recommended prerequisites:	Theoretical foundations: mathematical foundations, basic number domains, basic proof techniques, complexity-O-notation, set theory, propositional logics, first order logics and predicate logics System foundations: combinatorial and sequential circuits, performance of computers, functionality of single processor computers, memory hierarchies, caches, pipelining, bus hierarchies Software development foundations: basic programming techniques, elementary and abstract data types, elementary algorithms, algorithms and data structures for searching and sorting, ability to determine characteristics of algorithms, interaction between software modules and larger software systems, fundamentals in design patterns and elementary modeling techniques.

[1] Class hours per week during the semester corresponds to „Semesterwochenstunden (SWS)“

[2] 12 consultations x 3 hrs = 36 hrs; 9 hrs on-campus weekend

Targeted learning outcomes:	After studying the first part (Software Engineering Introduction) the students • know the similarities and differences between software engineering (SE) and hardware engineering, • know about relevance of non-functional properties in embedded software, • understand the difference between reliability and correctness, • understand the abstraction levels used for hardware and software engineering, • know the existing body of SE knowledge and the risks of not applying it, • understand the basic principles of engineering-based software development and are able to relate these principles to concrete technologies, • know the V-Model as model describing a product lifecycle, • understand lifecycle processes, • know different types of empirical studies, and how to apply them to obtain information about products and processes, • know the most important laws in software engineering, • have basic knowledge regarding requirements elicitation, management, and specification, • have basic knowledge regarding architecture development and view-based architecture specification, • know about the concepts of units, unit interfaces, unit development, and unit testing, • are able to judge and apply the key techniques, tools, and methods for software engineering. After studying the second part (Software Development for Embedded Systems) the students • obtain an awareness of the role that embedded systems developers play in product safety and in meeting industrial or government standards, • understand the principles of discrete and continuous time and their implications to embedded systems development, • understand the basics of digital signal processing and the concept of control loops, • understand information flows within typical embedded systems, e.g. embedded control systems, • understand the concepts of automata and their role in embedded software development and testing, • understand the difference between time- and event triggered systems, as well as the difference between event based input and polling for input data, • know the important concepts of hardware architectures running embedded software, • understand the basic concepts of embedded operating systems and the difference between static and dynamic scheduling strategies, • learn how to use sequence-based specification as construction technique for embedded software, • understand structural design issues specific to embedded systems, • understand the aspects of code writing that are unique to embedded software,

	• understand automated statistical testing and system certification.
Content:	First part (Software Engineering Introduction): • Introduction (embedded systems, engineering processes, software engineering: the discipline) • Basics of hardware engineering (physical laws, abstraction in hardware engineering, design metrics and views in hardware engineering, role of languages in hardware engineering, the future in hardware engineering) • Basics of software engineering (scientific basis, software engineering overview, software challenges, software engineering principles, logical software product model & the V-Model, lifecycle processes, techniques, methods, and tools) • Existing body of knowledge (empirical studies, example laws) • Requirements specification, elicitation, V&V, and management (introduction and principles, requirements engineering process, requirements documentation, requirements engineering elicitation, negotiation and analysis, Verification & Validation, requirements management) • Architecture/design, verification/inspection & testing (introduction and principles, planning, realization, documentation, assessment) • Unit development, Verification & Testing (introduction, unit interface specification, unit design, unit implementation, unit Verification & Validation) • Summary and outlook Second part (Software Development for Embedded Systems): • Introduction (applications, the dependability dilemma, dependability assurance cases, dependability and standards, best practice, code generation, proprietary versus open source, case study) • Abstractions (continuity, time ,non–determinism, reliability, number fields and computer arithmetic) • Automata fundamentals (strings and abstract languages, languages, relations, functions, partitions, state machines and sequence recognizers) • Software fundamentals (correct by design, design for verification, modularization and information hiding, box structure design) • Hardware fundamentals (information flow and discrete control, microcontrollers, memory, bicycle computer case study) • Operating system fundamentals (characteristics of embedded operating systems, components of embedded operating systems, scheduling, rate monotonic scheduling, resource control, hardware abstraction) • Requirements analysis (requirements, system boundary, sequence enumerations, sequence abstraction and predicate refinement, canonical sequence analysis, Mealy machine generation, code generation) • Architecture fundamentals (key design issues, incremental development, V–model) • Code development (embedded source code, the virtual machine, style guides, structured programming, code writing, code review) • Automated statistical testing (application of statistical science to testing, failures in the field, understanding the software intensive system and its use, model validation with customer and user, representing usage models with constraints, the benefits of usage

	modeling, product and process improvement, conclusion) • Case study: cycle computer
Exam / Study achievements:	Written examination and participation in the on-campus weekend at the end of the semester
Forms of media:	Written lectures provided via internet (PDF)
Literature:	First part (Software Engineering Introduction): Endres A., Rombach D.: A handbook of Software and systems Engineering – Empirical Observations, Laws and Theories, Pearson, 2003 Jalote, P.: “A Concise Introduction to Software Engineering”, Springer London, 2010 Sommerville, I.: “Software Engineering”, 9. Edition, Addison Wesley, 2012. Second part (Software Development for Embedded Systems): Oshana, R.: DSP Software Development Techniques for Embedded and Real-Time Systems, Elsevier, Oxford, UK, 2006 Prowell, S.J., Trammell, C.J., Linger, R.C., and Poore, J.H.: Cleanroom Software Engineering: Technology and Process, Addison-Wesley-Longman, Boston, 1999 Simon, D.E.: An Embedded Software Primer, Addison-Wesley, New York, 1999 Liggesmeyer P., Rombach D. (eds.): Software Engineering eingebetteter Systeme (in German), Elsevier, 2005

B-M.2 – Project Management

Module name:	**Project Management**
Abbreviation:	Module No. B-M.2
Semester:	1st (winter semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	Dr. G. Pews (author of written lectures)
Language:	English
Classification within the curriculum:	It is a basic module which covers relevant foundations of project management.
Teaching format / class hours per week during the semester (SWS):	Textbook, self-control assignments, online tutorial (7 weeks) / 3
Workload:	Contact study workload: 12 hrs per semester[3] Self-study workload: 88 hrs per semester Overall workload: 100 hrs per semester
Credit points:	4
Recommended prerequisites:	Theoretical foundations: mathematical foundations, basic number domains, basic proof techniques, complexity-O-notation, set theory, propositional logics, first order logics and predicate logics System foundations: combinatorial and sequential circuits, performance of computers, functionality of single processor computers, memory hierarchies, caches, pipelining, bus hierarchies Software development foundations: basic programming techniques, elementary and abstract data types, elementary algorithms, algorithms and data structures for searching and sorting, ability to determine characteristics of algorithms, interaction between software modules and larger software systems, fundamentals in design patterns and elementary modeling techniques. Prerequisite modules: First part of Software Engineering Basics
Targeted learning outcomes:	After learning this module the students • know what a project is, and understand the interdependency between scope of work, time, budget, and quality, • understand what happens during a software development project,

[3] 4 consultations x 3 hrs = 12 hrs

	• understand the relationship between time, quality, and budget, • are able to initialize a project, • are able to plan a project, • are able to control and manage a project, • are able to perform basic project quality management.
Content:	• Project basics (what is a project, project constraints, diminishing returns, different project sizes) • The top-level course of a project (top-level view on a project, project idea, study, and initiation, project initialization, kick-off and touch-down, project execution, the impact of software development models on project execution) • Project charter, project objectives, and work breakdown structure (project objectives and charter, quality criteria, work breakdown structure) • Project organization and roles (organization basics, special roles within a project, sub-teams in a project, aligning project organization and business organization) • Cost estimation (basics, function point analysis, Delphi method, getting from implementation efforts to costs) • Project planning (basics, how to plan a project, advanced topics, planning techniques, planning tools) • Project controlling (basics, how to determine the current status, how to get a project back on schedule, managing requirements analysis, managing the writing of concept documents, managing system specification, managing system design, project reporting, change request management) • Quality assurance (basics, quality assurance and project management, quality management plan, testing, quality actions, formalisms, risk management) • Project management toolbox (teambuilding, leadership, meetings, project diary, list of open issues)
Exams / Study achievements:	Written examination at the end of the semester
Forms of media:	Written lectures provided via internet (PDF)
Literature:	Brooks, Frederick: The mythical man month, Addison Wesley, 1995. DeMarco, Tom: Slack, B&T, 2002. DeMarco, Tom; Lister, Tim: Peopleware: Productive Projects and Teams, 3rd Edition, Addison Wesley, 2013, ISBN 978-0321934116 Heldman, William: IT Project+™ Study Guide, Sybex Inc., 2002. Angermeier, Georg: Projektmanagement- Lexikon, www.projektmagazin.de, München, ISBN 3-00-018114-8 A Guide to the Project Management Body of Knowledge: PMBOK Guide, 5th Edition, Project Management Institute, 2013

B. Advanced Modules

V-M.1 – Software Quality Engineering

Module name:	**Software Quality Engineering**
Abbreviation:	Module No. V-M.1
Semester:	2nd (summer semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	Prof. P. Liggesmeyer and Dr. D. Muthig (authors of written lectures)
Language:	English
Classification within the curriculum:	This module is the first of the advanced modules. The first part covers relevant quality assurance techniques for software systems in general and embedded software systems in particular. The second part covers important aspects of software product lines and product line architectures.
Teaching format / class hours per week during the semester (SWS):	• Textbook, self-control assignments, online tutorial (15 weeks) / 5
Workload:	Contact study workload: 30 hrs per semester[4] Self-study workload: 145 hrs per semester Overall workload: 175 hrs per semester
Credit points:	7
Recommended prerequisites:	Theoretical foundations: mathematical foundations, basic number domains, basic proof techniques, complexity-O-notation, set theory, propositional logics, first order logics and predicate logics System foundations: combinatorial and sequential circuits, performance of computers, functionality of single processor computers, memory hierarchies, caches, pipelining, bus hierarchies Software development foundations: basic programming techniques, elementary and abstract data types, elementary algorithms, algorithms and data structures for searching and sorting, ability to determine characteristics of algorithms, interaction between software modules and larger software systems, fundamentals in design patterns and elementary modeling techniques. Prerequisite tests: Passed exams after modules B-M.1 and B-M.2

[4] 10 consultations x 3 hrs = 30 hrs

Targeted learning outcomes:	After studying the first part (Software Quality Assurance) the students • know classifications of test, analysis and verification techniques, • understand the differences between dynamic testing, statistical analysis and formal techniques, • are able to evaluate, select, and apply approaches of dynamic testing, statistical analysis, and formal verification, • know tools for statistical code analysis, • are able to perform formal inspections and reviews. After studying the second part (Software Product Line Engineering) the students • understand the differences between conventional software development and software development based on software product lines, • are able to plan and realize product lines, • understand the impacts of software product lines on the organization, • understand how the product line approach can be introduced to an organization, • know the relevant characteristics of software architectures for the product line approach, • know the techniques for realizing variability on the implementation level, • understand the relevance of maintenance and evolution of a reuse infrastructure, • understand how quality assurance activities can be integrated into the product line approach.
Content:	First part (Software Quality Assurance): • Introduction (motivation, definition of software quality assurance, definition of terms, classification of test analysis and verification techniques) • Function-oriented test (characteristics and aims, functional equivalence class construction, state-based test, other techniques, evaluation) • Control-flow testing (characteristics and objectives, statement coverage test, branch coverage test, condition coverage test, techniques for testing loops, path coverage test, evaluation) • Data-flow test (characteristics and objectives, Defs/Uses test, evaluation) • Tool-supported static code analysis (characteristics and objectives, style analysis, slicing, data-flow anomaly analysis, evaluation) • Software inspections and reviews (characteristic and objectives, formal inspections, structured walkthrough, reviews without review meeting, evaluation) Second part (Software Product Line Engineering): • Product line engineering (PLE) (This chapter discusses challenges in software industry and introduces product line engineering as one possible solution. The terms “product lines”, “product families”, and “product line engineering” are defined.) • Scoping (This chapter introduces the initial step of the overall product line life cycle, that is, the process of product line scoping.

	It describes the process of specifying requirements and expectations to be introduced by a product line infrastructure.) • Product line organizations (This chapter introduces product line engineering as a fundamental paradigm that impacts nearly all aspects of organizations. It refines the general product line life cycle and enables the description of product line organizations as complex hierarchies of product lines of product lines.) • Quality assurance (This chapter discusses the challenges of making quality assurance as efficient as product construction in a product line context. It defines an overall framework for defining quality strategies in the context of the overall product line life cycle and explains how quality assurance techniques applied in single system contexts today can be integrated in such strategies.) • Reuse infrastructures (This chapter discusses the core part of the product line life intro that is the reuse infrastructure. It focuses on the information that is captured, organized, and evolved as part of the infrastructure.) • Product line architectures (This chapter introduces the key role of architectures for the success of product lines. It briefly introduces the basic definition and principles of architectures but focuses then in-depth on the consideration of product-line-specific properties such as flexibility or reusability at the architectural level.) • Reusable components (This chapter discusses the pitfalls of defining reusable components and how product line concepts can help to avoid them. The chapter then elaborates on the design process of components that systematically address the variability required.) • Implementing variability (This chapter surveys available techniques for realizing variability at the implementation level and introduces a framework for classifying the different ways of implementing variability. Furthermore, aspect-oriented programming and pre-processor-based techniques are introduced in detail.)
Exam / Study achievements:	Written examination at the end of the semester
Forms of media:	Written lectures provided via internet (PDF)
Literature:	First part (Software Quality Assurance): Feigenbaum A.V., Total Quality Control, New York: McGraw-Hill 1983 Braverman J.D., Fundamentals of Statistical Quality Control, Reston: Reston Publishing Co., Prentice Hall 1981 Wheeler D.J., Chambers D.S., Understanding Statistical Process Control, Knoxville: SPC Press 1992 Gilb T., Graham D.: Software Inspection, Reading: Addison-Wesley, 1993 Wiegers, Karl: Peer Reviews in Software – A Practical Guide, Addison-Wesley Information Technology Series, 2002 Second part (Software Product Line Engineering): Apel, S., Batory, D., Kästner, C. & G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer,

	2013 Atkinson et. al., Component-based Product Line Engineering with UML, Addison-Wesley, 2001 Clements: Software Product Lines. Practices and Patterns. Northrop, 2002 van der Linden, F. J., Schmid, K. & E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer, 2007 Pohl, K., Böckle, G. & F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2010 Weiss, Lai: Software Product-Line Engineering. A Family-Based Software Development Process Addison-Wesley, 1999

V-M.2 – Software Concept Engineering

Module name:	**Software Concept Engineering**
Abbreviation:	Module No. V-M.2
Semester:	2nd (summer semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	Dr. Chr. Webel, I. Paech and Prof. Dr.-Ing. N. Wehn, Dr. P. Antonino de Assis, Dr. T. Kuhn, Dr.-Ing. M. Jung, Dr. A. Morgenstern
Language:	English
Classification within the curriculum:	This module is the second of the advanced modules. The first part of this module covers requirements engineering processes, requirements specification, analysis and negotiation. The second part provides an overview of the topic architecture including a clear motivation that shows why architectures are needed and what they are good for.
Teaching format / class hours per week during the semester (SWS):	Textbook, self-control assignments, online tutorial (15 weeks), participation in the Study Module (V.1) / 6
Workload:	Contact study workload: 57 hrs per semester[5] Self-study workload: 118 hrs per semester Overall workload: 175 hrs per semester
Credit points:	7
Recommended prerequisites:	Theoretical foundations: mathematical foundations, basic number domains, basic proof techniques, complexity-O-notation, set theory, propositional logics, first order logics and predicate logics System foundations: combinatorial and sequential circuits, performance of computers, functionality of single processor computers, memory hierarchies, caches, pipelining, bus hierarchies Software development foundations: basic programming techniques, elementary and abstract data types, elementary algorithms, algorithms and data structures for searching and sorting, ability to determine characteristics of algorithms, interaction between software modules and larger software systems, fundamentals in design patterns and elementary modeling techniques. Prerequisite tests: Passed exams after module B-M.1

[5] 10 consultations x 3 hrs = 30 hrs; 27 hrs during Study Module (Classroom phase V.1)

Targeted learning outcomes:	After studying the first part (Requirements Engineering) the students • understand the relevance of requirements engineering and consequences when requirements engineering is ignored, • know the basic terms of requirements engineering • use methods and techniques for requirements elicitation and summarize conflict resolution strategies • know the different types of specifications and templates for documenting requirements and related quality criteria, • are able to apply requirements solve conflicts and know conflict solving strategies techniques • are able to apply verification and validation techniques for requirements, • know the main quality criteria for requirements and requirements documents • are able to write good requirements, • understand the basics of systems engineering is and why it is important • explain the basic concepts of model-based requirements engineering • apply basic notations for documenting solution-oriented requirements by SysML or UML After studying the second part (Software Architectures) the students • have an overview of the topic architecture including a clear motivation that shows why architectures are needed and what they are good for, • understand what the basic principles of architecture are and how they can be leveraged in order to utilize architectures throughout the complete system engineering process, • have a consolidated view on the notion of architecture in the context of engineering software-intensive systems, • know the process of architecting in general, • know the different phases that architecture development comprises, • know the notions stakeholders and concerns as concepts describing the main sources for architectural drivers, • know what the means are that architects make use of in order to create architectures that meet stakeholders' concerns, • know documentation and architecture evaluation techniques that can be applied in order to validate an architecture against the stakeholder concerns originated from business goals, functional and quality requirements.
Content:	First part (Requirements Engineering): • Motivation (relevance of requirements in the system life-cycle and major challenges) • Foundations and Basic Terms (basic terms, problem vs. solution, system scope and context) • The requirements engineering (RE) process (main activities in the RE process, actors in the RE process, embedding the RE process in the overall software/system development process, interface to other process areas, RE and project management) • Requirements Elicitation (elicitation techniques, auxiliary techniques)

	• Quality Assurance of Requirements (importance of QA, quality criteria for requirements and requirements documents, requirements in natural language, quality assurance of requirements in natural language) • Requirements Negotiation (decisions, conflict management, win-win approach, documentation) • Requirements Verification and Validation (basic terms, importance of V&V, V6V techniques) • Requirements management (managing requirements, traceability, version control and configuration management, managing requirements changes, requirements status tracking) • Systems and Requirements Engineering (Systems Engineering Approach, characteristics of software-intensive Systems, SIMILAR, SysML) • Model-based Systems and Requirements Engineering (Use of models during development, goal modeling, requirements modeling, MBRE with UML/SysML) Second part (Software Architectures): • Architectural drivers (business goals, functional requirements, quality requirements, constraints) • Architecture development process • Stakeholder analyses (scoping of architectural context, organize stakeholder concerns) • Architectural views and viewpoints (typical architectural views, perspectives, view definition process) • Architecture realization (process overview, functional decomposition, scenario-driven design) • Architecture documentation (usage scenarios of architecture documentation, content of architecture documentation, representation of architecture documentation, creation of architecture documentation, architecture documentation in practice) • Architecture evaluation (architecture evaluation types, architecture evaluation purposes, architecture evaluation techniques, architecture evaluation audits)
Exam / Study achievements:	Written examination at the end of the semester
Forms of media:	Written lectures provided via internet (PDF)

Literature:	First part (Requirements Engineering): Pohl, K.: Requirements Engineering - Fundamentals, Principles, and Techniques, Springer, 2010 K. Pohl and Chris Rupp. Requirements Engineering Fundamentals. Rocky Nook, 2011. Wiegers, K. & J. Beatty: Software Requirements, Microsoft Press, 3rd edition, 2013 Robertson, S. & J. Robertson: Mastering the Requirements Process: Getting Requirements Right, Addison Wesley, 3rd revised edition, 2012 R. K. Kandt. Software Requirements Engineering: Practices and Techniques, Jet Propulsion Laboratory, California Institute of Technology, 2003. C. Rupp, S. Queins, and the Sophists. UML 2 crystal clear. Hanser, Munich, 4th revised edition, 2012. (In German) INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th Edition, Wiley, 2015 P. Checkland. Systems Thinking, Systems Practice, New York, NY, USA: John Wiley & Sons, 1999. S. Friedenthal, A. Moore and R. Steiner. A Practical Guide to SysML – The Systems Modeling Language. 3rd edition. MK/OMG Press. 2015 Second part (Software Architectures): M. Jeckle, C. Rupp, J. Hahn, B. Zengler, S. Queins: UML 2 glasklar. Hanser Fachbuchverlag; 1. Auflage, ISBN 3-44622-575-7, 2007 M. Fowler: UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition). Addison-Wesley Object Technology Series, ISBN 0-32119-368-7, 2003 T. Weilkins: Systems Engineering with SysML/UML: Modeling, Analysis, Design. Morgan Kaufmann, ISBN 0-12374-274-9, 2008

V-M.3 – Software Component Engineering

Module name:	**Software Component Engineering**
Abbreviation:	Module No.V-M.3
Semester:	3rd (winter semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	Prof. A. Poetzsch-Heffter and Dr. M. Trapp (authors of written lectures)
Language:	English
Classification within the curriculum:	This module is the third of the advanced modules. The first part of this module covers component-based software development and its specific approaches. The second part covers model-based development techniques for component-based development techniques.
Teaching format / class hours per week (SWS)	Textbook, self-control assignments, online tutorial (11 weeks) / 5
Workload:	Contact study workload: 36 hrs per semester[6] Self-study workload: 139 hrs per semester Overall workload: 175 hrs per semester
Credit points:	7
Recommended prerequisites:	Theoretical foundations: mathematical foundations, basic number domains, basic proof techniques, complexity-O-notation, set theory, propositional logics, first order logics and predicate logics System foundations: combinatorial and sequential circuits, performance of computers, functionality of single processor computers, memory hierarchies, caches, pipelining, bus hierarchies Software development foundations: basic programming techniques, elementary and abstract data types, elementary algorithms, algorithms and data structures for searching and sorting, ability to determine characteristics of algorithms, interaction between software modules and larger software systems, fundamentals in design patterns and elementary modeling techniques. Prerequisite modules: First semester of the Software Development Studio where the development of basic architectures is emphasized Prerequisite tests: Passed exams after modules B-M.1 and B-M.2.

[6] 12 consultations x 3 hrs = 36 hrs

Targeted learning outcomes:	After studying the first part (Component-based Software Development) the students • understand the role and important aspects of component-based software development in software engineering, • know similarities and differences regarding components-based software development approaches in general and for embedded systems, • understand the goals, structures, features, and services in component frameworks, • know important component frameworks for generic component based software development, and know how to apply them, • know important embedded systems specific component frameworks, such as AUTOSAR, • are able to develop a software system with components, • are able to apply existing component frameworks to embedded systems development. After studying the second part (Model-based Component Engineering) the students • know the principles of model-based development of embedded systems, • know the difference between components and other structural elements in software systems, • are able to decompose a system into subcomponents and are able to specify these subcomponents, • are able to realize components by means of behavior models, • understand the integration of discrete and continuous behavior modeling, • are able to customize generic purpose UML modeling for the embedded systems domain using UML profiles, • understand embedded systems specific modeling approaches like dataflow-oriented modeling, • understand the integration of object-oriented development and dataflow-oriented development approaches, • understand modeling of embedded platforms, modeling of runtime platforms, and tasking, • understand the tasks and responsibilities of platform software, • understand deployment modeling, • understand model transformations and validation.
Content:	First part (Component-based Software Development): • Introduction to component-based software development (CBSD) (goals of CBSD, approaching component-based software development, CBSD as a software engineering discipline) • Component software (software systems and component software, software components, component models, component infrastructure and framework support) • Developing component-based systems (an explanation of CBSD, CBSD and software engineering processes, software architecture as skeleton for reuse, language and tool support) • General component framework concepts (component frameworks: notions and core concepts, OSGi: a component framework for Java, other features of component frameworks) • Distribution and heterogeneity in component frameworks (distribution: problems and concepts, heterogeneity: multi-

	language development for systems running on different platforms, Common Object Request Broker Architecture (CORBA)) • Component frameworks for application servers (basic concepts of application servers, enterprise JavaBeans, dependency injection and bean composition) • Components for embedded software (requirements for embedded component software, frameworks for embedded software) • Review and current developments Second part (Model-based Component Engineering) • Fundamentals (engineering of embedded software systems, model-driven engineering, component engineering) • Component specification (modeling embedded systems with the UML, component interfaces and dependencies, type systems, the concept of views, functional views, non-functional views, modeling of non-functional aspects) • Compositional component realization (modeling component decompositions, modeling collaborations) • Behavioral component realization (control-flow oriented behavior modeling, data-flow oriented behavior modeling, hybrid modeling) • Platform integration (model deployment and platform integration overview, meta-models, model verification and transformation, platform modeling, platform-specific modeling, deployment and code generation, execution platforms and platform integration)
Exam / Study achievements:	Written examination at the end of the semester
Forms of media:	Written lectures provided via internet (PDF)
Literature:	First part (Component-based Software Development): Clemens Szyperski – Component Software: Beyond Object-Oriented Programming – 2nd Edition – Addison-Wesley / ACM Press, 2002 Second part (Model-based Component Engineering): M. Jeckle, C. Rupp, J. Hahn, B. Zengler, S. Queins: UML 2 glasklar. Hanser Fachbuchverlag; 1. Auflage, ISBN 3-44622-575-7, 2007. M. Fowler: UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition). Addison-Wesley Object Technology Series, ISBN 0-32119-368-7, 2003. T. Weilkins: Systems Engineering with SysML/UML: Modeling, Analysis, Design. Morgan Kaufmann, ISBN 0-12374-274-9, 2008. Noergaard, T.: Embedded Systems Architecture, Elsevier, Oxford, UK, 2005

V-M.4 – Embedded Software Engineering

Module name:	**Embedded Software Engineering**
Abbreviation:	Module No. V-M.4
Semester:	3rd (winter semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	Prof. H. Hansson, Prof. D. Isovic, Prof. M. Nolin, Dr. T. Nolte, Prof. C. Norström, Prof. P. Pettersson, Prof. S. Punnekkat and Dr. R. Adler (authors of written lectures)
Language:	English
Classification within the curriculum:	This module is the fourth of the advanced modules. The first part of this module covers the development of real time systems, and provides specific knowledge regarding schedulability, real-time constraints, real-time communications, and real-time platforms. It also introduces the specific properties of real-time systems compared to regular embedded systems. The second part covers important techniques in the domain of dependability engineering, fault types and tolerance, fault detection approaches, and the handling of faults.
Teaching format / class hours per week (SWS)	Textbook, self-control assignments, online tutorial (11 weeks), participation in the Studio Module (SM) / 6
Workload:	Contact study workload: 66 hrs per semester[7] Self-study workload: 109 hrs per semester Overall workload: 175 hrs per semester
Credit points:	7
Recommended prerequisites:	Theoretical foundations: mathematical foundations, basic number domains, basic proof techniques, complexity-O-notation, set theory, propositional logics, first order logics and predicate logics System foundations: combinatorial and sequential circuits, performance of computers, functionality of single processor computers, memory hierarchies, caches, pipelining, bus hierarchies Software development foundations: basic programming techniques, elementary and abstract data types, elementary algorithms, algorithms and data structures for searching and sorting, ability to determine characteristics of algorithms, interaction between software modules and larger software systems, fundamentals in design

[7] 12 consultations x 3 hrs = 36 hrs; 30 hrs during Studiy Module (Classroom phase V.2)

	patterns and elementary modeling techniques. Prerequisite subjects: Component-based Software Engineering, Software Product Line Engineering, first semester of the Software Development Studio where the development of basic architectures is emphasized Prerequisite tests: Passed exams after modules B-M.1 and B-M.2
Targeted learning outcomes:	After studying the first part (Realtime Systems) the students • understand the issues, concepts and main challenges of real-time systems, including design paradigms and architectures, as well as understanding the difference between real-time systems and traditional computer systems, • understand basic concepts in real-time operating systems and have an overall perspective of the various commercial operating systems and academic research kernels, • understand basic concepts of real-time scheduling and algorithms to schedule tasks in both event-triggered and time-triggered systems, • understand the different classes of real-time communication technologies existing today, both academic and commercial ones, including which application domains that these technologies are used in, • understand basic architectural styles and when they are used, as well as understanding why several styles are often used in the same system, • understand basic concepts used in programming and modeling languages for real-time systems, including understanding when and how modeling languages and real-time analysis can be used in the development of real-time systems. After studying the second part (Dependability Engineering) the students know • what dependability is and which attributes it subsumes, • the threats to dependability, • the means for handling threats to dependability, • why safety is an outstanding dependability attribute, • which approaches exist to achieve safety, • why the assurance of dependability requires new approaches if systems act autonomously and collaborate with each other.
Content:	First part (Realtime Systems): • Introduction (historical perspective, what is a real-time system?, close interaction with the environment, timing constraints, characteristics of real-time systems, classifications of real-time systems, example applications, some misconceptions about real-time systems, challenges for real-time systems) • Real-time operating systems (concurrent task execution, what is a real-time operating system?, RTOS characteristics, RTOS vs. GPOS, types of RTOSs, event-triggered real-time operating systems, time-triggered RTOSs, example commercial RTOSs) • Real-time scheduling (introduction to real-time scheduling, task model, schedulability and feasibility, hyperperiod, processor utilization factor, classification of scheduling algorithms, rate monotonic scheduling, deadline monotonic scheduling, response

	time analysis, static priority scheduling with shared resources, earliest deadline first, comparison between rate monotonic and earliest deadline first, offline scheduling) • Real-time networking (introduction, medium access control (MAC) protocols, networks, real-time networking: CAN – a concrete example, summary) • Modeling and analysis (issues and concepts, UML and MARTE, timed abstract state machine, timed automata and model-checking, summary) Second part (Dependability Engineering): • Understand the fundamental terms: function, behavior, service, (service) failure, dependability, availability, reliability, safety and risk, confidentiality, integrity, maintainability, security • Understand the threats to dependability and the terms fault, error, and failure; classification of faults, errors, and failures; “pathology of failures” and its application to an architecture with different levels of abstraction • Understand what fault avoidance, fault tolerance, fault removal, and fault forecasting are and how they relate to each other; the differences concerning the avoidance of development faults and the avoidance of operational faults; understand how different types of redundancy can be used to achieve fault tolerance; differences between fault removal during development and fault removal during use; fault forecasting and popular techniques for achieving the purposes of fault forecasting. • “safety” as outstanding dependability attribute; “safety process and project management”, “safety related-product engineering”, and “safety engineering”; hazard and risk assessment; integrated model-based safety engineering; Component Fault Trees; safety concept and a safety case; GSN; autonomous cooperative systems and dependability Furthermore, the second part of the module covers the following aspects: • Introduction (elements of dependability, role of the software engineer, our dependence on computers, consequences of failure, need for dependability, systems and their dependability requirements) • Dependability requirements (why we need dependability requirements, role of terminology, evolution of dependability concepts, requirements and specification, failure, dependability and its attributes, systems, software and dependability, defining dependability requirements, as low as is reasonably practicable – ALARP) • Errors, faults and hazards (errors, complexity of erroneous states, faults and dependability, manifestation of faults, degradation faults, design faults, Byzantine faults, component failure semantics, fundamental principle of dependability, anticipated faults, hazards, engineering dependable systems) • Dependability analysis (fault tree analysis, failure modes, effects and criticality analysis, hazard and operability analysis) • Dealing with faults (faults and their treatment, fault avoidance, fault elimination, fault tolerance, fault forecasting) • Degradation faults and software (redundancy, redundant architectures, quantifying the benefits of redundancy) • Software dependability (faults and the software lifecycle, correctness by construction, formal techniques, model checking, model-based development, software fault avoidance, software fault eli-

	mination, managing software fault avoidance and elimination) • Software fault avoidance in specification (role of specification, difficulties with natural languages, specification difficulties, formal languages, model-based specification, the declarative language Z, a simple example, a detailed example, summary of formal specification development) • Software fault avoidance in implementation (implementing software, programming languages, programming standards, implementation correctness by construction) • Software fault elimination (why fault elimination?, inspection, testing) • Software fault tolerance (components subject to design faults, issues with design fault tolerance, software replication, design diversity, data diversity, targeted fault tolerance) • Dependability assessment (approaches to assessment, quantitative assessment, prescriptive standards, rigorous arguments, regulation based on safety cases, applicability of argumentation) • Exercise scenario: automobile braking system
Exam / Study achievements:	Written examination at the end of the semester
Forms of media:	Written lectures provided via internet (PDF)
Literature:	First part (Realtime Systems): M. Barr & A. Massa: Programming Embedded Systems – with C and GNU Development Tools, 2nd edition. O'Reilly Media, 2006. B. P. Douglass: Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison-Wesley, 1999 H. Kopetz: Real-Time Systems, Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997 Q. Li & C. Yao: Real-Time Concepts for Embedded Systems. Elsevier, CPM Books, 2003 Second part (Dependability Engineering): Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. on Dependable and Secure Computing, Vol 1, No. 1, 2004 Kececioglu D., Reliability Engineering Handbook, Prentice-Hall, 1991 Knight, J., Fundamentals of Dependable Computing for Software Engineers, Crc Pr Inc, 2012 Laprie, J.-C. (Editor), Dependability: Basic Concepts and Terminology, Springer-Verlag, 2013 Lyu M.R., Handbook of Software Reliability Engineering, McGraw-Hill, New York, 1995 Petre, L., Sere, K., Troubitsyna, E. (Editors), Dependability and Computer Engineering: Concepts for Software-Intensive Systems, Engineering Science Reference, 2012

C. Studio Modules and Master Thesis

SM – Software Development Studio

Module name:	**Software Development Studio**
Abbreviation:	Module No. SM
Semester:	2nd and 3rd (summer and winter semester)
Module coordinator:	Prof. Dr. Dr. h.c. Dieter Rombach
Lecturer:	-
Language:	English
Classification within the curriculum:	The project work is mandatory for all students.
Teaching format / class hours per week during the semester (SWS):	The students join a team of about two to five students to develop software. The projects are mentored by experts. / 6 The software development studio takes place in two parts: The first takes five days at the end of the 2nd semester and the second part takes five days at the end of the 3rd semester.
Workload:	Overall workload 150 hrs
Credit points:	6
Recommended prerequisites:	Prerequisites: Solving the mail-in exercises before starting each of both parts is mandatory. Basic prerequisites: Theoretical foundations, system foundations and software development foundations as listed in table M 1 Prerequisite modules: The tasks of the studio correspond to the current topics taught by the modules
Targeted learning outcomes:	The students • know how stick to predefined processes and procedures while considering issues such as effort, time, and quality, • know the necessary steps to develop a system from on a set of requirements until the delivery of the final system, • are able to perform quality assurance activities, • are able to use selected methods, techniques, and tools for the specification, design, and implementation of a software system, • are able to document and interpret software engineering artifacts of all the abstraction levels, • are able to present, explain, and discuss (intermediate) results to other persons, • are able to make decisions, to critically evaluate facts and circumstances in a team.

Content:	Contents of part 1: • Stakeholders and System's scope • Requirements elicitation & scenario derivation • Requirements specification • Architectural views & scenario derivation • Architecture decomposition framework: applying the ADF • Architecture decomposition framework: capturing architectural design using the enterprise architect • Architecture evaluation Contents of part 2: • ACC Requirements • Simulink introduction • Embedded systems and code generation • Microcontrollers • ACC development • Testing and validation of requirements
Exam / Study achievements:	Oral examination of 30 minutes after each of the two parts
Forms of media:	-
Literature:	According to modules

MT – Master Thesis

Module name:	**Master Thesis**
Abbreviation:	Module No. MT
Semester:	4th (summer semester)
Module coordinator:	The student must find a professor, who supervises the master thesis. If the master thesis will be done in an institution other than the University of Kaiserslautern, the agreement of the chairperson of the examination board is required.
Lecturer:	-
Language:	English
Classification within the curriculum:	The master thesis is mandatory for all students.
Teaching format / class hours per week during the semester (SWS):	Self-studies or project work / 15
Workload:	Overall workload: 375 hrs, distributed over 6 months
Credit points:	15
Prerequisites:	If the student has not yet passed all examinations of the distance study program, he must have passed at least the exams of the modules B-M.1, B-M.2 and of two of the modules V-M.1, V-M.2, V-M.3 or V-M.4 and he must have participated in the on-campus weekends at the end of the 2nd and 3rd semesters.
Targeted learning outcomes:	Student shows his/her ability to work autonomously to a large extent under direction of a professor or assistant.
Content:	Engineering project of manageable size.
Cooperation (international or industry):	Working on an industry project under supervision of a professor is possible.
Exam / Study achievements:	Written documentation on project planning, work and output. To check that the Master's thesis was written by the student on his/her own, a final colloquium is held about the thesis with at least one of the supervisors/ examiners.
Forms of media:	-

Literature:	**-**