KNOWLEDGE FOUNDATION
Reutlingen University

INF
Informatik

MODULE MANUAL

Professional Software Engineering, MSc.

Master's programme (according to §33 LHG BaWü)

Reutlingen University Faculty
of Computer Science
Alteburgstraße 150
72762 Reutlingen
https://www.inf.reutlingen-university.de/home/

Status: 01/08/2024

Table of contents

BRIEF DESCRIPTION AND OBJECTIVES OF THE STUDY PROGRAMME 3

MODULE OVERVIEW PROFESSIONAL SOFTWARE ENGINEERING (MSC.) 4

M 1 METHODS AND TECHNOLOGIES OF PROFESSIONAL PROGRAMMING 5

M 2 SOFTWARE ENGINEERING 9

M 3 DATABASE SYSTEMS 13

M 4 CLOUD COMPUTING 16

M 5 FRONT-END DEVELOPMENT 19

M 6 BACKEND DEVELOPMENT 23

M 7 SOFTWARE ARCHITECTURE 26

M 8 SOFTWARE PROJECT 1 30

M 9 SOFTWARE PROJECT 2 32

M 10/M 11 ELECTIVE MODULE 1 AND 2 34

W1 Distributed Ledger Technology 34

W2 Digital Product Management 37

W3 Internet of Things 40

M 12 MASTER THESIS 43

Brief description and objectives of the study programme

Agile methods, DevOps, microservices and cloud computing have changed the job profile of software developers. Specialised knowledge, skills and competencies are required to develop applications for the cloud or to migrate to it. In order to truly utilise the advantages of the cloud, applications must be programmed in such a way that they can be continuously revised, tested, built, automated in the cloud and monitored. The strict separation of development and operation no longer exists - "you build it, you run it" (DevOps). This enables new software architectures, such as microservices or serverless computing, which in turn place new demands on the design and development of applications.

Graduates of the degree programme develop software using modern methods and tools, especially for modern cloud environments.

In addition to the necessary technical and methodological knowledge, skills and competences, graduates should be taught the following values and principles:

- We work in an agile manner and place customer benefit at the centre of our work.
- We emphasise quality in software engineering and apply corresponding principles.
- We are committed to responsible, ethical behaviour and respect users' right to data protection.

An overview and detailed description of the modules can be found in the following sections.

Module overview Professional Software Engineering (MSc.)

<table>
<tr><th>Semester</th><th colspan="30">Master of Science degree</th></tr>
<tr><td>4</td><td colspan="30">Master thesis</td></tr>
<tr><td>3</td><td colspan="5">Compulsory elective module 1</td><td colspan="5">Compulsory elective module 2</td><td colspan="10">Software project 2</td><td colspan="10" rowspan="3">Research paper, practical project or professional practice (credit for pre-study programme < 210 ECTS)</td></tr>
<tr><td>2</td><td colspan="5">Front-end development</td><td colspan="5">Backend development</td><td colspan="7">Software architecture</td><td colspan="3">Software project 1</td></tr>
<tr><td>1</td><td colspan="5">Methods and technologies of professional programming</td><td colspan="5">Software Engineering</td><td colspan="7">Database systems</td><td colspan="3">Cloud computing</td></tr>
<tr><td>ECTS</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td></tr>
</table>

1 ECTS means approx. 30 hours of work (attendance and personal contribution)

M 1 Methods and technologies of professional programming

Module:	Methods and technologies of professional programming
Abbreviation:	M 1
Subtitle:	
Courses:	Lecture
Module coordinator:	Prof Dr Christian Kücherer
Lecturer(s):	Prof. Dr Christian Kücherer Mr Paul Lajer Dr Robin Braun
Language:	German
Assignment to the curriculum:	Professional Software Engineering, Master, compulsory subject, 1st semester
Teaching form / SWS:	Lecture, block course / 4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	None
Study/examination achievements/form of examination:	Project work

Module objectives:

This course raises all participants of the Master's degree programme to a common level of knowledge and expertise with regard to their programming skills and the tool chain required for modern software development. The course highlights the links to the Software Engineering course. In particular, participants will learn what is important when developing source code and how the many different tools can be used throughout the entire development cycle. In addition to the development of maintainable and comprehensible code with the principles of clean code and continuous refactoring, the course also covers the use of test frameworks to ensure software quality through to the complete infrastructure for the continuous delivery of software with Continuous Integration and Continuous Deployment (CI/CD).

Intended learning outcomes

Knowledge:

Participants in this module learn how current infrastructures and the tools used in them can be used effectively and efficiently in current software development projects. The main focus is on the methodological and conceptual foundations of the tools on the one hand, and on the practical application of the tools on the other. The key skills that are taught are Principles for object-oriented development, development of high-quality source code, versioning strategies, use of test tools at different levels of abstraction, continuous delivery and continuous deployment, use of software metrics and static code analysis, code documentation, collaborative software development and development environments. Various branching strategies, feature toggles, tagging and merging strategies are considered in the versioning and configuration of source code.

Skills:

The participants

- Create and manage the requirements for a software system with the help of requirements management tools and transfer them step by step into an executable system using an agile approach.
- Create component-based software systems independently, taking into account the principles of classic design patterns.
- Apply the principles of clean code and clean architecture to create maintainable code.
- Manage created code with the help of version management systems and further develop the code together in teams.
- Check created code functionally and non-functionally using test frameworks.
- Document created code.
- Apply the basics of architectural patterns practically in projects.
- Deliver software systems continuously as executable applications with the help of continuous integration, continuous delivery and continuous deployment, using the necessary infrastructure.

Expertise:

After completing the module, participants will be able to

LE#	Learning outcome (LE)	Tested by
LE1	To create stable, high-quality and maintainable source code, taking into account design patterns, object-orientated design and the SOLID principles.	Assessment task
LE2	Create the basic architecture of software systems and develop software using suitable patterns.	Assessment task
LE3	Apply the principles of clean code and clean architecture and document software in a meaningful way	Assessment task
LE4	Practical application of CI / CD methods, selection of suitable tools for the problem and documentation of the entire deployment pipeline.	Term paper and presentation
LE5	quality assurance with test frameworks and static analysis using software metrics.	Assessment task
LE6	development environments effectively and to solve the problems of collaborative software development by using suitable tools and methods.	Lecture-accompanying exercise
LE7	Use versioning strategies and tools sensibly in projects.	Lecture-accompanying exercise

Contents:

At the beginning of the course, the participants deal with the basic principles of modern software development and apply software design patterns in order to structure their object-oriented systems and provide them with functionality based on established standards (EU1). This results in stable, high-quality and maintainable source code that simplifies the evolution of the system. Based on defined requirements, participants create the basic architecture of a software system and use suitable pat- terns (LO2). As a further theoretical module, the principles of clean code and clean architecture are taught, which participants apply to a practical example. This is not only about improving the quality of source code and the architecture that provides the structure, but also about documenting the software code in a meaningful way (LO3). The state of the art is the continuous compilation and integration of the components of a software system in order to prevent a creeping deterioration of the code. To this end, tool chains that realise this deployment pipeline are selected by the participants and assembled into a meaningful whole (EU4). Regression tests are used to check the freedom from side effects of changes in order to ensure the continuous functionality of the overall system. Automated test frameworks are used for this purpose, which participants select and use using a practical example (LO5). With the help of

of collaboration and version management tools, participants learn to recognise the problems of distributed software development and to take appropriate measures (LO6). Finally, the basics of different versioning approaches and tools are taught so that parallel development on the same code base by many developers is possible (LO7).

Media forms:

The teaching material consists of a slide script, which is available in electronic form, exercise sheets and programme examples. Seminar-style teaching with whiteboard, PC projector and presentation slides, in which examples of the theoretical content are illustrated as well as demonstrations of sample programmes and interactive programme development. Participants work individually or in groups on exercises on the topic of modern software development under the supervision of lecturers. Another element is the development by participants as an inverted classroom: participants have to work on a sub-area of the complete tool infrastructure based on a defined problem. The knowledge gained is then passed on to the other participants in a presentation with a 90-minute workshop.

Literature:

- Alur, Deepak; Crupi, John; Malks, Dan (2003): Core J2EE patterns: best practices and de- sign strategies, 2nd edition, Upper Saddle River, NJ: Prentice Hall PTR; Palo Alto, Calif., Sun Microsystems Press.
- Duvall, Paul M.; Matyas, Steve and Glover, Andrew (2010): Continuous integration: improv- ing software quality and reducing risk, 5th edition, Upper Saddle River, NJ; Munich [u.a.] : Addison-Wesley
- Fowler, Martin (2000): Refactoring: how to improve the design of existing software. Munich [et al.], Addison Wesley,
- Fowler, Martin (2008): Patterns of enterprise application architecture, 14th edition. Boston, Mass.; Munich [et al.], Addison-Wesley
- Gamma, Erich (2005): Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software, Munich [u.a.], Addison-Wesley
- Humble, Jez and Farley, David (2011): Continuous delivery: reliable software releases through build, test, and deployment automation, 1st edition, Upper Saddle River, NJ ; Mu- nich: Addison-Wesley
- Link, Johannes; Fröhlich, Peter (2002): Unit Tests with Java: the test-first approach, 1st edition, Heidelberg, dpunkt
- Loeliger, Jon (2009): Version Control with Git: Powerful tools and techniques for collabora- tive software development, O'Reilly Media.
- Martin, Robert C. (2009): Clean code: Refactoring, patterns, testing and techniques for clean code, 1st edition, Heidelberg; Munich; Landsberg; Frechen; Hamburg, mitp
- Martin, Robert C. (2014): Clean Coder: Rules of behaviour for professional programmers, 1st edition Heidelberg, mitp.
- Tabaka, Jean (2006): Collaboration Explained: Facilitation Skills for Collaborative Leaders. Agile Software Development Series, Addison Wesley.

M 2 Software Engineering

Module:	Software Engineering
Abbreviation:	M 2
Subtitle:	
Courses:	Lecture
Module coordinator:	Prof Dr Christian Decker
Lecturer(s):	Prof. Dr Christian Decker Prof. Dr Christian Kücherer Dominik Neumann
Language:	German
Assignment to the curriculum:	Professional Software Engineering Master, compulsory subject, 1st semester
Teaching form / SWS:	Lecture / 4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	None
Study/examination achievements/form of examination:	Term paper, presentation

Module objectives:

Targeted and systematic management of software development is crucial to the success of a software project.

The module introduces the engineering approach to software engineering along the software life cycle, the associated processes and process models. One focus is on agile process models. Requirements engineering and the connections to the other phases of the software life cycle play a central role in the module. Requirements, which are systematically collected and documented from and with stakeholders, form the basis for the scope and function of the system to be created. The resulting specification describes the expected functionality and is the basis for quality assurance, e.g. testing and test processes. Other forms of quality assurance include organisational measures such as the introduction of regular reviews or the implementation of the V-model of testing. The system design and software modelling activities are influenced by this.

Finally, the module teaches the basics of strategic Domain Driven Design (DDD) in order to create an important foundation for the development and understanding of modern and frequently used approaches to software architectures for innovative software products. The application of this approach is introduced to the module participants through a practical simulation of a customer-oriented environment. With the strategic Domain Driven Design, the Software Engineering module also distinguishes itself from the tactical Domain Driven Design of the Software Architecture module.

Intended learning outcomes

Knowledge:

Participants acquire knowledge and experience of engineering methods in the creation and implementation of software projects and knowledge of different systematic procedures in the development of software. They know about important current standards and procedures for defining the requirements of a system, such as common elicitation techniques and forms of documentation for software specifications. For implementation, the module provides participants with know-how in strategic DDD and system design. They learn the methods of event storming, context mapping and ubiquitous language. Finally, participants are trained in the practices of end-to-end quality assurance. This includes software quality assurance through test-driven methods and organisational methods such as the V-model of testing and fully includes documentation artefacts.

Skills:

In this module, participants acquire the skills of a software engineer and can apply them. This means in detail: Selecting suitable process models, adapting them if necessary and then proceeding in a targeted manner in software development; specifying and documenting software in a systematic and comprehensible process and ensuring quality assurance; creating a suitable system design using the practices and methods of strategic DDD for customer-specific software projects and making it feasible.

Expertise:

Participants are able to successfully specify and design complex software projects. They qualify to take responsibility for software development and to continue to develop successful and realisable software projects. In the module, participants gain a competent grounding in the development of modern and frequently used approaches to software solutions in innovative software products, for example with cloud-native application designs and the use of microservice architectures. Participants gain an understanding of the conflicting priorities of ethics in software engineering and learn how to apply ethical principles.

LE#	Learning outcome (LE)	Tested by
LE1	Processes and procedure models of plan-driven and agile software development	Term paper
LE2	Requirements and requirements engineering process, elicitation techniques and documentation, SW quality assurance, organisational quality assurance and static testing, e.g. reviews	Term paper
LE3	Use cases, system design and definition of system boundaries; test specifications, test-driven development (TDD); V model of testing	Term paper
LE4	Strategic domain-driven design (DDD), identification of specialised domain intersections (bounded context); methods and practices: event storming, context mapping, ubiquitous language	Unit
LE5	Simulation of DDD practices in a customer-facing environment	Unit

Contents:

- Processes and procedure models of software engineering
- Agile vs. plan-driven processes
- Requirements and requirements engineering process
- Software quality assurance, organisational quality assurance and static testing, e.g. reviews
- Test specifications, test-driven development (TDD), V model of testing
- Specify survey techniques and quality requirements
- Documentation standards and their quality assurance
- Specification of use cases and system boundaries
- Strategic Domain Driven Design (DDD) and identification of bounded context
- Methods and techniques of strategic DDD: Event Storming, Context Mapping, Ubiquitous Language
- Simulation of strategic DDD practices in a customer-facing environment
- Ethics in software engineering based on current ethical guidelines for software development

Exercises are carried out in changing team compositions. The aim is to deepen the content and open up different perspectives.

Media forms:

Lecture, exercises, script with the PDF of the lecture slides, exemplary publications, simulation of a customer-oriented environment

Literature:

- Sommerville, Ian. Software Engineering, Pearson Studium; Edition: 9. Aktual. (1 March 2012), ISBN-10: 3868940995
- Patton, Jeff. User Story Mapping- Understanding User Needs as the Key to Successful Products , O'Reilly; Edition: 1 (30 April 2015) ISBN-10: 3958750672
- Ludewig, J., Lichter H.: Software Engineering - Grundlagen, Menschen, Prozesse, Techniken. dpunkt.verlag Heidelberg, 2007. ISBN 3-89864-268-2
- Klaus Pohl, Chris Rupp: Basiswissen Requirements Engineering: IREB Foundation Level, dpunkt.verlag Heidelberg
- Lauesen S: Software Requirements: Styles & Techniques: Styles and Techniques, 2002, Addison-Wesley, ISBN 978-0201745702
- Spillner A, Linz T: Basiswissen Softwaretest: Education and Training for Certified Testers - Foundation Level according to ISTQB Standard, 5th edition 2012, dpunkt.verlag GmbH; ISBN 978- 3864900242
- Vernon, Vaughn, Domain-Driven Design kompakt, (German translation by C. Lilienthal), dpunkt Verlag 2017, ISBN: 978-3864904394

Further literature will be announced during the lecture.

M 3 Database systems

Module:	Database systems
Abbreviation:	M 3
Subtitle:	
Courses:	Lecture
Module coordinator:	Prof Dr Peter Hertkorn
Lecturer(s):	Prof Dr Peter Hertkorn
Language:	German
Assignment to the curriculum:	Professional Software Engineering Master, compulsory subject, 1st semester
Teaching form / SWS:	Lecture, block course / 4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	None
Study/examination achievements/form of examination:	Project work

Module objectives:

Participants acquire knowledge about the functionality of modern database systems and different database technologies. They understand the underlying principles, methods and techniques and are able to apply their theoretical knowledge in practice. In the further course of the programme, successful completion of the module should ensure that participants are able to select suitable database technologies for given problems and to create and use the various database systems with the help of database and programming languages.

Intended learning outcomes

Knowledge:

The participants

- know methods for accessing a database from an application programme.
- are able to describe methods for migrating and versioning data.
- know methods for storing semi-structured data.
- can describe concepts of recent developments in database systems and explain the differences to relational database systems.
- are able to explain the properties of distributed systems with regard to database systems.
- know the properties of document-orientated databases and can describe and evaluate them.
- can explain the properties of key value databases.
- are able to explain the properties of graph databases.
- know the properties of column-orientated databases.
- can describe and evaluate different methods for scaling database systems.

Skills:

Participants formulate queries and changes to relational databases from an application programme. They apply methods for migrating and versioning data. Participants create database schemas and queries for semi-structured data using the XML extensions of the relational model. They analyse the requirements for given problems and select a suitable form of NoSQL database. Participants create data models for the different types of NoSQL databases and create queries and changes to the NoSQL databases using suitable database languages. Participants apply methods for scaling database systems.

Expertise:

After completing the module, participants will be able to

LE#	Learning outcome (LE)	Tested by
LE1	Methods for accessing a database from an application application programme.	Artefact
LE2	Methods for the migration and versioning of data turn to.	Artefact
LE3	Modelling alternatives when creating the database to evaluate the results.	Artefact
LE4	Databases for different data models with data bank languages.	Artefact
LE5	To formulate queries to the database for given requirements and to develop alternative options for queries to the database. evaluate the database and assess its performance.	Artefact

LE6	Evaluate different database technologies for a given use case and select a suitable database. technology.	Artefact
LE7	Current developments in the field of database systems to judge and appropriate.	Artefact

Contents:

In the lecture, the knowledge of the relational model is deepened (LE1) and extended by the use of methods for migration and versioning of data (LE2). In addition to classic relational database systems, the XML extensions of the relational as well as the different types of NoSQL databases are covered in detail and their properties are compared with those of relational database systems (EU3-6). Access to databases from an application is presented and explained using example programmes (EU1). Furthermore, recent developments in the field of database systems are presented and their properties are compared with those of the database systems covered (LO7). In the practical implementation, care is taken to ensure that tools used in industry are used so that practical knowledge is also acquired.

Media forms:

The teaching material consists of a slide script, which is available in electronic form, exercise sheets and programme examples. Seminar-style teaching with whiteboard, PC projector and presentation slides, in which examples of the theoretical content are illustrated as well as demonstrations of sample programmes and interactive programme development. Participants work individually or in groups on exercises on the subject of database systems under the supervision of the lecturer.

Literature:

- Ambler, Scott W. and Pramodkumar J. Sadalage (2011): Refactoring Databases: Evolution- ary Database Design. Addison-Wesley.
- Edlich, Stefan and Achim Friedland (2011): NoSQL: Getting started in the world of non-relational Web 2.0 databases. 2nd edition. Hanser.
- Fowler, Adam (2015). NoSQL for Dummies. Dummies Tech.
- Harrison, Guy (2015): Next Generation Databases: NoSQL, NewSQL, and Big Data. Apress.
- McCreary, Dan and Ann Kelly (2013): Making Sense of NoSQL: A Guide for Managers and the Rest of Us. Manning.
- Perkins, Luc et al (2018): Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. 2nd edition. O'Reilly.
- Robinson, Ian et al (2015): Graph Databases: New Opportunities for Connected Data. 2. Edition. O'Reilly.
- Sadalage, Pramod J. and Martin Fowler (2012): NoSQL Distilled: A Brief Guide to the Emerg- ing World of Polyglot Persistence. Addison-Wesley.
- Sullivan, Dan (2015): NoSQL for Mere Mortals. Addison-Wesley.
- Trelle, Tobias (2014): MongoDB: The practical introduction. Dpunkt.
- Vonhoegen, Helmut (2018): XML: Getting started, practice, reference. 9th edition. Rheinwerk Compu- ting.

M 4 Cloud computing

Module:	Cloud computing
Abbreviation	M 4
Courses:	Lecture
Module coordinator:	Prof Dr Marcus Schöller
Lecturer(s):	Prof Dr Marcus Schöller Prof Dr Wolfgang Blochinger André Lindenberg
Language:	German / English
Assignment to the curriculum:	Professional Software Development Master, compulsory subject, 1st semester
Teaching form/SWS:	Lecture / 4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	none
Recommended prerequisite:	
Study/examination achievements/ form of examination:	Project work

Module objectives:

Cloud computing is now a central component of corporate IT and also enables a variety of new digital business models and digital products. In this module, participants will be familiarised with key aspects of cloud computing. In particular, they should acquire comprehensive knowledge of the design, development and operation of distributed cloud services and applications.

Intended learning outcomes:

Knowledge:
After successfully completing this module, participants will have knowledge of the key principles and characteristics of cloud computing. They know the basic characteristics of cloud services and their delivery models and have developed an understanding of technical, organisational, commercial and security-relevant aspects of cloud computing. They have in-depth knowledge of cloud architecture patterns, programming tools and platforms and their areas of application.

The participants have a good understanding of the fundamental laws, regulations and strategies in data protection.

Skills:
Participants will be able to assess typical service and delivery models with regard to case studies. They will be able to analyse the requirements of (server) services and develop and evaluate suitable deployment variants. These variants range from in-house server solutions to hybrid cloud models and pure cloud solutions. To do this, they apply a range of methods they have learnt. Based on these requirements, participants will be able to design and develop services that utilise the characteristics of the cloud. Furthermore, participants will be familiar with the details of cloud environments, which will enable them to compare the different deployment variants in greater depth. Participants will therefore be able to analyse and evaluate the solutions holistically and thus make technically sound decisions for service provision. They will also be able to analyse cloud applications with regard to data protection aspects so that they can take the appropriate (technical) measures when designing and implementing these applications.

Expertise:

LE#	Learning outcome	Tested by
LE1	Have an understanding of the different cloud busi- ness models (IaaS, PaaS, SaaS) and be able to apply them	Project work
LE2	Relate components and their tasks in a cloud architecture	Project work
LE3	Understand and evaluate operational aspects of a cloud infrastructure	Project work
LE4	Understand and apply software development methods for a cloud platform nes	Project work
LE5	be able to analyse cloud applications with regard to legal data protection aspects	Project work

Contents:

The service models Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are presented from both the provider's and the user's perspective. The focus here is on the topics of software design and development for the cloud as well as a basic understanding of cloud environments. The public cloud, private cloud, hybrid cloud and community cloud delivery models are explained using case studies. Technical, organisational, commercial and security-related aspects of cloud computing are covered, evaluated and discussed in detail. Data protection concepts and regulations are presented and organisational and technical measures for the protection of personal data are discussed.

Media forms:

Seminar-style lecture, slides and blackboard notes; case studies in small groups.

Literature:

- Antonopoulos, Nick; Gillam, Lee (2010): Cloud Computing. Principles Systems and Ap- plications. London: Springer London (SpringerLink: Books, 0).
- Baun, Christian; Kunze, Marcel; Nimis, Jens; Tai, Stefan (2011): Cloud Computing. Web-based dynamic IT services. Berlin, Heidelberg: Springer Berlin Heidelberg (SpringerLink : Books).
- Buyya, Rajkumar (2011): Cloud computing. Principles and paradigms. Hoboken, NJ: Wiley (Wiley series on parallel and distributed computing).
- Fehling, Christoph; Leymann, Frank; Retter, Ralph; Schupeck, Walter; Arbitter, Peter (2014): Cloud Computing Patterns. Fundamentals to Design, Build, and Manage Cloud Applications. Vienna: Springer.

M 5 Front-end development

Module:	Front-end development
Abbreviation:	M 5
Subtitle:	
Courses:	
Module coordinator:	Prof Dr Natividad Martínez Madrid
Lecturer(s):	Prof. Dr Natividad Martínez Madrid Prof. Dr Peter Hertkorn Ralf Kretzschmar-Auer Matthias Gutbrod
Language:	German
Assignment to the curriculum:	Professional Software Engineering Master, compulsory subject, 2nd semester
Teaching form / SWS:	Lecture4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	Methods and technologies of professional programming
Study/examination achievements/form of examination:	Project work

Module objectives:

The module addresses the topic of user-centred front-end development, both from a methodological and a technical perspective. Students gain knowledge of the basic steps involved in designing a frontend. They understand what good interaction between frontend and backend looks like, can design frontend architectures and apply the relevant technologies and frameworks. Students design mobile frontends and familiarise themselves with the characteristics of mobile platforms. After successfully completing the module, students will be able to design a suitable frontend for given problems and develop it with the help of different technologies.

Intended learning outcomes

Knowledge:

- Students know the basics of communication psychology for UX design: visualisation, colours/images/animations, behaviour, target groups, user guidance, error tolerance, feedback to the user.
- Students are able to describe the characteristics of barrier-free interactions.
- Students are familiar with conceptual tools of UX design: context navigation maps, screen flows, paper prototypes.
- Students can explain the basics of responsive design / mobile-first approach.
- Students know the basics of component-based design in the front end and can describe the methods of interaction between the front end and back end.
- Students know the basics of programming front ends based on web technologies.
- Students are able to describe and evaluate the characteristics of modern frameworks for front-end development.
- Students are familiar with various design patterns that are used in the development of front ends.
- Students know the characteristics of mobile devices as front-end platforms.
- Students can explain and evaluate the differences between native and web-based/cross-platform mobile frameworks.
- Students are able to describe the special features of front-end development on native frameworks.

Skills:

- Students can develop a concept for a given task.
- The students apply learnt UX techniques in the prototyping of front ends (desktop/mobile device).
- Students can implement a UX design prototype: Design interactions between frontend and backend, perform error handling, mock up the backend if necessary and put the software into operation on a server.
- Students create front ends based on web technologies.
- Students use suitable frameworks for front-end development.
- Students analyse the requirements for given problems and select suitable technologies for implementation.
- Students apply design patterns in the development of front ends.
- Students compare different mobile native or cross-platform frameworks under certain requirements.
- Students will be able to implement a mobile front end for given tasks on the basis of cross-platform and native platforms.

Expertise:

After completing the module, students will be able to

LE#	Learning outcome (LE)	Tested by
LE1	be able to apply methods for modelling UX designs.	Project work
LE2	UX-based prototypes of front ends.	Project work
LE3	frontends with the help of web technologies.	Project work
LE4	alternatives in the design and realisation of components.	Project work
LE5	frameworks for front-end development.	Project work
LE6	frameworks for cross-platform mobile front-end development.	Project work
LE7	aspects of native mobile frontends.	Project work

Contents:

In the lecture, the basics of user-centred design (UX design) are explained through an introduction to communication psychology. Different methods for user-centred modelling of frontends are addressed and implemented by the students in a concrete example (LO1). Students then learn techniques for prototyping frontends in interaction with (possibly mocked-up) backends. The modelled UX frontend is then implemented as a prototype (LO2). Different web technologies for the development of frontends are discussed on the basis of various problems (LO3). Alternatives are analysed and evaluated during the design process and proven solutions are discussed (LO4). The students then apply the knowledge they have acquired independently when solving exercises and creating frontends with the help of suitable frameworks (LO5). From the outset, students are familiarised with the development of mobile front-ends using the responsive design/mobile first approach. Students also learn about the possibilities of integrating the functionality and properties of the native framework of the end device using the cross-platform approach (e.g. React Native) (LO6). You will analyse and evaluate the advantages and disadvantages of web-based frontends versus native mobile frontends. You will then cover the basics of native mobile programming (e.g. Android) by analysing examples from selected subject areas (LO7).

Media forms:

The teaching material consists of a slide script, which is available in electronic form, videos, examples of signatures and programmes and practical tasks. The seminar-style teaching with whiteboard, PC projector and presentation slides, in which examples of the theoretical content are illustrated, is combined with practical parts and demonstrations of sample programmes as well as interactive design and programme development. Students work on tasks individually or in groups with supervision from the lecturer.

Literature:

- Banks, Alex and Eve Porcello (2017): Learning React: Functional Web Development with React and Flux. O'Reilly.
- Beyer, Hugh and Karen Holtzblatt (1998): Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann Publishers.
- Constantine, Larry and Lucy Lockwood (2004): Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design. 4th edition. ACM Press.
- Flanagan, David (2011): JavaScript: The Definitive Guide. 6th edition. O'Reilly.
- Godbolt, Micah (2016): Frontend Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable Websites. O'Reilly.
- Heimann, Monika and Michale Schütz (2017): How design works: Principles of successful design. Rheinwerk publishing house.
- Krug, Steve (2014): Don't make me think! Web & Mobile Usability: The intuitive web. mitp Verlag.
- Nielsen, Jakob (2001): Designing Web Usability. New Riders Publishing
- Philips, Bill et al (2017): Android Programming: The Big Nerd Ranch Guide. 3rd edition. Pearson Technology Group.
- Shneiderman, Ben et al. (2017): Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th edition. Pearson.
- Tidwell, Jenifer (2010): Designing Interfaces: Patterns for Effective Interaction Design. 2. Edition. O'Reilly.
- Vollmer, Guy (2017): Mobile App Engineering. dpunkt Verlag.
  Wolf, Jürgen (2016): HTML5 and CSS3: The comprehensive handbook for learning and reference. 2nd edition. Rheinwerk Verlag.

M 6 Backend development

Module:	Backend development
Abbreviation:	M 6
Subtitle:	Compulsory elective module
Module coordinator:	Prof Dr Martin Schmollinger
Lecturer(s):	Mr Marcus Schießer Prof Dr Martin Schmollinger
Language:	German
Assignment to the curriculum:	Compulsory module 2nd semester
Teaching form / SWS:	Lecture with integrated exercises / 4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	All modules from semester 1
Study/examination achievements/form of examination:	Project work.

Module objectives:

Enterprise applications have become more granular in recent years and are operated on cloud platforms. Heavyweight, monolithic applications are being replaced by a large number of smaller services that can be developed and put into production independently of each other. The resulting software systems are distributed systems consisting of a large number of interacting services. This division is intended to prevent the architecture from decaying and ensure that company applications remain maintainable and expandable in the long term.

The aim of the module is to teach participants methods, concepts, patterns, technologies and tools that can be used to develop backend services for these architectures. The knowledge imparted is practised by means of integrated project work.

Intended learning outcomes

Knowledge:

Knowledge of flexible software architectures for enterprise applications and current variants of backend services. Participants know basic concepts and patterns of programming platforms and frameworks for backend services and have practical knowledge of at least one specific programming platform or framework (currently Go). They know different styles and technologies for the communication of different services. They are familiar with various interface technologies and know the rules for good API design during development. You will be familiar with test architectures and test strategies for building services, as well as the basics of their validation. They also know how to create the prerequisites for backend services for provision in the cloud.

Skills:

Participants will be able to design and develop backend services for a technical scenario. They use current development tools and methods as well as design patterns and secure the services as required. They are able to select and use the appropriate communication styles for the use case. They can integrate other services and systems when implementing a service. The quality of the services is ensured by designing and implementing a suitable test strategy and by adhering to guidelines for API design and programming. They are able to create the prerequisites for a fully automated build, test and deployment process that provides the service in the cloud (e.g. a PaaS).

Expertise:

LE#	Learning outcome (LE)	Tested by
LE1	Experience with professional software development and operating environments for backend services.	Project work
LE2	Practical application of general concepts and patterns of programming platforms/frameworks for the development of backend services.	Project work
LE3	Reflection and application of non-functional aspects of backend services (e.g. security, quality of source code, well thought-out API design).	Project work
LE4	Practical application of general test procedures to ensure functional aspects of the services, as well as creating the prerequisites for deployment in the cloud.	Project work
LE5	Creation of your own backend service with a current framework, possibly with the integration of further services	Project work

	using suitable communication styles and interface technologies	

Contents:

- Motivation: Flexible software architectures for enterprise applications.
- Professional software development environment for backend services (e.g. IDEs, build tools, version management).
- Basic concepts of programming platforms for backend services (using Go as an example).
- Styles of inter-service communication
- API design guidelines.
- Test concepts, architectures and technologies.
- Basic security concepts and technologies.
- Containerisation of the backend services.

Media forms:

The module consists of a seminar-style lecture and integrated exercises on the lecture content. The material for the lectures is available in electronic form: Lecture slides, exercise sheets with exercises and code repositories with examples and solutions.

Literature:

- Bodner, Jan (2021). Learning Go - An idiomatic approach to real-world Go programming. 1st Edition, O'Reilly.
- Newman, Sam (2021). Building Microservices - Designing fine-grained systems. 2nd Edition O'Reilly.
- Titmus, Matthew A. (2021). Cloud Native Go - Building Reliable Services in Unreliable Environments. O'Reilly.
- Köhler, Kristian (2021). Microservices with Go - Concepts, Tools, Best Practices. Rheinwerk Computing.

M 7 Software architecture

Module:	Software architecture
Abbreviation:	M 7
Subtitle:	
Courses:	Lecture
Module coordinator:	Prof Dr Christian Kücherer
Lecturer(s):	Prof. Dr Christian Kücherer Prof. Dr Jürgen Münch Markus Großmann
Language:	German
Assignment to the curriculum:	Professional Software Engineering, Master, compulsory subject, 2nd semester
Teaching form / SWS:	Lecture / 4 SWS
Labour input:	Classroom study60 hours Self-study90 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	Methods and technologies of professional programming, software engineering
Study/examination achievements/form of examination:	Project work

Module objectives:

The aim of the software architecture course is to impart the basic knowledge in the field of software architectures for a deeper understanding of different forms of software architectures. The focus is on the development of software structures, orientated towards the domain in which the software system is later to be used. Detailed knowledge of non-functional requirements, which are both decisive for product quality and have a significant influence on software architecture, is of great importance. Important aspects of modern software systems are fast and lightweight development as well as the reusability of functionality. For this reason, the course looks at various component technologies in the context of software architecture. In particular, different types of architectures are dealt with and deepened through practical application by the participants. As software architectures are immaterial and therefore difficult to grasp, the participants use modelling languages and visualisation techniques to model and illustrate software architectures.

The most important methods for analysing architectures are also taught. Modern values and requirements for software architectures are conveyed by discussing the criteria and characteristics of long-lasting architectures and their documentation. Aspects of further development are considered and analysed using the principles of evolutionary architectures.

Intended learning outcomes

Knowledge:

- Basic understanding of the principles of domain-driven design
- Knowledge of different component technologies
- Understanding the influence of non-functional requirements on software architectures
- Knowledge of the procedure for finding an architecture
- Basic architecture patterns, in particular architecture layers, reactive and micro-service architectures.
- Understanding the advantages and application of modelling languages and architecture visualisation with software cities, software tomographs and other software imaging tools.
- Knowledge of methods for analysing architectures
- Knowledge of relevant standards for the documentation of architectures

Skills:

- Participants are able to independently design component-based software architectures according to given domain-specific requirements and quality requirements.
- Refinement of non-functional requirements during the design phase.
- Comparison and selection of different architectural patterns
- Application of modelling languages and tools for the visualisation of software architectures and interpretation of the results.
- Independent implementation of architecture assessments for the validation of software architectures.
- Application of templates and standards for the documentation of architectures.

Expertise:

After completing the module, participants will be able to

LE#	Learning outcome (LE)	Tested by
LE1	Analyse and classify domain-specific and non-functional requirements.	Artefact
LE2	components and document their interfaces.	Artefact

LE3	independently design software architectures based on requirements and different procedures for architecture development.	Artefact
LE4	to be able to assess and apply alternatives when using architectural patterns.	Artefact
LE5	apply methods and procedures for modelling and visualising software architectures and interpret the results.	Artefact
LE6	To be able to carry out an evaluation of a software architecture using suitable methods.	Artefact
LE7	document software architectures using templates and standards.	Artefact

Contents:

At the beginning of the course, participants are immersed in the principles of domain-driven design and deepen their knowledge of non-functional requirements with regard to their relevance for the quality of a software system (LO1). By learning the principles of component technologies and component frameworks for the realisation of system modules, participants apply these principles to a given problem (LO2). As a further theoretical module, various methods for finding an architecture are taught and practised directly using an example. Based on the requirements, a new architecture is defined according to TOGAF, Zachman, the 4+1 model according to Kruchten, and the C4 model for software architectures, and the design decisions are justified (LO3). Important architecture patterns such as layers, direction-based and reactive architectures and microservice architectures are then considered and discussed. Central concepts such as monitoring, tracing and scaling of architectures are discussed as well as principles of data management (EU4). Furthermore, the architecture of an existing open source system is visualised using a software tomograph or software cities. The insights gained will be presented and discussed by the participants (LO5).

The principles of architecture assessment according to ATAM or SAAM are taught for the evaluation of software architectures. Participants learn how to carry out an architecture evaluation using a real example (EU6). The requirements for software architectures are then discussed with regard to their further development, stability, robustness and expandability in the context of evolutionary software architectures. The documentation of architectures is discussed using various templates and standards, such as the arc42 template or the ISO/IEC/IEEE 42010 standard, and practised by the participants using examples (LO7).

Media forms:

The teaching material consists of a slide script, which is available in electronic form, exercise sheets and programme examples. Seminar-style teaching with whiteboard, PC projector and presentation slides, in which examples of the theoretical content are illustrated as well as demonstrations of sample programmes and interactive programme development. Participants work individually or in groups on exercises on the topic of software architecture under the supervision of lecturers.

Literature:

- Bass, Len et al (2013): Software architecture in practice. 3rd edition. Upper Saddle River, NJ ; Munich [et al.] : Addison-Wesley
- Clements, Paul et al. (2011): Evaluating Software Architectures: Methods and Case Studies. Boston, Mass.; Munich [et al.] Addison-Wesley.
- Eilebrecht, Karl and Starke Gernot (2019): Patterns kompakt: Design patterns for effective software development. 5th edition. Berlin, Heidelberg: Springer Vieweg.
- Evans, Eric (2009): Domain Driven Design - Tackling Complexity in the Heart of Business Software. Boston; Munich [and others]: Addison-Wesley.
- Fowler, Martin (2002): Patterns of Enterprise Application Architecture. Addison-Wesley.
- Gharbi, Mahbouba et al (2018): Basic knowledge for software architects: Training and further education according to the iSAQB standard for Certified Professional for Software Architecture - Foundation Level. 3rd edition. Heidelberg, dpunkt.
- Goll, Joachim (2018): Design principles and construction concepts in software engineering: strategies for weakly coupled, correct and stable software. Wiesbaden: Springer Vieweg.
- Reussner, Ralf (ed.) (2009): Handbook of Software Architecture. 2nd edition. Heidelberg: dpunkt.
- Lilienthal, Carola (2017): Long-lived software architectures: analysing, limiting and reducing technical debt. 2nd edition. Heidelberg: dpunkt.
- Starke, Gernot (2018): Effective software architectures: a practical guide. 8th ed. Munich: Hanser.
- Vogel, Oliver et al. (2009): Software-Architektur: Grundlagen - Konzepte - Praxis. 2nd edition. Spektrum Akademischer Verlag.

M 8 Software project 1

Module:	Software project 1
Abbreviation:	M 8
Subtitle:	
Courses:	Project
Module coordinator:	Prof Dr Martin Schmollinger
Lecturer(s):	Supervisors from companies, possibly also professors of the programme
Language:	German
Assignment to the curriculum:	Compulsory subject, 2nd semester
Teaching form / SWS:	Project / The proportion of attendance time varies greatly depending on the chosen project method and organisation.
Labour input:	150 hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	All modules of the 1st semester.
Study/examination achievements/form of examination:	Project work, presentation

Module objectives:

A software project is carried out as part of the Master's programme. In this project, the knowledge learnt can be deepened in an application-oriented manner. Didactically, the Software Project 1 module lays the foundation for the successful implementation of the project in the Software Project 2 module.

In the Software Project 1 module, participants should apply the professional skills they have acquired so far to set up a comprehensive software project and develop them further in a practice-oriented manner. Furthermore, they should reflect on the project assignment with regard to ethical issues and discuss them with the lecturers.

In addition, considerations should already be made during the set-up phase of the project as to how data protection can be guaranteed in the software to be created and during development.

Intended learning outcomes

Knowledge:

Participants are familiar with the scientific and practice-oriented methods for organising and managing software projects. They are familiar with supporting tools for software project management. They are familiar with fundamental topics from the respective application domain. They are familiar with systems, software architectures, technologies and frameworks for software development. They know the basic principles of professional ethics in software development and the concepts of responsibility, value and dilemma.

Skills:

Participants can use relevant methods to systematically create a project organisation, taking into account the division and consolidation of work. They can document and communicate the work status appropriately and adjust the work plan and schedule if necessary. You can use the technologies, frameworks and tools necessary for the realisation of the project. They are able to explain the patterns and software architectures used. You can explain measures to protect personal data. They can present the results of the project group to an expert audience and with specialised visual support at a high level.

Expertise:

Participants have solid technical skills and improved methodological competences. The participants' communication and teamwork skills are further developed. Their social and problem-solving skills are strengthened. They will be able to recognise the challenges of professional ethics in the work of software developers in a specific project situation.

Contents:

Moderated by the lecturer/supervisor but largely independent set-up of a software project for a comprehensive task from any application domain. As part of the module, methods, processes, roles, infrastructures, tools, frameworks, as well as technologies and systems are presented, selected, set up or deepened. In addition, the first simple increments of the software product are created for practice. The project is carried out in groups of approx. 4 - 6 participants. The project result and the project experiences are to be presented to an expert audience in the form of a final presentation and defended in a subsequent discussion. The project results must be summarised in a project documentation.

Media forms: Project-specific media forms.

Literature: Depending on the respective task

M 9 Software project 2

Module:	Software project 2
Abbreviation:	M 9
Subtitle:	
Courses:	Project
Module coordinator:	Prof Dr Martin Schmollinger
Lecturer(s):	Supervisors from companies, possibly also professors of the programme
Language:	German
Assignment to the curriculum:	Compulsory subject 3rd semester
Teaching form / SWS:	Project / The proportion of attendance time varies greatly depending on the chosen project method and organisation.
Labour input:	300 hours in total
Credit points:	10 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite:	Software project 1
Study/examination achievements/form of examination:	Project work, presentation

Module objectives:

A software project is carried out as part of the Master's programme. In this project, the knowledge learnt can be deepened in an application-oriented manner. Didactically, the Software Project 1 module lays the foundation for the successful implementation of the project in the Software Project 2 module.

In the Software Project 2 module, participants will carry out a software project based on the methodological and technical skills they have already acquired. The aim at the end of the module is to have created a software product using modern software engineering methods and technologies in a development team.

Intended learning outcomes

Knowledge:

Participants are familiar with the complexity of applying scientific and practice-orientated methods for implementing software projects. They will be familiar with the details of using supporting software project management tools. They are familiar with advanced topics from the respective application domain. They know the details of the programming frameworks, systems, software architecture and patterns used.

Skills:

Participants are familiar with the tasks of the various roles in the context of software development methods and can take on these roles themselves. They are familiar with methods for estimating effort and prioritising tasks and have gained experience in their application. You can use the technologies, frameworks and tools necessary for the realisation of the project with confidence. They are able to explain and apply the patterns and software architectures used. They can present the results of the project group to an expert audience and with specialised visual support at a high level.

Expertise:

Participants have advanced technical skills and enhanced methodological expertise. Participants' communication and teamwork skills are significantly enhanced. Their social and problem-solving skills are strengthened.

Contents:

The Software Project 2 module focuses on the implementation of a software project. Further topics such as scaling the software are also considered.

Partly moderated by the lecturer/supervisor but largely independent software project for a comprehensive task from any application domain. The task is worked on in a team of participants. The size of the team is 4-6 developers. The project result and the project findings must be presented to an expert audience in the form of a final presentation and defended in a subsequent discussion. The project results must be summarised in a project documentation.

Media forms:

Project-specific media forms.

Literature:

Depending on the respective task

M 10/M 11 Compulsory elective module 1 and 2

W1 Distributed Ledger Technology

Module:	Distributed Ledger Technology	
Abbreviation:	M 10 / M11	
Subtitle:	W1	
Courses:	Lecture	
Module coordinator:	Prof Dr Marcus Schöller	
Lecturer(s):	Prof Dr Peter Hertkorn Prof Dr Marcus Schöller	
Language:	German	
Assignment to the curriculum:	Professional Software Engineering Master, elective subject, 3rd semester	
Teaching form / SWS:		Lecture4 SWS
Labour input:	Classroom	study60 hours Self-study90 hours
Credit points:	5 ECTS	
Prerequisites according to StuPro:	None	
Recommended prerequisite:	None	
Study/examination achievements/form of examination:	Project work	

Module objectives:

Blockchain technologies are the current manifestation of the solution for distributed, consistent and trustworthy data storage. They belong to a broader group of solutions: Distributed Ledger Technology (DLT). This module deals with the underlying concepts and selected methods and procedures in the field of DLT, discusses current developments in this area and examines and evaluates their possible applications. Successful completion of the module should ensure that students are able to weigh up the use of DLT, select and understand suitable technologies and use them to develop applications. Building on this, students should acquire knowledge about the operation of DLT applications.

Intended learning outcomes:

Knowledge:
After successfully completing this module, students will have knowledge of the underlying concepts of DLT solutions and will be able to explain the differences between centralised and decentralised systems. They will have developed an understanding of technical and application-specific aspects of DLT, in particular different consensus protocols, the levels of decentralised applications and various technologies for distributed ledgers. They also have knowledge of DLT-based applications and their development and are able to describe, evaluate and use various methods for creating decentralised applications. Students will be familiar with current trends and developments in the field of DLT and will be able to assess and categorise their impact on the field of DLT.

Skills:
Students will be able to analyse given problems with regard to the use of DLT, select suitable technologies and design and develop applications based on DLT. To this end, they will be familiar with existing DLT implementations and the requirements for operations. Students will be able to analyse and evaluate solutions holistically and thus make technically sound decisions on the possible uses of these technologies.

Expertise:
After completing the module, students will be able to

LE#	Learning outcome (LE)	Tested by
LE1	to know and assess the basics of DLT.	Artefact
LE2	components and their tasks in DLT solutions.	Artefact
LE3	Understand and evaluate the use of DLT.	Artefact
LE4	Understanding software development methods for DLT and apply them.	Artefact

Contents:

First, basic concepts of DLT are introduced: Consensus protocol, cryptography and smart contracts (LE1). Building on this, various DLT solutions are analysed and compared, e.g. Bitcoin, Ethereum, Hyperledger (LE2, LE3). Finally, concrete application scenarios for DLT will be developed and prototypically implemented (LO4).

Media forms:

The teaching material consists of a slide script, which is available in electronic form, exercise sheets and programme examples. Seminar-style teaching with whiteboard, PC projector and presentation slides, in which examples of the theoretical content are illustrated as well as demonstrations of sample programmes and interactive programme development. Students work individually or in groups on exercises relating to DLT with support from the lecturer.

Literature:

- Antonopoulos, Andreas M. (2017): Mastering Bitcoin. 2nd edition. O'Reilly.
- Antonopoulos, Andreas M. and Gavin Wood (2018): Mastering Ethereum: Building Smart Contracts and Dapps. O'Reilly.
- Cachin, Christian and Marko Vukolic (2017): "Blockchain Consensus Protocols in the Wild", CoRR, eprint arXiv:1707.01873, https://dblp.org/rec/bib/journals/corr/CachinV17.
- Drescher, Daniel (2017): Blockchain Basics: A Non-Technical Introduction in 25 Steps. Apress.
- Kosba, Ahmed et al. (2016): "Hawk: The Blockchain Model of Cryptography and Privacy- Preserving Smart Contracts," 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2016, pp. 839-858.
- Laurence, Tiana (2017): Blockchain for Dummies. For Dummies.
- Raval, Siraj (2016): Decentralised Applications: Harnessing Bitcoin's Blockchain Technology. O'Reilly.
- Swan, Melanie (2015): Blockchain: Blueprint for a New Economy. O'Reilly.
- Tapscott, Don and Alex Tapscott (2016): Blockchain Revolution: How the Technology Be- hind Bitcoin Is Changing Money, Business, and the World. Portfolio.
- Wattenhofer, Roger (2017): Distributed Ledger Technology: The Science of the Blockchain. 2nd edition. CreateSpace Independent Publishing.

W2 Digital Product Management

Module:	Digital Product Management	
Abbreviation:	M10/M11	
Subtitle:	W2	
Courses:		
Semester of study:		
Module coordinator:	Prof Dr Jürgen Münch	
Lecturer(s):	Prof Dr Jürgen Münch	
Language:	German	
Assignment to the curriculum:	Compulsory elective module, 3rd semester	
Teaching form / SWS:		Lecture4 SWS
Labour input:	Classroom	study60 hours Self-study90 hours
Credit points:	5 ECTS	
Prerequisites according to StuPro:	None	
Recommended prerequisite:	None	
Study/examination achievements/form of examination:	Lecture:	

Module objectives:

The "Product Management" module enables you to plan the development of digital or digitalised products and coordinate the teams involved in such a way that customer and business goals are successfully achieved. You will gain in-depth knowledge of product conception, digital prototyping, user experience, ideation, data science, validation and leadership.

The development of software-based products today takes place in an environment in which technologies and markets are changing rapidly. Product requirements cannot usually be defined in advance. Nor can you ask customers what features they would like. Steve Jobs already said that it is not the customer's job to find out what they want. In situations like this, traditional approaches to product development with upstream requirements definition are not suitable. But how do you still come up with products that customers want and need? There are new development approaches for this, such as dual-track agile, which combine agile software development methods with modern product management methods. It is particularly important for software developers to have a basic knowledge of product management, as they are often involved in the development and implementation of product strategies. The module prepares you for professions such as product manager, product owner or technical development manager. The knowledge imparted is also particularly important for founders, members/leaders of innovation teams, product developers, user experience designers, software engineers and product marketing managers.

Intended learning outcomes

Knowledge:

Students are able to describe a product vision, identify important customer problems and product assumptions, develop experiments to validate these assumptions and develop a product strategy iteratively. In addition, students have the necessary knowledge for the management of product development and the coordination and leadership of product teams.

Skills:

Students are able to independently apply suitable methods, techniques and tools of agile product management in order to make important product decisions. They develop product visions, roadmaps, learning prototypes and other artefacts. You will validate ideas and products in terms of their viability.

Expertise:

Students will be able to assess different product management methods, select suitable methods and apply them. In doing so, students ensure that customer benefits are generated, business goals are pursued, development costs are reduced and development risks are controlled. The practical relevance of the topic is taught in group exercises and by means of concrete examples and case studies.

Contents:

- Role of the product manager and the product owner
- Principles of product management
- Definition of a product vision
- Development of a product strategy
- Important procedures for identifying customer benefits
- Definition of business and customer goals
- Design thinking, creativity
- Identification and evaluation of various development options
- Definition of good product hypotheses
- Designing, conducting and analysing experiments to validate critical product hypotheses
- Minimum Viable Product (MVP)
- Development of roadmaps
- Release Management
- Impact Mapping
- Value Mapping
- User story mapping
- Metrics, analytics & insights
- Lean Startup
- Customer Development
- Minimum Viable Products
- Corporate culture and product leadership
- Organisational aspects of product management

Media forms:

- Lecture with integrated exercises
- Seminar part with practical exercises
- Lecture material in electronic form

Literature:

- Alvarez, C. (2014): *Lean Customer Development: Building Products Your Customers Will Buy.* O'Reilly.
- Bland, D. (2019): *Testing Business Ideas: A Field Guide for Rapid Experimentation*, Wilely,
- Cagan, M. (2017): *INSPIRED: How to Create Tech Products Customers Love*. 2nd edition. Wiley.
- Cagan, M. (2020): *EMPOWERED: Ordinary People, Extraordinary Products.* Wiley.
- Gothelf, J., Seiden, J. (2016): *Lean UX: Designing Great Products with Agile Teams*. O'Reilly.
- Klein, L. (2016): *Build Better Products. A Modern Approach to Building Successful User-Centred Products.* Rosenfeld.
- Knapp, J., Zeratsky, J., Kowitz, B. (2016): *Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days.* Simon & Schuster.
- Kohavi, R., Tang, D., Xu, Y. (2020): *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing.* Cambridge University Press.
- Lombardo, C. T., McCarthy, B., Ryan, E., Connors, M. (2017): *Product Roadmaps Re- launched: A Practical Guide to Prioritising Opportunities, Aligning Teams, and Delivering Value to Customers and Stakeholders*. O'Reilly.
- Nguyen-Duc, A., Münch, J., Prikladnicki, R., Wang, X., Abrahamsson, P., Eds. (2020): *Fundamentals of Software Startups - Essential Engineering and Business Aspects.* Springer.
- Olsen, D. (2015): *The Lean Product Playbook - How to Innovate with Minimum Viable Prod- ucts and Rapid Customer Feedback*. Wiley.
- Fagerholm, F., Sanchez Guinea, A., Mäenpää, H., Münch. J. *The RIGHT Model for Continuous Experimentation.* Journal of Systems and Software, 123:292-305, January 2017.
- Pichler, R. (2010): *Agile Product Management with Scrum*. Addison Wesley. 2010.
- Pichler, R. (2016): *Strategise: Product Strategy and Product Roadmap Practices for the Digital Age.* Pichler Consulting.
- Ries, E. (2011): *Lean Startup - How Constant Innovation Creates Radically Successful Businesses.* Portfolio Penguin. 2011.

W3 Internet of Things

Module:	Internet of Things
Abbreviation:	M10/M11
Subtitle:	W3
Courses:	
Semester of study:	3
Module coordinator:	Prof Dr Natividad Martínez
Lecturer(s):	Prof Dr Natividad Martínez
Language:	German
Assignment to the curriculum:	Compulsory elective module, 3rd semester
Teaching form / SWS:	Lecture4 SWS
Labour input:	On-campus study Hours Self-study Hours
Credit points:	5 ECTS
Prerequisites according to StuPro:	None
Recommended prerequisite :	None
Study/examination achievements/form of examination:	Lecture

Module objectives:

The Internet of Things (IoT) describes new ecosystems in which different objects with embedded computing and communication capabilities can interact with each other and with the user. This module aims to present students with the basics of the technologies and applications of the Internet of Things and to practice them by developing an IoT project. This includes learning about embedded hardware and software, sensor technology, communication protocols, operating systems and software development environments.

Intended learning outcomes

Knowledge:

- Principles and requirements of ubiquitous systems.
- IoT architectures for the acquisition and aggregation of data.
- Properties of the sensors and actuators for interaction with the environment and with the users.
- Basic hardware and software platforms for IoT systems.
- Data integration through IoT cloud connection.
- Domain-specific business models through IoT: Industry 4.0, automotive and medicine.
- Web-of-Things as an integrative communication platform.

Skills:
Students are able to design and develop new systems and products using IoT. They can plan the architecture across layers; select and prototype suitable hardware, software and communication protocols and integrate networked information.

Expertise:

LE#	Learning outcome (LE)	Tested by
LE1	Apply design principles of IoT applications	Elaboration, project work
LE2	Create and analyse IoT system requirements.	Elaboration, project work
LE3	Create a prototype IoT application (sensor technology, hardware, software, communication and integration of networked information)	Project work
LE4	Use modern development environments and tools.	Project work
LE5	Working in a team to solve complex tasks.	Project work
LE6	competently present and discuss IoT solutions using technical language.	Unit
LE7	Evaluate IoT solutions in terms of their feasibility and business relevance	Peer review

Contents:

The lecture introduces the basics of the Internet of Things (IoT). The lecture includes a seminar-based introduction to the topics together with small exercises that enable students to design their own IoT solution in a team project (L5). In addition to the design principles and architectures for IoT (LE1), students learn the special features of IoT system requirements (LE2). The basic technologies (sensors and actuators, communication protocols, embedded systems, integration of networked information) are presented and prototypically used in the project (LO3). To this end, students learn the typical development methods, environments and tools (LO4). Students present the results and findings of their project (LO6) and assess the approaches of other teams using peer review methods (LO7).

The module covers the following topics:

- Introduction to IoT: Design principles and requirements.
- Data acquisition and interaction through sensors and actuators.
- IoT architectures, communication networks and protocols.
- From computer to smart object through embedded systems
- IoT applications and development methods
- IoT platforms for integration with other information processing systems
- Web of Things (WoT)

Media forms:

Different media forms are selected depending on the content and expertise. Some topics are dealt with in the traditional way using slide scripts, which are projected using a projector and can be deepened, explained and illustrated using the blackboard. The module comprises a lecture with an integrated exercise. To this end, the students will receive information about the necessary installation and the requirements for the systems, which they should develop prototypically under the supervision of the lecturers in the laboratory. The students will then apply the skills they have learnt in their own project and present their results.

The lecture is part of the "International Programme" of the Faculty of Computer Science and is held in English.
held. The examinations may be held in German and/or English

Literature:

- Greengard, Samuel (2015): The Internet of Things (MIT Press Essential Knowledge series). ISBN: 9780262527736.
- Mattern F., Flörkemeier, Ch. (2010): From the Internet of Computers to the Internet of Things. Informatik Spektrum, Vol. 33, no. 2, pp. 107-121
- Porter, M.E., Heppelmann, J.E., (2014): How Smart, Connected Products Are Transforming Competition. Harvard Business Review 92, no. 11, pp. 64-88
- Weiser, M. The computer for the 21st century
- In addition, current articles from specialist journals and conferences as well as Internet resources.

M 12 Master Thesis

Module:	Master thesis
Abbreviation:	M 12
Subtitle:	
Courses:	Master Thesis
Semester of study:	Each
Module coordinator:	Prof Dr Martin Schmollinger
Lecturer(s):	All professors involved in the Master's programme
Language:	German
Assignment to the curriculum:	Compulsory subject, 4th semester
Teaching form / SWS:	Master's thesis / No compulsory attendance time
Labour input:	900 hours
Credit points:	30 ECTS
Prerequisites according to StuPro:	50 ECTS for previous studies with at least 210 ECTS, otherwise 80 ECTS
Recommended prerequisite:	All other courses of the Master's programme Professional Software Engineering
Study/examination achievements/form of examination:	Master's thesis / colloquium

Module objectives:

The Master's thesis is a final examination paper with which the participants prove that they can independently work on a comprehensive task in computer science according to basic scientific methods within a given time frame and defend the procedure and the goals achieved.

Intended learning outcomes

Knowledge:

Participants have comprehensive knowledge of the subject area of the thesis. Participants are familiar with all formal requirements for writing academic papers.

Skills:

Participants will be able to work independently on a completed topic using scientific methods. They will have mastered the relevant techniques for writing a scientific paper, such as structuring, citing and maintaining an appropriate external form.

Expertise:

Participants are capable of abstraction and modelling for the purpose of practical analysis, conception and design. They have analysis, design, realisation and project management skills. They are able to develop goal-orientated solutions.

Contents:

Topics for Master's theses relate to tasks in computer science that are relevant to the discipline today and for the foreseeable future. The topics include several information technology, software technology, media, psychological, didactic, economic or other aspects that have a complex connection to the solution of the task.

Media forms:

Technical and methodological support for participants through counselling and support meetings, which also take place on site for company-related work. Participants are also required to research and reference relevant information and, if necessary, to prove the relevance and goal-orientation in the company environment. Presentations by the participants regarding the progress of their work and further planning.

Literature:

Depending on the respective task