Module Handbook 2009 Version 2

Bachelor and Master Program

Computer Science

Faculty of Computer Science, Electrical Engineering and Mathematics of the University of Paderborn

Bachelor Degree Course

Computer Science (6 sem.)

Master Degree Course

Computer Science (4 sem.)

Paderborn, March 30, 2017

Table of Contents

Preliminary remarks

The module information in this catalog

- comprehensively describes the aims, contents, and interrelationships of the Degree Course classes and modules,
- provides students with the useful, authoritative information they need to plan their studies,
- helps academic staff and other interested persons to understand the design of the modules of the Degree Course.

The module descriptions are structured according to a scheme developed by a committee of subject matter colleagues, the faculty, and the Paderborner Lehrerausbildungszentrum (Paderborn Teacher Training Centre, PLAZ). The committee also considered guidelines, suggestions, and ideas from a wide range of literature regarding developing modules. Apart from specifying the content and organization, we have placed special emphasis on describing the role of the modules in the Degree Course as meaningfully as possible, including the learning goals envisaged. The Handbook shows students and teaching staff not only **what** is taught, but also **why** it is taught. We have explained the scheme of the descriptions below. Please note that we regard the first part (up to and including the "Mode" section) as fixed for several years. We may adjust the latter part of the description, if necessary, for each instance of the module.

Besides that, this Module Handbook also records the examination modalities and describes the grading policies for the individual modules.

Scheme for class and module descriptions

Class: Name of the class

Role of the class in the Degree Course

– Places the class within the degree (associated with the aims of the Degree Course and with sections of the Degree Course regulations).

– Describes the significant contents and methods of the class and their significance for a subject area or the subject of Computer Science as such.

– Relates the class to other classes / modules.

– Uses of the class in other Degree Courses.

Content structure of the class

This section provides the content structure and time schedule of the class.

Use of the content

This section describes the typical application fields of the contents and methods of the class, using suitable examples.

Prerequisites and prior knowledge

This section specifies the formal prerequisites and prior knowledge required for the student to participate successfully in the class.

Learning goals of the class

This section specifies the learning goals by linking contents (central areas of knowledge) to abilities (central areas of competency): e.g., "*Students should be able to ...*". We have divided these goals into four areas of instruction.

Teaching of factual knowledge – content competency

Relevant knowledge areas for the class including selected relationships to applications.

Teaching of methodological knowledge – methodological skills

Academic methods of the subject taught in the class. The students are expected to apply these methods to typical examples.

Teaching of transfer skills

Examples for applying the methods learned in the class to new contexts - starting from the class contents.

Teaching of normative evaluation skills

Criteria and examples for evaluating the contents and methods students acquired in the class regarding problem scenarios in computer science (e.g., suitability and limitations of the methods, the quality of solutions / approaches, the implications of solutions / approaches or products for society and community).

Key qualifications

After completing this class (module), the students will have acquired

– the ability to cooperate and work in teams

– competency in oral presentations and discussion moderation

– ability to use modern information and communication technologies

– ability to develop strategies for acquiring knowledge

– intercultural competencies

– foreign language skills relevant to the subject

Module assignment

– Required, required elective, or elective class

– Assignment to the modules

Mode

– Credit points for each module (workload)

– Credit points of the class

– Extent and type of the class offered, e.g., 6 SWS (4 lectures and 2 tutorials per week)

– Frequency of the class being offered, e.g., every winter semester

– Duration, e.g., 1 semester

Methods of implementation

This section specifies the social structure of the class and its didactic/methodical procedures (e.g., the class may include tutorials in small groups, project learning with a high degree of active participation by the students, a continuously case-oriented approach when teaching contents, small application examples as starting points for introducing a subtopic, theoretical concepts followed by examples putting them into a practical context, individual study phases with learning objects, guided tours in virtual learning environments, deconstruction of computer science systems with transfer, blended learning ....)

Organizational arrangements / media use / literature references

This section describes the

- Organizational form of the class (e.g., lecture class, tutorial class, seminar, lab, project study, individual study, virtual seminar)
- References to activities expected of the students
- Materials used, e.g., exercise sheets, sample solutions, animations....
- Media used, e.g., notes regarding IDEs, software tools....
- Literature references for the class
- References to (web-based) class material, where applicable

Examination modalities

This section describes the

- Types of examinations held as part of the class (e.g., written or oral examinations, presentation, assignment, project, certificate about an internship/cooperative) that permit a conclusion as to whether student has achieved the standards / learning targets.
- Specifications regarding the compensation of a class-oriented partial examination within a module examination
- Information on grading policies

Person responsible for the module

Name of the module supervisor

Study Goals

Goals of the entire study program

Typical for the computer science classes at the University of Paderborn is their distinctive scientific orientation, an emphasis on specific contents and the adequate structure of the different forms of studying. The computer science classes are scientific classes which are organized foundationally- and methodically oriented. Due to their scientifically based structure, they enable the student to successfully work as a computer scientist in an industrial or scientific environment his/her entire working life because they do not only teach today's actual contents but also theoretical based fundamental concepts and methods that will abide unchanged even after actual trends are over. This global goal is reflected by the entire study program. For this reason, fundamental concepts are presented coherently and cross-functionally, a sound education in mathematic basics is taught and deepening modules of individual areas are offered.

Study results of the Bachelor Degree Course

In general, the students should

- master theoretically based fundamental concepts and methods of computer science;
- be able to act responsible with regard to the effects of the technological changes;
- dispose of a wide spectrum of general computer science knowledge;
- should be able to recognize a computer science problem, choose an adequate scientific method to solve this problem and apply it properly.

The graduates of **the Bachelor Degree Course** should

- master the mathematical basics of computer science;
- understand the structure of software systems and their construction as an integral process of production;
- master scientifically sound programming methods;
- master concepts for the design and analysis of efficient algorithms;
- be able to assess the performance limits of computer systems;
- be able to create distributed and embedded systems with an efficient and safe resource management;
- be able to apply special methods and techniques for designing and programming human-computer-interactions at the computer and computer graphics.

Study results of the Master Degree Course

In general and including the study results of the Bachelor Degree Course, the students should

- know basic concepts and methods from other disciplines for the interdisciplinary discourse;

- be able to communicate in English in subject-relevant matters, orally and in writing;
- be able to take over management functions for demanding projects in research, development, industry or administration.

The **Master Degree Course** is intended to deepen the knowledge and abilities that have been acquired in the previous Bachelor Degree Course. The student chooses one out of the four computer science fields (SWT&IS, MuA, ESS, MMWW) to consolidate his/her knowledge in this field and of which he/she has to successfully complete at least three. Furthermore, the students complete their knowledge and abilities in the three remaining fields by successfully passing one of the Master modules each. There is a huge variety of classes to choose from within the modules, so that each student may form his/her individual profile and, at the same time, he/she has the possibility to intensively deepen his/her knowledge.

Goals in the field of Software Technology and Information Systems

Software Technology covers all measures, institutions and development procedures that are necessary for the maintenance and application of software systems. The main challenges are the size and complexity of today’s and future’s software systems.

Our education will teach the students basic scientific principles, concepts and methods of software technology. After having finished their education, the students should be able to develop software systems according to predetermined technical, economical and sociological conditions and (later) manage software projects. They should be able to take necessary measures to solve problems that arise among software projects. In addition to technical competence, the students have to be able to communicate their thoughts and ideas and they have to be able to work as part of a team.

In addition to the application of most recent methods and measures of software technology, the graduates of the Bachelor Degree Course should be able to independently familiarize themselves with future technologies. Furthermore, the graduates of the Master Degree Course should be able to master the scientific basics of software technology and they should be able to adjust, to further develop and to scientifically back up techniques.

Goals in the field of Models and Algorithms

The main focus of the area *Models and Algorithms* (MuA) is on the analysis and modeling of problems and on the algorithmic transfer and evaluation of solutions according to their quality and in particular with their efficiency.

On the one hand, this education teaches the students knowledge of scientifically based algorithms - e.g. for graphs, geometry, coding and optimizing problems and communication problems in networks - and, on the other hand, it teaches the ability to classify problems according to their computability and complexity and then design creative efficient algorithms for these problems and to analyze them with regard to their correctness and efficiency.

The Bachelor Degree Course teaches the essential modeling and algorithmic techniques. The graduates should be able to realize the fundamental and the complexity limits of computability and, furthermore, they should be able to master scientific basics of different algorithmic methods and fields of application.

The Master Degree Course teaches knowledge of advanced algorithmic techniques (efficient algorithms, approximation algorithms, optimization, parallel algorithms, communication algorithms for networks). The graduates should also be able to apply the algorithmic theory in important fields like optimization, algorithmic coding theory and algorithmic geometry. With regard to the field of computer safety, the students should be able to apply the methods of complexity theory and cryptography and they should possess a detailed knowledge of the limits of algorithmic theory.

The overall goal of all classes in this field is to acquaint all students with the basic ways of thinking and working in modeling and in algorithmic theory. This also includes – apart from recognizing fundamental mathematic structures in problems - the ability to apply mathematical methods or adjust them to new problems.

Goals in the field of Embedded Systems and System Software

The field of *Embedded Systems and System Software* (ESS) is an interface between computer science and engineering. It consists of the sections operating systems and distributed systems, real time systems, embedded systems and computer communications.

The classes of ESS will enable the students to understand the interaction of hardware and software on different levels of computer science as well as the effect of computer science on applications beyond classical computing. The students should be able to master procedures for managing resources efficiently and safely, especially with external predetermined physical restrictions and they should be able to assess their importance. The students should master the general scientific concepts, methods and tools and they should be able to adjust them to specific problems and requirements. Furthermore, they should be able to break down complex systems into abstract components and then determine and evaluate corresponding realization possibilities for hardware and software components, according to predetermined constraints. In addition, the students should be able to apply the acquired concepts and methods to future developments e.g. in the fields of computer communication or intelligent technical systems.

Graduates of the Bachelor Degree Course are able to recognize the requirements of embedded and machine oriented systems and choose adequate solution concepts and methods for them in the Field of ESS. Furthermore, they are able to work out new approaches independently and to apply them practically. In addition to this, the Master Degree Course teaches some deeper knowledge in some special fields of ESS (e.g. storage systems, mobile communication). The graduates of the Master Degree Course are able to further develop concepts and methods on their own and to systematically include system connections in an extensive system optimization.

Goals in the field of Human-Machine-Interaction

The graduates of the Bachelor and Master Degree Course should be able to design human-machine-interfaces ergonomically. Furthermore, they should master the design of information supplies on the net as well as concepts and techniques for generating and editing of three-dimensional scenes and digital pictures. They should know and apply scientifically based techniques to support cooperative knowledge work while learning and working. They should be able to apply general ethical and legal principles in the field of development and usage of software systems and to evaluate their practical consequences in the corresponding working area (data protection, copyright, freedom of information, ethical guidelines).

The Bachelor Degree Course is the scientific base for all of the above. This mandatory class teaches essential design principles, rules and norms, effects on humans as well as concepts and methods of usability evaluation on this subject. The second part of the studies will deepen these subjects and, furthermore, it will teach a sound scientific approach to computer graphics. The Bachelor Degree Course gives a scientific foundation in the field of *human-machine interaction* which every computer scientist should know for his/her practical work.

The Master Degree Course deepens all of the above mentioned knowledge up to latest research in order to give the graduates a sound scientific foundation which will be essential for their research career. This affects current, individually adjusted lectures on subjects like development tools, modeling, usability, assistive technologies, rendering, image processing, design of digital media, cooperation support systems, e-learning and contextual computer science.

Grading Policies

The final grade for the Bachelor-/Master thesis is calculated from the grades of the module exams according to BScPO §19 (2) and MScPO §19. The grades for the module examinations either result from separate module examinations or from class-related partial examinations (BScO §5 (1) and MScPO §5 (1)). The following list includes all modules in the Bachelor and Master Degree Course.

Usually, the grade attained for a module of the Bachelor Degree Course that consists of **one class only** is, at the same time, the grade for the entire module. This applies to the following modules:

- I.1.3 Database Foundations
- I.2.1 Modeling
- I.2.2 Data Structures and Algorithms
- I.2.3 Introduction to Computability, Complexity, and Formal Languages
- I.3.2 Concepts and Methods of System Software
- I.4.1 Foundations of Human-Machine-Interaction
- I.5.1 Analysis
- I.5.2 Linear Algebra
- I.5.3 Stochastics
- II.5.2 Bachelor Thesis

The modules of the **Bachelor Degree Course** that consist of **several classes** - and for which there are class-related individual examinations - are graded according to the achieved performance credits. This applies to the following modules:

- I.1.1 Programming Techniques
- I.3.1 Foundations of Computer Engineering and Computer Architecture
- II.1.1 Software Technology and Information Systems
- II.2.1 Models and Algorithms
- II.3.1 Embedded Systems and System Software
- II.4.1 Human-Machine-Interaction

The following modules of the Bachelor Degree Course are graded as follows:

- I.1.2 Software Engineering

  The module consists of the two classes

  - Software Design (SE)
  - Lab exercises in Software Engineering Lab (SWTPRA)

  The grade for the module I.1.2 is computed as average of the grade for SE and the lab exercises (SWTPRA).
  The exam for SWTPRA is done as project-oriented partial exams and a final written exam. All parts must be successfully passed. The grade for the SWTPRA is computed as a weighted sum of the grades of all partial exams and the written exam.

- II.5.1 Key Competences

  The module consists of two classes:

  - Undergraduate seminar
  - Mentoring

  The grade for the undergraduate seminar is the module grade.

The Mentoring Program is completed with one mentoring-credit-point awarded by the mentor following a reasonable dialogue with the student.

The modules and classes of the **Master Degree Course** are not offered regularly. There may be adjustments of classes to state-of-the art-technology from time to time which means that classes within the module are being cancelled or added.

A.
The following regulations apply to the modules under § 16 Part 4 No. 1 and No. 3 (Modules No. III.1.* to III.4.*) of these Regulations *for the Conduct of Examination (Prüfungsordnung)*:

The final examination is an oral examination.

The following regulations apply to the modules III.1.1 Model-based software development and III.1.6 Constructive Methods in Software Engineering: The final oral examination mainly focuses on one of the two catalogue classes. The student has to pass a partial examination in the corresponding other class which is at the same time a prerequisite to be admitted to the final oral examination.

For all further classes **without seminars**, there will be a final oral examination on the contents of the module and the grade of this oral examination will also be the grade for the entire module.

With regard to the modules that **include a seminar**, the final oral examination mainly focuses on the content of the catalogue class. The student has to pass the seminar in order to be admitted to the final oral examination.

B.

There are no formal prerequisites for the module III.5.1 Project Group. The organizer watches and grades the performance of each individual participant during the entire duration of the project group. For the final grade of the project group, we consider the individual contribution to **the project group result** (implementation, for example), the **project group reports** as well as a **final professional conversation which is usually just as long as an oral examination**. The module grade is a summary of all above listed points.

The module III.5.2 *Master Thesis* is examined and graded according to § 18 Abs. 2 of the *Regulations for the Conduct of Examination (Prüfungsordnung.)*

I. Modules in the first stage of the Bachelor Degree Course

I.1 Field: Software Technology and Information Systems

I.1.1 Programming Techniques

Role in the Computer Science Degree Course

Software development is a central activity in computer science. Software developers must be able to analyze and model tasks, design software structures, and implement them in a programming language. This module teaches introductory and fundamental knowledge and skills in programming and, together with the Modeling and the Software Technology modules (see sections I.2.1 and II.1.1 below), sets and practices the foundations in software development.

This module enables the participants to

- apply a programming language relevant in software development (currently Java),
- use basic terms of the object-oriented programming methodology,
- understand in general the basic concepts of programming and application languages, and
- understand typical features of non-imperative languages.

Overall, this module should enable the student to learn new programming languages and their applications independently. Further, this module forms the core of the basic training in Software Technology in the Computer Science Degree Course, together with the mandatory modules on Modeling (I.2.1) and Software Technology (II.1.1). The elective modules on languages and programming methods in the Bachelor and Master Degree Course deepen the topics and goals of this module with respect to languages, their interpretation, and application.

Content structure

The module is structured into three parts: The first part, Foundations of Programming 1 (GP1, 1 semester) and the second part, Foundations of Programming 2 (GP2, 1/2 semester) carry out the basic training in a programming language. The third part, Foundations of Programming Languages (GPS, 1/2 semester), teaches the concepts of programming languages in general.

The class Foundations of Programming 1 (GP1) includes

- Basic terminology of programs and their execution
- Classes, objects, data types
- Program and data structures
- Object-oriented abstraction
- Object-oriented libraries

The class Foundations of Programming 2 (GP2) includes

- Graphical user interfaces
- Event handling and applets
- Parallel processes, synchronization, monitors

The Foundations of Programming Languages (GPS) include

- Syntactic structures
- Scope of definitions
- Lifespan of variables
- Data types
- Function calls and parameter passing
- Functional programming
- Logic programming

Usability of the content

You will use the knowledge and abilities acquired in this module during study and employment wherever you develop programs. This goal requires that you gain further practical experience beyond the tutorials of this module. Together with the Modeling (I.2.1) and Software Technology (II.1.1) modules, this module provides you with the ability to develop software in a study or work situation. Likewise, together with the knowledge from *Foundations of Programming Languages* (GPS, in German: *Grundlagen der Programmiersprachen*), you will use these abilities independent of the respective programming language. In addition, you will revisit and develop the topics from this module further in classes on languages and programming methods.

Prerequisites and prior knowledge

The classes belonging to Foundations of Programming (GP1and GP2) require basic abilities in computer use. We do not expect you to have programming skills, but they may simplify the introduction. The classes of *Foundations of Programming Languages* (GPS) require that you have acquired the basics of one programming language. These are taught, for example, in the first part of the module. Additionally, we expect you to know context-free grammars for example, from the Modeling module.

Learning goals

Teaching of factual knowledge

In this module, you will

- learn the constructs of the Java programming language (GP),
- understand the basic concepts of programming and application languages,
- understand typical features of non-imperative languages (GPS)

Teaching of methodological knowledge

In this module, you will

- apply the language constructs that you have learned in a sensible manner and with understanding (GP),
- understand and apply the basic concepts of object-orientation (GP),
- reuse software from object-oriented libraries (GP),
- be able to develop basic grammars, type specifications, and functional and logic programs (GPS)

Teaching transfer skills

In this module, you will

- transfer practical experiences in program development to new tasks (GP, GPS)
- independently learn new programming and application languages (GP, GPS)

Teaching of normative evaluation skills

In this module, you will

- assess the suitability of languages for specific purposes (GPS)

Key qualifications

For this module, we expect for you to

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework, and take part in class tutorials.

Module assignment

This is a mandatory module in Software Technology

Mode

Credit points: 8+4+4 ECTS (GP1, GP2, GPS)

SWS: 4+2, 2+1, 2+1

Frequency: annually; GP1 in the winter semester; GP2, GPS follow each other in the summer semester

Duration: 2 semesters

Methods of implementation

In GP,

- the language constructs are introduced and explained using typical examples; you will then try them in practice during the tutorials,
- object-oriented methods are demonstrated, mainly via using libraries,
- supervised computer work on lab programming problems is carried out in some tutorials.

In GPS,

- you will explore language constructs, language features, and programming paradigms in comparison and contrast to those learned in GP,
- you will acquire functional and logic language constructs and programming concepts, including lab examples in SML and Prolog.

Organizational arrangements / media use / literature references

- Lectures with overhead presentation

- Tutorial classes in small groups
- Some supervised laboratory work
- Activities expected of the students: cooperate during tutorials, prepare homework, and review the lectures
- Exercise sheets and sample solutions are presented in class tutorials
- Textbook for GP: J. Bishop: Java lernen, Pearson Studium, 2. Aufl., 2001
- Web-based lecture material

Examination modalities

Written examination in *Foundations of Programming 1*

Written examination in *Foundations of Programming 2*

Written examination in *Foundations of Programming Languages*

Person responsible for the module

Szwillus

I.1.2 Software Engineering

Role in the Degree Course

Software Engineering presents concepts, languages, methods, and tools for developing and maintaining large software systems. In this context, we will pay substantial attention to the quality of the software systems. In particular, this attention involves ensuring that functional and non-functional requirements for the software system are met. Depending on the area of application, we will place a different emphasis on the individual system requirements. Examples include safety requirements in embedded systems or usability requirements in interactive systems.

The classes in this module introduce the object-oriented specification of software systems using the UML language, which is now considered to be a de-facto standard. In a subsequent, practical exercise, a team develops a nontrivial software project, so students will apply the knowledge they acquired in both this module and the Programming Techniques module.

Content structure of the module

The module consists of two mandatory classes:

- Software Design (SE)
- Software Engineering Lab (SWTPRA)

The classes are structured as follows:

Software Design (SE):

This class introduces modeling languages for describing the static and dynamic aspects of software systems in general and of user interfaces in particular. This description includes in particular the object-oriented modeling language UML (Unified Modeling Language), which in turn is based on diagram languages such as class, sequence, collaboration, state, and activity diagrams. The class is concluded with notes on methods for using these languages in software development

Software Engineering Lab (SWTPRA):

This Software Engineering Lab is a 6-hour lab that includes one lecture on project management. A complex software development task is carried out by a team of about 10 students using UML and Java.

The main focus of the lab is on experiencing team-oriented software development, using commercially available tools and methods (Rational Rose, Configuration and Version Management [CVS]). At the start of the lab, the students familiarize themselves with the task by using a partially complete source code. The team will extend this code during the lab, including re-documenting this source code. Significant components that ensure that the project stays close to practice include generating milestone plans, implementing project management techniques (which are partly carried out by the students), creating cost estimates, and logging the effort using time sheets.

Usability of the content

The knowledge and abilities acquired in this module form the major foundation for a methodically demanding implementation and management of large software projects in industry.

Prerequisites and prior knowledge

For the Software Design Course basic knowledge in a language suitable for software development (e.g. Java) is required. Prerequisite for the Software Engineering Lab is: 1. Passing the written examinations in Foundations of Programming 1 and 2 as well as passing the written examination in Software Design.

Learning goals

Teaching of factual knowledge

In this module, you will

- learn techniques and tools for (object-oriented) modeling, documenting, and organizing large software projects

Teaching of methodological knowledge

In this module, you will

- be able to apply languages and tools in software development and learn a software project's organizational sequence of events from requirements definition to delivery

Teaching of transfer skills

In this module, you will

- learn languages and tools used within explicit software development processes

Teaching of normative evaluation skills

In this module, you will

- get to know the practical benefit of thoroughly planned projects
- familiarize yourselves with the problems of team-oriented software development and with initial approaches to overcome them

Key qualifications

For this module, we expect for you to

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework, and take part in class tutorials.
- present technical facts (SWTPRA)
- demonstrate technical writing while generating project documentation (SWTPRA)

Module assignment

Mandatory module during undergraduate studies.

Mode

Credit points: 4 (SE) + 10 (SWTPRA) ECTS
SWS: 2+1 (SE), 1+4+1 (SWTPRA)
Frequency: each class once per year

Methods of implementation

- Methods and technologies are introduced and explained using typical examples; they are then practiced during the tutorials (SE)
- Implement a project as described above, including regular, interim presentations and group meetings during which minutes are taken (SWTPRA)

Organizational arrangements / media use / literature references

- Lecture class with overhead presentation (SE)
- Tutorial classes in small groups (SE)
- Small group meetings (SWTPRA)

Examination modalities

SE: Written exam.

SWTPRA: Oral presentation on milestones, written delivery of the source code, the design, the documentation, protocols of the tests, protocols of group meetings, and an executable installation on a web page. The SWTPRA exam is done in the form of project-oriented partial exams and a written final exam. All exam parts must be successfully passed.

The grade of the whole module is the average of the written exam of SE and the grade of the lab exam (SWTPRA).

Person responsible for the module

Engels

I.1.3 Database Foundations

Role of the module in the Computer Science Bachelor Degree Course

Databases play a central role in enterprises because a large part of the company "knowledge" is stored as data in databases. It is crucial to the company that these data are correct and, in particular, that they are consistent. Also, the data must be able to be queried and accessed easily. As a result, the organization of large volumes of data in databases plays a central role in business, industry and government. Among other things, datasets must be organized in such a way that they are free of redundancy but nevertheless complete. In addition, datasets stored in databases are the main data source for many application programs. In turn, the datasets are often updated via application programs. This module provides the foundations of databases that almost all companies use.

Content structure of the class

This module consists of a class *Foundations of Database Systems*. This class is structured as follows:

- Entity-relationship model and conceptual database design
- Relational data model
- Relational algebra, tuple calculus, domain calculus, and relational completeness
- SQL data definition language
- SQL data manipulation
- The SQL query language
- Views, access rights, and the problem of view and update
- Transactions in SQL
- Embedded SQL
- Functional dependencies, keys, and other integrity conditions
- Database draft schema and normal forms

Usability of the content

You will find that the knowledge and abilities gained in this class are applied in practice in almost all companies. For that reason, you will study them further in additional classes, in particular in the class *XML Databases*, and in special lectures and seminars.

Prerequisites and prior knowledge

Programming knowledge is required to the extent to which it is taught in the classes *Foundations of Programming 1* and *2*.

Learning goals of the class

Teaching of factual knowledge

In this class, you will learn

- theory and concepts of relational databases
- basic concepts of relational query languages
- basics of database design

Teaching of methodological knowledge

In small groups during tutorial classes, you will learn

- to formulate complex queries correctly to relational databases

- to design a database schema that is as free of redundancy as possible

During lab tutorials, you will learn

- to submit your own SQL queries to existing relational databases
- to write programs that read or modify datasets in databases
- to define and develop your own databases

Teaching of transfer skills

In this class, you will learn

- to transfer the competencies and abilities gained to other data sources or database systems
- to manage access privileges

Teaching of normative evaluation skills

In this class, you will learn

- to evaluate and assess the suitability and limitations of the relational data model
- to assess the programming effort for database queries and database programming
- to recognize and estimate the consequences of a change in the database schema
- to evaluate the risks of a badly designed database schema.

Key qualifications

In lab tutorials, you will familiarize yourselves with SQL, the most significant database query language used in industry. Through your own computer exercises that expose you to this technology, you will gain the experience necessary to be able to acquire a multitude of database technologies based on SQL.

Module assignment

Mandatory module during undergraduate studies.

Mode

- Credit points for each module (workload): 4
- Credit points of the class: 4
- SWS: 2 lectures + 1 tutorial per week
- Frequency of the class offered: every summer semester
- Duration: 1 semesters

Methods of implementation

- The foundations and concepts are introduced as part of a lecture class.
- The theoretical concepts are subsequently developed and intensified in small groups during tutorial classes. This method is used in particular for the core concepts of databases (data model, algebra and calculus, integrity conditions and database draft schema).
- The practical skills are acquired through computer-based exercises where you submit your own database queries and develop your own databases based on examples from the lecture. This method is used in particular for the learning SQL.

Organizational arrangements / media use / literature references

- Lecture with textbooks, scripts, or overhead presentation and small executable example programs at the computer

- Tutorials: as tutorial classes in small groups with exercise sheets and homework, and as practical exercises on the computer.
- Activities expected of you: Co-operation during tutorial classes, properly prepare homework, develop and test your own database application software at the computer
- Standard textbooks dealing with databases
- Teaching materials on the web

Examination modalities

Written examination

Person responsible for the module

Böttcher

I.2 Field: Models and Algorithms

I.2.1 Modeling

Role of the class in the Degree Course

Modeling is a typical working method in computer science and is used in all areas of the subject. Tasks, problems or structures are analyzed and described as a whole or in part before they are solved or implemented by designing software, algorithms, data and/or hardware. By modeling a problem, you show whether it has been understood and how. Modeling is therefore a requirement and yardstick for the solution and often also provides the key for a systematic design. A wide range of calculi and notations are available as means of expression in modeling. They are specific for different types of problems and tasks. As a result, different modeling methods are used in the various fields of computer science. The significance of the modeling and the diversity of the methods are particularly pronounced in the design-oriented areas (software technology, hardware design).

Content structure of the class

1. Introduction

    Terminology: Model, modeling

2. Modeling with basic calculi

    Domains, terms, algebras

3. Logic

    Propositional logic, program verification, predicate logic

4. Modeling with graphs

    Path, connection, matching, dependencies, control structures, flows

5. Modeling of structures

    Context-free grammars, entity-relationship model

6. Modeling of processes

    Finite automata, Petri nets

Usability of the content

The knowledge and abilities you will acquire are applied and expanded in many lectures, for example, grammars in GPS, ER models in SE, logic in Knowledge-based Systems and in Computability, Petri nets in GTI, graphs in DuA. You should know the basic calculi, domains, terms and logic for any type of formal description. Modeling is a typical working method in the professional life of a computer scientist (see above).

Prerequisites and prior knowledge

Willingness and ability to learn several types of formal calculus.

Learning goals of the class

Teaching of factual knowledge

In this class, you will

- learn basic concepts of the calculi taught,
- gain an overview of the basic modeling methods and calculi

Teaching of methodological knowledge

In this class, you will

- have a good command of the conceptual core of the calculi,
- learn the techniques typical for the methods,
- apply calculi to typical examples

Teaching of transfer skills

In this class, you will

- use the calculi learned to model new tasks in tutorials and as homework.

Teaching of normative evaluation skills

In this class, you will

- investigate the suitability of the calculi for modeling partial aspects of larger tasks
- learn the practical value of precise descriptions.

Key qualifications

For this module, we expect for you to

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework, and take part in class tutorials.

Module assignment

Mandatory module in Models and Algorithms (I.2).

Mode

- Credit points for this module (workload): 10
- SWS: 4 lectures, 4 tutorials per week
- Frequency of the class offered: every winter semester
- Duration: 1 semesters

Methods of implementation

For each modeling calculus,

- the conceptional core of the calculus is introduced with some typical small problem tasks as motivation,
- application techniques and application areas are demonstrated on examples and are applied in the tutorials,
- advanced aspects of the calculus, its role in fields of computer science and lectures, and algorithmic solution techniques are only referred to here.

- work is carried out on a medium-sized modeling task (for example, vending machine). At the end of the class, the applications are discussed for comparison.

Organizational arrangements / media use / literature references

- Class with overhead presentation
- Tutorial classes in small groups
- Activities expected of the students: Co-operation during tutorial classes, homework
- Exercise sheets and sample solutions are presented in class tutorials
- Web-based lecture material: Winter semester 2001/2002:
  U. Kastens: http://www.uni-paderborn.de/cs/ag-kastens/model

Examination modalities

Written examination

Person responsible for the module

Kastens

I.2.2 Data Structures and Algorithms

Role of the class in the Bachelor Degree Course

Algorithms form the foundation of every hardware and software: A circuit converts an algorithm into hardware, a program makes an algorithm "understandable to the computer." Algorithms thus play a central role in computer science. An important goal of algorithm design is (resource) efficiency, for example, developing algorithms that solve a given problem as quickly as possible or with as little memory requirement as possible.

Inextricably connected to efficient algorithms are efficient data structures, that is, methods that organize large data sets in the computer to permit an efficient response to queries such as *find*, *insert*, and *delete* as well as to more complex queries.

The design and analysis methods for efficient algorithms and data structures are introduced in this class, and the basic examples such as sorting algorithms, dynamic search structures, and graph algorithms form part of the foundations of algorithm development and programming for large areas of computer science.

Content structure of the class

- Introduction
  Models of computation, efficiency measures, examples
- Sorting procedures
  Quicksort, heapsort, mergesort
- Data structures
  Linked lists, trees, graphs
  Dynamic search structures
  Search trees, balancing of search trees, hashing
- Design and analysis methods
  Recursion and the master theorem, divide and conquer, dynamic programming, backtracking, branch & bound, greedy algorithms
- Graph algorithms
  Shortest paths, minimal spanning trees, flow problems

Usability of the content

The knowledge and abilities you acquire are applied and expanded in many fields, for example, in operating systems and information systems, hardware and software design, computer graphics, operations research, and of course in the advanced classes on algorithms, networks, optimization, and parallel computing. Algorithm design is a typical working method in the professional life of a computer scientist.

Prerequisites and prior knowledge

Willingness and ability to learn the creative process of algorithm design and efficiency analysis among others using mathematical methods.

Learning goals of the class

Teaching of factual knowledge

In this class, you will

- Design methods for efficient data structures and algorithms

- Develop efficient data structures and algorithms for selected basic problems
- Design methods for proof of correctness and efficiency analysis of algorithms and data structures

Teaching of methodological knowledge

In this class, you will

- Develop independent, creative algorithms and data structures ("How do I design the creative process from the algorithmic problem to the efficient algorithm?")
- Use mathematical methods for proof of correctness and efficiency analysis
- Understand the interaction between algorithm and data structure
- Assess the quality of algorithms and algorithmic approaches with regard to efficiency
- Independently acquire new algorithms, data structures, and algorithmic ideas and analyses

Teaching of transfer skills

You will practice designing and analyzing algorithms on selected examples during tutorials and homework.

Teaching of normative evaluation skills

In this class, you will

- Assess the quality of algorithms and algorithmic approaches under efficiency aspects
- Assess problems with respect to their algorithmic complexity

Key qualifications

For this class, we expect for you to

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework, and take part in class tutorials.
- use creative problem solving: for example, developing efficient algorithms.

Methods of implementation

For problems such as sorting or dynamic search structures, we will present and compare a wide range of algorithmic methods. For this purpose, the requirements for the necessary data structures are worked out. This approach also includes developing and analyzing different methods (for example, for search structures, priority queues, or union find structures). Such methods will teach you the mathematical methods for correctness and efficiency analysis.

Module assignment

Mandatory module in the field of Models and Algorithms (I.2).

Mode

- Credit points for each module (workload): 8
- SWS (4 lectures, 2 tutorials per week)
- Frequency of the class offered: every summer semester

- Duration (1 semester)

Organizational arrangements / media use / literature references

- Lecture class with data projector and use of the black board
- Tutorials in small groups
- Activities expected of the students: co-operation during tutorial classes, homework
- You will receive exercise sheets and sample solutions in the class tutorials
- Web-based lecture material: Summer semester 2001/2002: Friedhelm Meyer auf der Heide: http://www.uni-paderborn.de/fachbereich/AG/agmadh/vorl/DaStrAlg01/dua.html

Examination modalities

Written examination

Person responsible for the module

Meyer auf der Heide

I.2.3 Introduction to Computability, Complexity, and Formal Languages

Role of the class in the Degree Course

Modeling and analyzing problems and evaluating the solutions found are fundamental parts of computer science. Formal languages and grammars are important tools for problem modeling. Problems described through formal languages and grammars are accessible to mathematical analysis based on the distinction among different types of problems and the ability to compare the degrees of problem difficulty. The coarsest and most important distinction of problem types is that between problems that can in principle be solved on a computer and those that as a matter of principle cannot be solved on any computer. The class of problems that are solvable in principle is then classified further according to different complexity measures, such as time and memory requirements. The basis of this classification must always be models of computation that are both mathematically precise and realistic. You will apply the modeling and classifying concepts described during this class in both core areas in the University of Paderborn's computer science department. Modeling concepts find widespread application, in particular in software development; classification concepts form the basis for the core of algorithmics.

Content structure of the class

- Introduction
  Languages, models of computation, grammars, simulations
- Computability:
  Decidable and non-decidable languages, diagonalization, halting problem, reductions, examples
- Time complexity:
  Running times, P and NP classes, polynomial reductions, NP completeness, SAT, Cook-Levin theorem, examples
- Approximation algorithms and heuristics
  Approximation algorithms, approximation quality, examples, backtracking, branch-and-bound, local improvement
- Formal languages and grammars
  Types of grammars, relation to decidability, regular and context-free languages, finite automata, stack automata, pumping lemma

Usability of the content

The knowledge and abilities you acquire are applied and expanded in many areas, for example, in hardware and software design, computer graphics, operating systems, and information systems; and in the advanced classes on algorithms, complexity theory, cryptography, optimization, and parallel computing. The modeling concepts of formal languages and grammars are essential in the professional life of a computer scientist as are the concepts of decidability that form important background knowledge for the practice. The concepts from the algorithms and complexity areas are applied by every computer scientist working in algorithm design.

Prerequisites and prior knowledge

The modules *Modeling* and *Data Structures and Algorithms* have to be passed. Furthermore, the willingness and ability to describe intuitive concepts formally and to apply them to specific problems is a prerequisite.

Learning goals of the class

Teaching of factual knowledge

In this class, you will learn

- Concepts and methods of computability theory
- Formal languages, grammars, and the associated models of computation
- Concepts and methods of complexity theory and algorithmics

Teaching of methodological knowledge

In this class, you will

- Independently analyze and classify problems, develop hypotheses and the subsequent verification or falsification and reformulate the hypothesis
- Apply mathematical methods in analysis and classification
- Comprehend the basic structure of complexity statements
- Assess the complexity of problems using the complexity classes introduced during the lectures

Teaching of transfer skills

You will practise modeling, analyzing, and classifying problems on selected examples during tutorials and homework.

Teaching of normative evaluation skills

In this class, you will

- Develop Complex assessments of problems
- Assess solutions with respect to their practical usability

Key qualifications

For this class, we expect for you to

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework, and take part in class tutorials.
- use creative problem solving, using the example of individual modeling, analysis, and classification of problems.

Mode

- Credit points for each module (workload): 8
- Credit points of the class: 8
- SWS (4 lectures, 2 tutorials per week)
- Frequency of the class offered: every winter semester
- Duration (1 semester)

Module assignment

Mandatory module in the field of Models and Algorithms.

Methods of implementation

The class explains for different problems

- how they may be modeled using grammars and formal languages
- how they may be analyzed and categorized into the different complexity classes discussed during the lectures
- how references and comparisons to other problems can be established
- how the classification suggests or limits approaches for a solution
- how practical approaches to a solution may be found even if the problem is hard to solve.

Organizational arrangements / media use / literature references

- Lecture with data projector/overhead slides and black board use
- Tutorials in small groups
- Activities expected of the students: Co-operation during tutorial classes, homework
- Exercise sheets and sample solutions are presented in class tutorials
- Web-based lecture material: winter semester 02/03, summer semester 03: Johannes Blömer:
  - http://webserv.uni-paderborn.de/cs/ag-bloemer/lehre/bfs_WS2002
  - http://webserv.uni-paderborn.de/cs/ag-bloemer/lehre/auk_SS2003

Examination modalities

Written examination

Person responsible for the module

Blömer

I.3 Field: Embedded Systems and System Software

I.3.1 Foundations of Computer Engineering and Computer Architecture

Role of the module in the Degree Course

Students of computer science of any specialization should have a basic knowledge of the foundations of computer engineering and the basic principles of digital computer operation. Students wanting to specialize in software engineering and in fields such as compiler construction or system-level software areas quite obviously need this basic knowledge. Even for other specializations within computer science, computer engineering forms an important basis because of its modeling and solution techniques (such as, for example, Boolean algebra, automata theory, optimization processes in Boolean algebra and automata theory, arithmetic algorithms, caching principle, and parallel computing). Basic knowledge of digital computer operation is also essential in order to develop efficient software.

This module is mandatory for students of computer science and computer engineering. It consists of two classes, *Foundations of Computer Engineering* and *Foundations of Computer Architecture*. The first class places special emphasis on the modeling techniques of computer engineering. From these models, the methods of digital system design are derived. This class is thus seamlessly integrated into the conceptual design of the Computer Science Degree Course, which is quite significantly based on model development. The second class takes up on this modeling-based approach in order to develop step by step the operating principles of modern universal processors. Phenomenological aspects (description of actual processor architectures) are also dealt with, but solely for illustration of the principles.

Content structure of the module

The module consists of the classes *Foundations of Computer Engineering* and *Foundations of Computer Architecture*. The *Foundations of Computer Engineering* class provides insight into modeling combinatorial circuits (Boolean algebra) and sequential circuitry (finite transforming automata). We discuss optimization processes for both cases, and, based on these models, describe basic structures of digital circuits. In addition, there is a brief introduction to the underlying semiconductor technology and to the techniques of connecting continuous systems. Alternative forms of number representation and the arithmetic algorithms based on them are introduced. The basic approach in the design of digital systems concludes this class. It is rounded off with lab work that mostly deals with simulation via VHDL.

The *Foundations of Computer Architecture* class first teaches basic knowledge about the operating principle of a von Neumann computer. This teaching is done on the basis of a simplified MIPS architecture. The basic principle thus introduced is now refined step by step until the principles of modern computer architecture are covered. This approach addresses the following aspects: information storage (memory hierarchy), information access (addressing techniques), information transport (bus systems), access to remote information (I/O, interrupts), and parallel information processing (pipelining). The concepts are illustrated using current processor architecture (Pentium as an example for CISC, PowerPC for the RISC approach).

Usability of the content

The principles of computer engineering are found in wide areas of computer science. The knowledge gained from this mandatory module is thus widely applicable. Students intending to specialize in computer engineering, in particular in embedded systems, are equipped with essentials. They will also acquire important foundations for system-level software development. Developers of application software and of software development processes gain an understanding of the underlying processor architectures -- an understanding that is essential for developing efficient software and its design processes.

Prerequisites and prior knowledge

The content of *Modeling* is recommended as prior knowledge. Otherwise, only basic mathematical knowledge is expected, which should be covered by the general university entry qualification.

Examinable standards / learning goals of the module

The students should acquire an understanding of the specific features of digital systems, in particular of processors, and become familiar with the basic concepts for designing such systems. The students should understand the methods used in modeling such systems and the optimization processes that are based on them. They should be able to assess the specific restrictions that arise from the physical laws of technical systems and should learn to integrate them systematically into the design process. Finally, they should comprehend how the restrictions resulting from the digital technology and specific computer architectures affect higher abstraction layers, in particular in software technology.

Teaching of factual knowledge – content competency

In this module, you will learn

- Modeling digital technology system components
- Design technologies for digital systems
- Basic principles of processor architecture
- Interaction between software and hardware

Teaching of methodological knowledge – methodological skills

In this module, you will develop methods for

- modeling combinatorial systems
- modeling sequential systems
- optimizing complex systems
- parallel processing
- designing digital systems

Teaching of transfer skills

Transfer of the global strategies to specified individual situations, for example as part of exercises

Teaching of normative evaluation skills

In this module, you will

- Develop techniques for applying different strategies
- Recognize the practical value of the concepts and methods of processor architectures

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Module assignment

Mandatory module

Mode

- Credit points for each module (workload) : 10
- SWS (GTI: 2 lectures, 1 tutorial, 1 lab per week, GRA: 2 lectures, 2 tutorials per week
- Frequency of the class offered: GTI: every summer semester, GRA: every winter semester
- Duration (2 semesters)

Methods of implementation

Tutorials in small groups foster the practical application of the methods presented to selected examples. In particular, parameters and strategies must be adapted to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups including demonstrating the calculations for tutorial exercises and the sample solutions for the homework
- Activities expected of the students: co-operation during tutorial classes, homework, independent study of secondary literature
- Overhead script is available on the class homepage

Examination modalities

Written examination in *Foundations of Computer Engineering*

Written examination in *Foundations of Computer Architecture*

Person responsible for the module

Platzner

I.3.2 Concepts and Methods of System Software

Role of the class in the Degree Course

When looking at the teaching content of classic areas in computer science, such as compiler construction, operating systems, database systems, computer networks, distributed systems or computer architecture, one can see that there are fundamental problems that reoccur as variants in the different fields and are solved there by respective methods. This approach suggests extraction of these individual phenomena from their context to establish their commonalities and to discuss them fundamentally as general phenomena.

The aim of this class is to teach general principles, concepts, methods, and technologies that occur in complex hardware/software systems with concurrent processing. The students should be able to recognize the commonalities and comprehend the principles as being fundamental to the subject. In particular, they should be able to apply these methods sensibly in design situations because the class bridges computer engineering and practical computer science.

We introduce the basic components of the system software based on the foundations of computer architecture. After a review of the most important components of computer architecture, we introduce processes that form a functional and structuring description unit for system and application software. The concurrent and parallel execution of processes plays a significant role for the efficiency of the entire system and is crucial for the load on the resources. However, concurrent processing requires using synchronization concepts, based on locks, semaphores, critical sections, and transactions in order to manage interaction between processes and resource access. An examination of the basic techniques of transaction management and how to ensure desired features such as rollback, strictness, and restoration establishes a link to databases. In particular, general methods for resource management are introduced that are based on centralized, co-operative, and optimistic technologies. Technologies for detecting and preventing deadlocks conclude resource management. Subsequently, the methods introduced are explored in detail in the context of memory management and scheduling. The core topics include handling logical and virtual resources, memory hierarchies, virtualization, caching, and strategies based on the locality principle. In the case of scheduling, we explain the process planning for conventional processes, real-time systems, and dependent processes. The last part of the class deals with process interaction beyond computer limits and introduces the basic concepts of channel, bridge, and remote procedure calls.

The class provides the basics for advanced classes such as those on operating systems, distributed systems, computer networks, real-time systems and in part also for databases and compiler construction. The class is particularly suitable for students of computer engineering. For students in business computing with an interest in technological questions, the class offers a complete overview over the basics of system software.

Content structure of the class

The class is structured into three large parts: Foundations of system software, resource management, and inter-process communication. The time schedule of the class is divided into the following seven chapters:

- Foundations of computer architecture
- Processes and concurrency
- Process scheduling

- Process synchronization and transactions
- Resource management and deadlocks
- Memory management
- Co-operative process interaction

Usability of the content

Typical application for contents and methods in the class are mainly found in operating systems. The actual mechanisms for memory management or scheduling in modern operating systems are derived from the basic methods presented. The techniques for resource management are required in almost all areas of computer science, for example in designing and implementing efficient, real-time capable system software. The synchronization mechanisms are required for transaction management in databases. The parallel and concurrent processing is essential in powerful and/or fail-safe servers. Last but not least, the concepts of bridge and channel form the foundation for the practical implementation of network communication and remote procedure calls that are required for web-based information systems.

Prerequisites and prior knowledge

Students must have a basic knowledge of programming languages and computer architecture. We also expect a willingness to explore the relationships between resources and computer architecture and to internalize the basic principles of efficient software development. In particular, the student should independently apply the global concepts and methods to specific examples.

Examinable standards / learning goals of the class

Teaching of factual knowledge – content competency

In this class, you will learn the

- Relationship between hardware and system software
- Structure, management, and synchronization of processes
- Techniques for memory management and scheduling
- Techniques for securing critical areas
- Techniques for designing parallel and concurrent programs

Teaching of methodological knowledge – methodological skills

In this class, you will develop

- Methods for efficiently managing and allocating resources
- Methods for detecting and avoiding deadlocks
- Methods for the co-operation between processes in distributed systems
- Process interaction methods

Teaching of transfer skills

Transfer of the global strategies to specified individual situations, for example as part of exercises

Teaching of normative evaluation skills

In this class, you will

– Develop techniques for applying different strategies
– Recognize the practical value of the concepts and methods of system software

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Module assignment

Mandatory module: *Concepts and Methods of System Software* (I.3.2)

Mode

- Credit points for each module (workload): 8
- SWS (4 lectures, 2 tutorials per week)
- Frequency of the module offered: every summer semester
- Duration (1 semester)

Methods of implementation

Tutorials in small groups foster the practical application of the methods presented to selected examples. In particular, parameters and strategies must be adapted to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups. This includes the demonstration of the calculations for tutorial exercises and the sample solutions for the homework
- Activities expected of the students: co-operation during tutorial classes, homework, independent study of secondary literature
- Overhead script is available on the class homepage

Examination modalities

- Written examination
- The points achieved for the homework may be used to improve the grade of the written examination. If more than 35% of the points are achieved, the grade improves by 0.3; if more than 60% are achieved, the grade improves by 0.7, and if more than 90% are achieved, the grade is improved by 1.0. These specifications serve as a guide only. They are adapted every semester and are published in class. The bonuses only apply if the written examination is passed.

Person responsible for the module

Karl

I.4 Field: Human-Machine Interaction

I.4.1 Basics of Human-Machine-Interaction

Role of the module in the Degree Course

Today, the key demands for product design in computer science are learnability and restriction-free usage of software systems as well as fully accessible information from the World Wide Web. Taking this into account, it is important to support the understanding process of users on the one hand and, on the other hand, to avoid unnecessary workload when working with software systems. Furthermore, a methodical repertoire is necessary to ensure usability even during the design. This requires a lot of basic skills that range from legal requirements and physiological and psychological foundations to methods and techniques of system design.

The students will learn to recognize basic problem areas of Human-Machine-Interaction and to solve them by constructively designing. At the same time, this will teach the students to obtain follow-up knowledge which is especially required for the cooperation with designers and ergonomists but is also helpful for the discourse with media scientists and teachers (e-Learning). At the same time, the acquired knowledge and skills are a basis for the deepening classes in the field of Human-Machine-Interaction like, for example, Usability Engineering, Computer Graphics and Media Ergonomics.

Content structure of the module

Among other things, the students will acquire knowledge and skills from the following fields:

- Legal rules and norms:
  - EU guideline for working at a computer (90/270/EWG), EU guideline for workstations (BildscharbV), fully accessible information technique guideline (BITV), …
- Basics of cognitive psychology:
  - Realization, attention, memory
- Physiological basics:
  - Sensomotorics, motor activity, …
- Concepts:
  - Interaction technology, color models, interreference
- Design recommendations:
  - Norms (i. e. DIN EN ISO 9241) guidelines, criteria, …

- Methods:
  - Determination of requirements, modeling, evaluation

Usability of the content

The acquired knowledge and skills are helpful and necessary in every field of application when designing interactive systems. This applies to the work with visual display units as well as to the development of games or digital media.

Prerequisites and prior knowledge

Basic knowledge in the development of software systems.

Learning goals of the class

The students should be able to realize usage problem areas of software systems and to evaluate their social status on the basis of legal regulations. Furthermore, they should dispose of a repertoire of solution approaches for designing ready-to-use-systems. For this reason, students should also dispose of a methodical repertoire that allows them to define, assess and at least evaluate heuristically the requirements of future users.

Teaching of factual knowledge – content competency

Basic technological concepts are combined with concepts of human cognition.

Teaching of methodological knowledge – methodological skills

Apart from requirement determination methods, the students will learn which techniques and methods (paper prototyping, for example) are suitable for developer-user-communication.

Teaching of transfer skills

Some basic concepts and technology are generally also applicable to other fields of the design of software systems like, for example, the awareness problem in computer aided systems, the visualization of information or tools for knowledge processing.

Teaching of normative evaluation skills

The students will learn basics of Human-Machine-Interaction that will enable them to solve standard problems on the one hand and, on the other hand, to identify areas that require special scientific knowledge.

Key qualifications

Expected contribution of this class to teach key qualifications

- Ability to cooperate and work as part of a team through participation in tutorial classes.

- Presentation skills by acquiring the corresponding design basics.
- Ability to assess latest IT-technology.
- Follow-up knowledge for interdisciplinary cooperation.

Mode

- Credit points for the module: 4
- SWS: (2 lectures + 1 exercise)
- Frequency of the module offered: This module is offered every summer semester
- Duration: One semester.

Methodological Implementation

- Methods and concepts are introduced in a lecture.
- The methods and techniques are applied to new, typical example programs during exercises in small groups.

Organizational arrangements / media use / literature references

- Lecture with textbooks or scripts, respectively overhead slides.
- Exercises: Tutorial classes in small groups with exercise sheets and homework.
- Expected contribution from the students: Collaboration in tutorial classes, homework.
- Standard text books, teaching materials on the www.

Examination modalities

Written examination

Person responsible for the module

Reinhard Keil

I.5. Mathematics

I.5.1 Analysis

Role of the module in the Degree Course

Introduction to the mathematical basics – especially analysis - that are required for the Computer Science Degree Course.

Content structure of the module

The module consists of the lecture *Analysis* (for computer scientists)

Structure for Analysis (4 lectures + 2 tutorials per week)"

Chapter I: Basic terms

1. Sets and maps
2. Induction and recursion, combinatorics
3. Elementary number theory
4. Real numbers, fields
5. Complex numbers

Chapter II: Analysis

1. Convergence of sequences
2. Convergence of series and power series
3. Continuity
4. Exponential function and trigonometric functions
5. Polar co-ordinates, unit roots and the fundamental theorem of algebra
6. Differentiability
7. Local extrema, Taylor formula, Taylor series
8. Integrality (Riemann integral)
9. Approximation of roots and fix points. The Newton method.

Usability of the content

The students will apply the methods and factual knowledge that they acquired in these two classes during their Computer Science Degree Course. Likewise, the students will apply mathematical-methodical thinking (definition, theorem, and proof) that are practiced in these classes.

Prerequisites and prior knowledge

No special prior knowledge is required.

Learning goals of the class

The students

- describe the progressive construction of the numeric system (including complex numbers) and argue it by using the permanence principle as a formal guideline.
- are formally able to use terms of convergence of series and sequences as well as completeness of real numbers reliably and explain these terms with corresponding examples.
- describe the terms constancy and differentiability illustratively and formally; justify fundamental statements on constant and differential functions, use the idea of approximation by power series to describe the functions.
- formally define the term integrality and use it in a mathematical context; interpret integrality for area measurement and for determining the mean value.
- describe and proof the main clause of differentiability and integrality.
- use software to present and explorate mathematic models and as a heuristic tool to solve user problems.
- know and reflect questions regarding the conversion of numeric methods at the computer (complexity, exactness).

Key qualifications

The students

- present and explain mathematic facts
- think conceptionally, analytically and logically
- think and act independently
- work out independently mathematic insights motivated by their own interest

Module assignment

- Mandatory module

Mode

- Credit points: 8 credit points
  - Extent and type of the class offered: 6 SWS (4 lectures and 2 tutorials)
  - Frequency of the module: This module is offered every winter semester for a duration of one semester.

Examination modalities

Written examination

Person responsible for the module

The lecturing staff in Mathematics

I.5.2 Linear Algebra

Role of the module in the Degree Course

Introduction to the basics of linear algebra which are required for the Computer Science Degree Course.

The main focus of linear algebra is on solving linear systems of equations in a practical and theoretical way on different term levels and, furthermore, the concept of linearity as a universally applicable mathematical solution tool. The role of this tool in the subsequent Degree Courses is the importance of linearity (or linear approximation) in all fields of mathematics, in mathematical modeling and in mathematical applications.

Content structure of the module

The module consists of the lecture Linear Algebra (for computer scientists)

Structure for *Linear Algebra* (4 lectures + 2 tutorials per week)"

1. Basic terms
2. Vector spaces
3. Linear mappings
4. Foundations
5. Dimensions
6. Matrix
7. Linear systems of equations
8. Determinants
9. Eigenvalues
10. Characteristical polynomial
11. Problem of normal forms

Usability of the content

The students will apply the methods and factual knowledge that they acquire in these classes during their Computer Science Degree Courses.

Prerequisites and prior knowledge

Knowledge of the contents of module 1.5.1 *Analysis* is absolutely essential.

Learning goals of the class

The students

- understand and explain how abstract vector spaces are realized as coordinate-free generalizations of one- to three-dimensional spaces and show understandable examples from mathematics and from fields of application in this conceptional framework.

- understand linear mappings of vector spaces as structure friendly mappings and explain how they can describe linear systems of equations free of coordinates.
- understand the abstract terms of foundations and dimensions and explain how they can be understood as a generalization of the naïve terms of coordinates and dimensions.
- realize linear mappings in a matrix and understand them as a coordinate-subordinate realization.
- understand and explain how the (definite) proof of such systems can be characterized, solve linear systems of equations and explain solution procedures.
- understand the determinants as an alternating multi-linear form and explain it by using its geometrical significance; comprehend its meaning for the inversion of matrix and know the methods for its determination.
- know the term *eigenvalue*; understand and explain the problem of normal forms; know the criteria for diagonalization.

Key qualifications

The students

- reflect their own learning experiences
- present and explain mathematical facts
- think conceptionally, analytically and logically
- work out independently mathematical insights motivated by their own interest
- think and act independently

Module assignment

Mandatory module

Mode

- Credit points: 8 credit points
- Extent and type of the class offered: 6 SWS (4 lectures and 2 tutorials)
- Frequency of the module: This module is offered every summer semester and lasts one semester.

Examination modalities

Written examination.

Person responsible for the module

The lecturing staff in Mathematics

I.5.3. Stochastics

Role of the class in the Computer Science Degree Course

Introduction to the foundations of stochastics that are required during the Computer Science Degree Course.

Content structure of the class

Descriptive statistics and data analysis, classical models of probability, axiomatics, standard distributions (i.e. Binomial), formulas (Bayes) and applications, examples for non-discrete probability, random variables and its moments, quantiles, law of large numbers, central limit theorem, estimates (incl. confidence intervals) and testing, simulation and random numbers, Markov inequality, multi-dimensional probability estimates.

Descriptive Statistics and Data Analysis

- Students plan, conduct and evaluate statistical investigations (interviews, observation or experiments).
- They read and draw up graphical presentations for plain- and bivariate data (e.g. cross tables) and evaluate their suitability for the individual problem.
- They determine and apply plain- and bivariate values (e.g. mean value, distribution mass, correlations, index values) and interpret them accordingly.

Random Modeling

- Students model multi-step random attempts through finite sets and apply suitable depictions (tree chart, multi field table).
- They calculate and argue with probabilities, conditional probabilities, expected values and stochastic independence.
- They explain the Bernoulli law of large numbers and the central limit theorem and its consequences.
- They use discrete and continuous distribution and its qualities for modeling.

Stochastic Applications

- Students know examples for the application of stochastics in different sciences (economy, physics, …).
- They estimate parameters from data in random situations.
- They apply hypothesis tests and reflect its central steps and determine confidential intervals.
- They explain the differences between Bayes statistic and classical testing methods.

New Media

- Students use spreadsheets and statistical software for presentation and for explorative analysis of data.
- They simulate random experiments at the computer.

Usability of the content

You will use the methods or factual knowledge that you acquired in these two classes *during your Computer Science Degree Course*.

Prerequisites and prior knowledge

The modules *Analysis* and *Linear Algebra* are a prerequisite.

Learning goals of the class

The students should

- Obtain knowledge of the significance of stochastics in society and science.
- Be able to apply written and oral terms of stochastics.
- Understand mathematical situations and the corresponding ways of thinking.
- Understand proofs; be able to solve stochastic exercises, be able to realize connections within stochastics and between other mathematical fields.
- Perform simple statistical analysis, be able to use a stochastics software package.

Key qualifications

The students should

- reflect their own learning experiences;
- present and explain mathematical facts;
- think conceptionally, analytically and logically;
- work out independently mathematical insights driven by own interests;
- think and act independently.

Module assignment

- Mandatory module

Mode

- Credit points per module (workload): 6 credit points.
- Extent and type of the class offered, 5 SWS (3 lectures and 2 tutorials), for example.
- Frequency of the module: This module is offered every winter semester and lasts one semester.

Examination modalities

Written examination or oral examination.

Person responsible for the module

Will be announced on the homepage of the Institute of Mathematics

II. Modules in the second stage of the Bachelor Degree Course

II.1 Field: Software Technology and Information Systems

II.1.1 Software Technology and Information Systems

Role in the Degree Course

Developing, commissioning, and maintaining software systems are among the most important tasks for today's computer scientists. The biggest difficulty in these tasks is to manage the size and complexity of contemporary and future software systems. These tasks are complicated further by the need to have software and hardware precisely adapted to each other in certain areas. To master this challenge, computer scientists require a wide range of knowledge and skills in software technology and information systems.

Based on the fundamental concepts and methods of software engineering and the practical experiences from the Software Engineering Lab in the first stage of the Degree Course, this module gives a broad overview of the most important concepts, notations, and methods of software technology and its formal and mathematical foundations. The knowledge acquired should enable the students to develop software systems under given technical and economic conditions. In addition, students should have a command of the scientific tools that enable them to familiarize themselves with future techniques during their professional life.

The classes in this module cover various sections of *Software Technology and Information Systems* (II.1). The selection provides a representative overview of the entire field and different phases of software development.

Students may expand what they learn in these classes into different specialties later in the Computer Science Master Degree Course by selecting mandatory elective modules in the field of software technology.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Model-Based Software Development (MSWE)
- Programming Languages and Compilers (PSÜ)
- Principles of Knowledge-Based Systems (*GWBS*)
- Formal Methods in Software Design (SMFM)
- XML Databases – in English (XMLDB)
- Data Mining

The content of the class *XML Databases* is almost the same as the content of the previous classes *Processing, Indexing and Compression of Structured Data (PICSD)* and *Databases and Information Systems I*. For this reason, you may only choose one of these three classes for a module examination.

The aims and the content of the individual classes are described below.

Model-Based Software Development (MSWE)

Students should be familiar with basic methods for constructing large software systems and should master their use. They should also experience the advantages and disadvantages of formal and informal specification techniques, recognize the need for design, and be able to apply abstract models to improve software quality. This class places particular emphasis on the paradigm of the "Model Driven Development" that promises a substantial productivity and quality gain in software development.

Content:

1. Specification techniques for analysis and design:
   Structure-oriented, operational and descriptive techniques
2. Automatic code generation from the design
3. Validation and verification of software systems:
   testing and model checking

Programming Languages and Compilers (PLaC)

Languages play various important roles in software technology: As programming languages, they are a means of expression for program development and are tailor-made for a specific programming method. As specification languages, they are used for formulating task descriptions in general, or are tailor-made description methods for specific application areas. Designing and implementing such languages by means of compilers or generators are important topic areas in software technology.

This class teaches knowledge and skills for in-depth understanding, specifying, and implementing programming and specification languages. The participants learn to

- apply basic calculi for the precise description of language features,
- apply basic methods for the implementation of languages.

Content:

1. Levels of language features and the structure of compilers
2. Specification of basic symbols and lexical analysis
3. Syntactic specification and analysis
4. Semantic features and analysis
5. Specification of dynamic semantics and compilation

Principles of Knowledge-Based Systems (Introduction to Artificial Intelligence)

Introduction to Artificial Intelligence presents basic knowledge for deduction as well as methods of symbolic knowledge processing. Important aims for the class are teaching limitations and possibilities of well-established knowledge representation formats and introducing their formal basis.

The knowledge and skills gained in the class should enable the students to develop software systems that must take aspects such as uncertainty and vagueness into account, or that are to emulate human problem solving behavior.

Content:

1. Artificial intelligence (term, history, fields)
2. Knowledge formats (sub/symbolic, problem solution), expert systems
3. Deduction in statement logic and decision making problems
4. Deduction in predicate logic
5. Production rule systems

6. Fuzziness and vagueness (for example, fuzzy logic)
7. Classification of automatic learning algorithms

Formal Methods in Software Design (SMFM)

Formal methods are languages for specifying software systems on an abstract level. Being equipped with a formal semantics, models written in formal specification languages can be analyzed for correctness which is in particular important for safety critical systems.

This class teaches knowledge and skill for abstractly modeling software systems and analyzing their properties. The class should enable the students to choose an appropriate modeling language and associated analysis technique.

Content:

1. Introduction to formal specification languages
2. Modeling parallel communicating and timed systems
3. Analysis techniques
4. State-based specification languages

XML Databases - in English (XMLDB):

Content:

- XML as a data exchange format
    - XML, DTD, XML scheme
- Search and Navigation in XML documents
    - DOM, SAX, StAX, XPath
- XML storage and core XPath implementation
    - XPath queries on XML data streams
- XML compression
    - XMill, Succinct, DAG, RePAIR, Schema subtraction, Exalt
- Numbering Schemes
    - OrdPath, Pre/Post, DDE, DLN

Data Mining - in German:

The class *Data Mining* introduces the student to knowledge establishing procedures using certain methods to systematically search for patterns in data. The main emphasis is put on efficient algorithmic approaches for a potentially very high data stock.

Content Structure:

1. Introduction

2. Distributed processing of large data

3. Similarity search

4. Data mining in data flows

5. Link-analysis

6. Itemset mining

7. Cluster analyses

8. Recommender systems

9. Network analysis

10. Dimensionality reduction

Usability of the content

Students completing this module should be able to develop software and information systems in accordance with contemporary techniques, acquire new techniques, and evaluate them.

The class Introduction to Artificial Intelligence deals more with the general formal and mathematical basics of software technology, whereas the classes MSWE, PSÜ, *Formal Methods in Software Design* (SMFM) and XML DB deal more specifically with the concepts and methods of the respective field.

The classes XML DB and *Introduction to Artificial Intelligence* predominantly cover information systems, while MSWE, PSÜ and SMFM predominantly cover software technology.

Prerequisites and prior knowledge

A prerequisite for the successful completion of this module is the ability to model and formalize facts as taught, among others, in the *Modeling* module (I.2.1). In addition, command of a programming language and the usual notations of object-oriented modeling are required, as are first experiences in software development as they are taught in the modules *Foundations of Program Development* and *Software Engineering* and in the *Software Engineering Lab.*

Learning goals

Teaching of factual knowledge

After completing this module, you will

- have a broad overview over the fundamental concepts of programming and software technology
- know the commonly-used principles, notations, and languages for modeling and software development
- know the problems that occur during software development and should know methods and procedures for solving them

Teaching of methodological knowledge

After completing this module, you will

- be able to apply fundamental methods for formalization and modeling
- be able to apply fundamental methods of software development

Teaching of transfer skills

After completing this module, you will

- be able to acquire and evaluate new methods and notations in software technology
- be able to define and formulate new methods and concepts in software technology

Teaching of normative evaluation skills

After completing this module, you will

- recognize the necessity of systematic software development

Key qualifications

For this module, we expect for you to

- cooperate and work in teams during the tutorial classes;
- apply strategies for acquiring knowledge;
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework, and take part in class tutorials;
- evaluate and question new concepts.

Module assignment

Mandatory module of the 2nd stage in the field of *Software Technology and Information Systems*.

Mode

Credit points:	4+4 ECTS (4 for each class)
SWS:	2 lectures + 1 tutorial, 2 lectures + 1 tutorial

Frequency:

All classes of this module are offered in an annual cycle: The classes XML DB, PSÜ, and MSWE are probably going to be offered in every winter semester, the classes SMFM and *Introduction to Artificial Intelligence* in the summer semester.

Methods of implementation

- Methods and techniques are introduced and discussed using typical examples
- They are tested in practice with computer and tool support during the tutorials

Organizational arrangements / media use / literature references

- Lecture with overhead presentation or writing on the blackboard
- Materials on the Internet complementing the lecture
- In the tutorials, the problems are solved collectively
- Activities expected of the students:
    - co-operation during problem solving in tutorials,
    - homework, preparation and review of the lectures

Examination modalities

Oral examination or written examination in each of the two classes.

Person responsible for the module

Kastens

II.2 Field: Models and Algorithms

II.2.1 Models and Algorithms

Role of the class in the Degree Course

Algorithms form the foundation of every hardware and software: A circuit converts an algorithm into hardware, a program makes an algorithm "understandable to the computer". Algorithms thus play a central role in computer science.

For this reason, classifying problems with regard to their algorithmic complexity takes center stage in the Bachelor module *Models and Algorithms*. Running time and memory requirements in particular are used as measures of complexity, but also for example, parallelizability. Classes of this module discuss both developing and analyzing efficient algorithms and algorithmic techniques and investigating the complexity inherent in the problem, that is, the proof of lower complexity limits and the complexity comparison of problems. The module is further complemented by a class on cryptography. The inherent difficulty of problems that the complexity theory attempts to prove is used to an advantage here, for example, in the design of secure encryption algorithms.

Content structure of the class

For this module, the students choose two classes from the following catalogue.

- *Fundamental Algorithms*
- *Complexity Theory*
- *Algorithm Design*
- *Parallelism and Communication*
- *Optimization*
- *Introduction to Cryptography*
- *Distributed Algorithms and Data Structures*

In general, students may elect any two of these classes to successfully complete the module. Fundamental Algorithms, However, we recommend Complexity Theory, Cryptography, and Parallel Computing and Communications as introductory classes. Here, students should select either Fundamental Algorithms or Complexity Theory. If possible, students should only take Optimization and Methods of Algorithm Design after Fundamental Algorithms.

The structure of the class content is as follows:

1. *Fundamental Algorithms*

- Graph algorithms such as shortest paths, flows in networks (basics) and matchings
- Universal and perfect hashing
- String matching

2. *Complexity Theory*

- Hierarchy theorems
- Gödel´s incompleteness theorems
- P completeness, NP completeness, and PSPACE completeness

- Comparisons between complexity classes

3. *Algorithm Design*

- Introduction to online algorithms, randomization, and approximation
- Optimization heuristics

4. *Parallelism and Communication*
5. *Optimization*
6. *Introduction to Cryptography*

- Tasks in cryptography
- Symmetric and asymmetric methods
- Elementary security concepts and cryptanalysis
- The symmetric codes DES and AES
- Hash functions and MACs
- Diffie-Hellman key exchange protocol and RSA

7. *Distributed Algorithms and Data Structures*

- Network theory
- Routing and scheduling
- Hashing and caching
- The continuous-discrete method
- Anchored and decentralized data structures
- Distributed computing
- Distributed search structures
- Distributed heaps
- Safety and robustness

Usability of the content

The ability to design not just any algorithms, but to design resource-conserving, efficient algorithms for specific problems and the ability to assess problems with regard to their inherent complexity are important for many sub-fields of computer science. Databases and information systems, communication protocols and resource management in computer networks, computer graphics systems and scientific computation are important examples. In many applications, there are optimization problems to be solved efficiently. The class Optimization discusses this aspect in detail. The contents of Cryptography are applied in e-banking and e-commerce, among others.

Prerequisites and prior knowledge

Students must understand the basic concepts of algorithms, data structures, computability, and complexity theory, as they are taught in the first four semesters. The dependencies among the classes of the module are explained in the content structure section.

Learning goals of the class

Teaching of factual knowledge

- Concepts and methods of complexity theory and algorithms.
- Methods for the design of distributed and parallel algorithms.
- Fundamental concepts and methods of optimization.
- Cryptographic procedures and fundamental terms of safety.

Teaching of methodological skills

- Application of mathematical methods to analyze and classify algorithmic problems.
- Complexity analysis of problems and determination of problem complexity in accordance with essential complexity classes.
- Independent acquisition of new algorithms, data structures and algorithmic ideas and analysis.
- Determination of the quality of algorithms and algorithmic approaches considering different aspects of efficiency.
- Design of simple safety analysis and evaluation of the safety of cryptographic procedures.

Teaching of transfer skills

- The ability to work out and apply new methods and concepts of algorithm, complexity theory and cryptography.

Teaching of normative evaluation skills

- Evaluation of the quality of algorithms and algorithmic approaches considering different efficiency aspects.
- Evaluation of problems considering their algorithmic complexity.
- Evaluation of the safety of cryptographic primitives.

Module assignment

Mandatory module in Models and Algorithms.

Mode

- Credit points: 4 + 4 ECTS (per class)
- SWS 2+1, 2+1
- Frequency of the class offered: Every semester, between 2 and 4 classes from the catalog are offered.

Examination modalities

Oral examination or written examination in each of the two classes.

Person responsible for the module

Blömer

II.3 Field: Embedded Systems and System Software

II.3.1 Embedded Systems and System Software

(The following description of the module is currently being revised)

Role of the module in the Degree Course

(The following text is currently being revised)

Embedded systems and system software play a central role because of the continuous computerization of all technical systems. System software comprises every fundamental software layer that connects the computer hardware to the software. Together with other components of the system software, it enables developing applications and provides an interface to the hardware resources. Computer networks are of particular importance in this context, as they establish links between spatially separated resources and provide the basis for developing distributed systems. The latter include, among others, web-based services, co-operative applications, and efficient and fail-safe processing. In all cases, however, it is necessary that implementing for the user is carried out as transparently, reliably, and securely as possible.

One of the important areas for designing and applying system software is in embedded systems. This importance refers to the information processing components in systems, which usually consist of dedicated hardware and software that builds on it. Both are designed using the fundamental methods of computer science. The interaction between HW and SW plays a particularly significant role here and is discussed under the aspect of HW/SW codesign. Computer scientists must take the physical laws of the entire system into account, including, apart from real-time requirements, resource limits (for example, with respect to power consumption or chip space available). As a consequence, the general design cycle of computer science systems must be adapted to the specific case in all phases. This adaptation requires case-specific specification and modeling techniques.

This module builds on the foundations presented in the modules *Concepts and Methods of System Software* and *Computer Engineering* and introduces students at the Bachelor stage to *Embedded Systems and System Software*. The classes of this module ensure as broad an overview as possible. They may roughly be classed into SW-intensive (Distributed Systems and Computer Networks) and HW-intensive (Embedded Systems and HW/SW Codesign) aspects. Because students can only select some of the classes in this module, some classes are also offered in individual modules for the Masters stage. The introduction to the ESS area can thus be covered completely. However, we encourage a targeted selection of classes according to their own interests because this knowledge can lead to a higher degree of specialization in certain parts of the Masters stage.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Networked Embedded Systems
- Computer networks
- Distributed Systems
- Embedded Processors

The classes are structured as follows:

- Networked Embedded Systems
  1. Design and architecture of embedded systems
  2. Sensor networks
  3. Wireless communications
  4. Wireless access
  5. Routing
  6. Cooperation and clustering
- Computer Networks
  1. Introduction
  2. Physical layer
  3. Safety layer
  4. Media access control
  5. Internetworking
  6. Routing
  7. Overload defense
  8. Transport layer
- Distributed Systems
  1. Introduction
  2. Simple interaction patterns
  3. Advanced interaction patterns
  4. Time in distributed systems
  5. Distributed algorithms
  6. Replication and consistency
  7. Case studies
- Embedded Processors
  1. Introduction: Instruction set architectures, embedded processors, design objective
  2. Processor architectures: General-Purpose processors, digital signal processors, microcontroller, ASIPs; FPGAs and ASICs by comparison; case studies, TI DSP C55x and ARM
  3. Compiler and Code Generation: Compilerstructure, interim code, code optimization, code generation for specialized processors, retargetable compiler
  4. Processor, Performance and Energy: Performance metrics, worst-case execution time analysis, energy metrics, energy minimizing techniques

Usability of the content

(The following text is currently being revised)

Students will typically find the class content in application development, system administration, and in designing and implementing special systems. The mechanisms presented for resource management, security, and cross-platform communication are applied both in classic information systems and -- in an adapted version -- in special hardware resources. The methods presented for specification, modeling, analysis, synthesis, and verification are required in the entire area of technical systems. However, real-time applications are also applied in a non-technical environment, for example, for weather forecasts or for the strategic planning of financial services. The foundations of computer networks and the basic building blocks for developing distributed systems are required for Internet applications, web services, enterprise software, and so on. With the knowledge gained, the student should be able to assess, select, and adapt different solution methods and solution components to a specific task.

Prerequisites and prior knowledge

(The following text is currently being revised)

Prerequisites are the contents of the modules *Computer Engineering* and *Concepts and Methods of Systems Software*. In addition, basic knowledge of modeling principles from the module *Modeling* and of programming languages from the module *Programming Techniques* are mandatory. Students are expected to become familiar with system-level programming languages and hardware description languages.

Examinable standards / learning goals of the module

(The following text is currently being revised)

Upon completing the module, students will understand the specific features of system software and become familiar with the basic building blocks for developing operating systems and distributed systems. Students should also learn basic concepts and different operating principles of computer networks and their use, and thus understand that it is possible to transfer the basic concepts to new network structures and technologies. In embedded systems, students will understand the specific features of embedded systems and will learn the elementary concepts for designing such systems as mixed HW/SW implementations. They will also learn criteria for the HW/SW partitioning. They will be able to assess the specific restrictions that result from the physical laws of the surrounding system and learn to include them systematically into the design. Finally, students should learn how to combine specific methods from both software technology and hardware design to achieve a powerful design methodology.

Teaching of factual knowledge – content competency

After completing this module, you will know the

- Relationship between hardware and system software
- Structure, management, and synchronization of processes
- Techniques for memory management and scheduling
- Techniques for securing critical areas
- Techniques for designing parallel and concurrent programs

Teaching of methodological knowledge – methodological skills

After completing this module, you will understand the

- Methods for efficiently managing and allocating resources
- Methods for detecting and avoiding deadlocks
- Methods for the co-operation between processes in distributed systems
- Process interaction methods

Teaching of transfer skills

Transfer of the global strategies to specified individual situations, for example as part of exercises

Teaching of normative evaluation skills

After completing this module, you will be able to

- Develop techniques to apply different strategies
- Recognize the practical value of the concepts and methods of system software

Key qualifications

Activities expected of the students: co-operation during tutorial classes, homework, independent study of secondary literature.

Module assignment

Mandatory module in Embedded Systems and System Software

Mode

- Credit points for each module (workload): 6
- Credit points of the class: 3 each
- SWS (2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week)
- Frequency of the classes offered: "*Embedded Processors*" is offered in the summer semesters only, the remaining 3 classes are offered in the winter semesters only
- Duration (2 semesters)

Methods of implementation

Tutorials in small groups foster the practical application of the methods presented to selected examples. In particular, parameters and strategies must be adapted to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups. This includes the demonstration of the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Oral examination or written examination in each of the two classes.

- In general, there are individual examinations for each class with subsequent calculation of the average of the two individual results. The examinations for the individual classes in the module are carried out either as written examinations or as oral discussions, depending on the number of participants. Deviations from this scheme and additional requirements are specified when the respective class is announced.

Person responsible for the module
Karl

II.4 Field: Human-Machine Interaction

II.4.1 Human-Machine Interaction

Role of the module in the Degree Course

Human-Machine Interaction introduces the basic aspects of communication between human and machine. From a computer science viewpoint, the central topics are developing user-friendly interfaces, the creative aspect of computer operation in general and of Internet operation in particular, and classifying computers into the social context. On the development side, we discuss the conceptual and mathematical foundations and algorithms and tools for developing graphical displays. On the other side, we teach the models and techniques from the area of usability engineering. This approach has now gained considerable influence on the professional practice of computer scientists in the framework of industrial software development. In particular, we examine the necessities and challenges in designing web presences as a highly relevant topic. This topic complements the knowledge and abilities in development and application that the students have gained in the technically oriented fields of computer science. Because the technology exists in the context of society, a further class deals with the role of computer science, consequences that computer scientists need to consider as well as problem areas and points of contact with other disciplines and occupations. This approach acquaints students in computer science with the responsibilities of their occupation and makes them aware of aspects that go beyond the mere technical side.

Content structure of the module

The module consists of the following classes:

- Introduction to Computer Graphics
- Contextual Informatics
- Usability Engineering
- Web Design
- Data Mining

The structure of the class content is as follows:

1. Introduction to Computer Graphics
   - Mathematics of computer graphics and computer games
   - The graphics pipeline
   - OpenGL
   - Transformations in 2d and 3d
   - Modeling of three-dimensional scenes
   - Projections
   - Lighting, reflection, shading
   - Clipping
   - Removal of hidden surfaces
   - Rastering and anti-aliasing of lines
   - Pixel graphics: Textures, color, dot- and vicinity operations

2. Contextual Informatics
    - Features of software as an engineering product
    - Automatic data processing models of cognitive processes
    - Biological information processing
    - Faults in technical and natural systems
    - Product-process complementarity
    - Software development as a learning process
    - Information ethics
    - Responsibility of the computer scientist
3. Usability Engineering (class held in English)
    - Usability engineering: basic definition and examples
    - The human user
    - Modeling rational human behavior
    - Rules for the design of usable user interfaces
    - The usability development process
    - Web site usability
4. Web Presence Design (preliminary structure)
    - Web design problems
    - Content design
    - Design of page structures
    - Navigation
    - Layout, graphics, typography
    - Internationalization
    - Personalization
5. Data Mining
    - Please see the description of this class in module II.1.1.

Usability of the content

During their professional practice, probably all computer scientists will be confronted with the class content of this module, except for the content taught in the class *Introduction to Computer Graphics*. User tests are nowadays carried out widely and have been recognized as absolutely indispensable. User-friendly website design and placing computer science activities within the social and work environment are also topics that are highly relevant for every computer science graduate. This requirement applies both to the role of system developer and to the consulting environment. Computer graphics is a special field with significant relevance in the development area as the spread of powerful computers and screens now requires a mature graphics display in almost any context.

Prerequisites and prior knowledge

The module I.5.2 *Linear Algebra* has to be passed before the class *Introduction to Computer Graphics* can be taken[1]. Apart from basic knowledge in computer science - that is acquired especially in the mandatory module *I.4.1. Basics of Human-Machine-Interaction* - no prior knowledge is required.

Learning goals of the module

The aim of the classes in this module is to provide the students with insights into some of the most important topics and issues in Human-Machine Interaction. In Computer Graphics, the students should get to know the mathematical foundations of graphics generation and the problems of software technology that occur in this process and their algorithmic solution. This is at the same time the basis for developing a graphic engine for the development of games. The class also teaches skills and knowledge that permit the students to use and assess typical graphics systems. In Contextual Informatics, we explain the role of computer science in society – topics include the sociological, psychological, economical, workplace, and legal aspects of this technology. We will make students aware of the effect that their future occupational activities may have in different areas of human life (assessment of the consequences of a technology), which should lead to responsible use of computer science. Usability Engineering teaches the students the fundamental knowledge and methodical approaches for developing user interfaces that are designed to be user-friendly and task-oriented. Developing, planning, and implementing user tests is taught and practically tested in this context. This approach makes students aware of how to include human software users in the development process, how unpredictable usability problems are, even in the case of diligent development work. In Website Design, we examine the usability problems on the Internet in more detail. Because of the Internet's high penetration of society, significant additional problems and questions occur here compared to "classical" interactive software. To complement the technical skills gained elsewhere, students will acquire the necessary knowledge that enables them to develop user-friendly websites adapted to the information requirements of humans.

Teaching of factual knowledge – content competency

We put basic technical concepts and developments into context with human behavior and evaluate them. Relevant conditions in society are taught, especially laws, standards and guidelines.

Teaching of methodological knowledge – methodological skills

Apart from methods to determine and evaluate demands, students get to know different approaches to design interactive systems.

Teaching of transfer skills

Some basic concepts and techniques are generally applicable to other areas of software design, e.g. design of cooperation supporting software or development of tools for knowledge processing and for scientific visualization.

[1] This is only mentioned here in order to make it stand out – classes of the 2nd part of the Degree Course cannot be attended before having successfully completed all modules of the first two semesters in the main subject anyway.

Teaching of normative evaluation skills

Basics of Human-Machine-Interaction are being taught in order to enable the students to solve standard problems on the one hand and, on the other hand, to identify areas that require other specific scientific competences.

Key qualifications

Expected contribution of the class to teach key qualifications:

– The tutorials in small groups foster the ability to cooperate and work in teams.
– Presentation competences are acquired through the according basics of design.
– Students are enabled to evaluate modern IT-technologies.
– Follow-up knowledge for interdisciplinary cooperation is taught.

Module assignment

This is a mandatory module in the field of Human-Machine-Interaction.

Mode

- Credit points: 4 + 4
- SWS (2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week)
- Frequency of the classes offered: A minimum of 2 catalogue classes is offered each semester

Methods of implementation

– The basics are taught in a lecture
– Following the lecture, concepts and techniques are practiced in tutorial classes and small groups.
– The class *Kontextuelle Informatik* has its own didactic approach “Medi@Thing”-consisting of small groups each working on a complex subject and finishing off with a presentation as a guided tour through this virtual knowledge area.

Examination modalities

Oral examination or written examination in each of the two classes.

Person responsible for the module

Szwillus.

II.5 Interdisciplinary

II.5.1 Key Qualifications

Role of the module in the Degree Course

The acquisition of key qualifications is an often – implicit- goal of many classes of the Bachelor Degree Course. This is especially true for the period of practical training during the module *Software Technology*. It is an implicit part of the didactic schedule for tutorial classes in small groups that are offered for almost every class. Here, we focus on communication skills, team work, language skills as well as on aspects of social competency and professional skills.

The module *Key Qualifications* is intended to improve the students' methodological and life skills in order to enable them to **act competently** as a person in complex situations by using adequate methods. Thus, presentation- and media competences are essential.

Content structure of the module

This module consists of two parts: The undergraduate seminar (3 credit points) and the mentoring (1 credit point). The contents of the undergraduate seminar are to be understood exemplarily; they originate from one of the sub-areas and usually follow up the mandatory classes of the 1st part of the Degree Course.

Learning goals

In the undergraduate seminar the students will exemplarily learn to analyze a scientific text, and abstract thinking is trained. The contents have to be presented in written form and orally by acquiring and applying basic knowledge with regard to rhetoric skills and most recent presentation techniques as well as with regard to the ability to be critical and to learn about feedback methods.

During the mentoring the students are assigned to individual teachers and their staff, and mentoring groups of 15 to 20 students are established. During the entire Bachelor Degree Course, there are - as needed - about two meetings in each semester. The aim is to work on study and subject problems by consulting the students individually or in small groups. The intent is to improve engagement, motivation and independence as an aspect of self-competence. The aim of the mentoring is to avoid unnecessary long study periods and to reduce the dropout rate.

The gathered scientific findings are intended to improve the quality of studies and structures offered.

Teaching of factual knowledge – content competency

The content competency with regard to the professional orientation of the undergraduate seminar depends on the individual subject of the class. Regardless of this, the students will learn facts for drawing up and holding a presentation (media competence) as well as for using literature and writing reports. During the mentoring program, the students will receive information on contents and structure of the studies as well as on rules that may influence their studies and the involved institutions.

Teaching of methodological knowledge – methodological skills

The undergraduate seminar focuses on the processing of a subject and its presentation apart from the aspects of content. The students learn to process a subject, to choose, to present and to deal with questions and discussion contributions in practical work as well as to write extensive papers. During the mentoring program, the student receives enough information to plan or refine his/her studies actively and to share problems with co-students in order to receive or give help.

Teaching of transfer skills

The competences that have been acquired during the undergraduate seminar prepare the students for similar situations at a later point of their studies (seminar, project group, final thesis) and in working life (presentations, reports). The activities during the mentoring program prepare the students for situations during their studies and during their future occupational activities including planning activities that are controlled by rules and other regulations.

Teaching of normative evaluation skills

During the undergraduate seminar, the students practice how subjects are processed and presented, which will enable them to assess the suitability of a subject for a presentation, to assess the presentations of other students and to put it into relation with their own work. The interviews and discussions during the mentoring program support the students to assess the progress of other students and to put it into relation with their own study progress.

Module assignment

This is a mandatory interdisciplinary module, essential for the 2nd part of the Degree Course.

Mode

- Credit points: 3 (from undergraduate seminar) and 1 (from mentoring S1)
- SWS: 2 + 1
- Frequency: The undergraduate seminars are offered every semester, depending on the individual areas. The mentoring takes place through the entire Bachelor Degree Course. We organize two 2-hour-meetings in each semester for the mentoring group.

Examination modalities

Written elaboration during the undergraduate seminars and active participation in the mentoring group. The mentoring groups are not being graded. The grade for the module is the grade achieved in the undergraduate seminar.

Person responsible for the module

Szwillus

II.5.2 Bachelor Thesis

In this module we describe aspects of the final thesis of the Bachelor Degree Course independent of the particular subject.

Role in the Computer Science Degree Course
A Bachelor Thesis consists of working on a subject followed by a written report and an oral presentation of the results. The student has to show that he/she is able to work on a computer science subject by using scientific methods and within a given period of time. The subject of a Bachelor Thesis may be, for example, the development of software or hardware, a presentation of proofs or a literature enquiry. The Bachelor Thesis is intended to be a "part-time-work" (15 ECTS-points altogether) in the Degree Course apart from possible other classes in the 6th semester.

Content structure of the module
A Bachelor Thesis is written in four phases: Defining a subject, planning the work (about one month), **writing** the thesis (fixed term of 5 months and formally under surveillance of the examination office) and the presentation of results.

- **Defining a subject:** Every professor and also every scientific staff member who has a doctorate at the Institute of Computer Science and who possesses lecturing experience, is entitled to assign a subject. Subject proposals of the student may be considered.
- After informally having defined a subject, the student will draw up a **work schedule** in coordination with the mentor. About one month part-time work (3 ECTS) is scheduled for phase 1. For this reason, the examination of the thesis should explore the subject thoroughly and plan the following work carefully. The work schedule should present these activities accordingly through profoundly and completely examined aspects and it should include the following elements:

    - Description of the **subject** to be worked on
    - **Motivation** for the work
    - Explicit wording of the **objective**
    - Description of the necessary preparatory work in order to reach the objective including the corresponding **time schedule**
    - A list of the preliminary structure of the thesis

- **Writing of the thesis**: After the work plan has passed, the student has to register formally at the Examination Office and he/she has to inform the supervisor about the beginning of the fixed term. The knowledge level of the Bachelor Thesis matches the contents of the Degree Course up to the 5th semester. The required knowledge will be discussed in advance before the subject is assigned. Furthermore, the supervisor ensures that the Bachelor Thesis can be finished appropriately within the given period of time. He/she advises the student during the entire working period and checks regularly on the progress of the thesis. He/she interferes if any problems occur, e.g. the subject cannot be worked on as formerly intended or the fixed term that is scheduled in the *Regulations for the Conduct of Examination (Prüfungsordnung)*) may be exceeded. The supervisor helps with the written report on time and points out deficiencies.

- **Presentation**: Typically – but not mandatory – the student will present his/her results in an internal public lecture followed by a discussion, at the end of or after having finished the written work. This presentation is also considered for the final grade of the Bachelor Thesis. The supervisor advises the student with regard to presentation tools and points out common mistakes. He should give the student the opportunity to hold a short test presentation since this may be the student's first extensive and graded presentation.

Prerequisites and prior knowledge
With regard to the **content**, the Bachelor Thesis is based on fundamental knowledge and skills as they are obtained in the general Degree Course and it may also be based on some classes the student has taken in the 2nd part of the Degree Course (5. semester).

Learning goals

The student has to prove that he/she can handle a computer science subject on the basis of scientific methods within a given period of time. This includes the ability to apply skills and consult literature, but also the application of previous results and/or relevant development tools. The student has to prove the ability to edit and present a result and to explain how he/she came up with this result. Furthermore, the student has to prove that he/she is competent to process structured and to present his/her solutions/results, the approach, the objective and the necessary basics grammatically correct and on an appropriate abstraction level. The development of a research-relevant contribution of the student himself/herself is desired but not mandatory for the Bachelor Thesis (or on a limited scale only).

Key qualifications
Through the obligatory work with literature, the Bachelor Thesis explicitly promotes the development of knowledge-acquiring-strategies as well as - in most cases - specific foreign language skills. Furthermore, the presentation of working results in computer science in a written and oral form is explicitly required and therefore promoted.

Module assignment
The Bachelor Thesis is a mandatory module. However, its contents may be chosen from a wide spectrum and - as described above –the student may influence parts of it himself/herself.

Mode

- Credit points per module (planning of the work: 3 ECTS, writing and presentation: 12 ECTS)

- Content structure of the module: The period of time designated for writing a Bachelor Thesis by the *Regulations for the Conduct of an Examination" (Prüfungsordnung)* is 9 weeks of full-time work corresponding to about one semester of part-time work.

- Frequency: To be freely defined between students and supervisor

– Duration: Typically, 1 month planning + 5 months writing

Methods of implementation
As described above, an adequate supervision of the student is essential for his/her support. Besides this, the student has to act in an extensive independent way and with adequate initiative regarding the definition of a subject and the work itself. The student will listen to fellow students' presentations of Bachelor and Master Thesis while attending the seminar of the working group which is usually requested by the supervisor. This may also serve as an orientation for the student himself/herself.

Examination modalities

Grading according to § 18 Abs. 2 of the *Regulations for the Conduct of Examination (Prüfungsordnung)*:

The Thesis will be evaluated by the supervisor after submission. He will also consider the presentation and the discussion. Furthermore, a second examiner will evaluate the thesis. Both evaluations will be sent to the Examination Office and will serve as a basis for the grade to be awarded (average of all single evaluations). The student may have access to the evaluations in the examination office.

Person responsible for the module
Szwillus

III. Modules in the Master Degree Course

In every module of the Master Degree Course, one of the additional classes may be substituted by a **seminar** (which is allowed for this module according to the *Module Handbook)*. Furthermore, each possible module assignment will be announced in the corresponding seminar.

III.1 Field: Software Technology and Information Systems

III.1.1 Model-based software development

Role in the Computer Science Degree Course

In a model-based design, models abstractly describing the system to be developed are the central parts. Models are developed in early design phases and on different levels of abstraction, which describe the system in different detail. Model transformations are used to transfer models of one level of abstraction into another. One such transformation is the final code generation. A model-based design process supports the general software engineering principles of abstraction and structuring. It furthermore enables an early analysis of the design and thus improves its quality.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Generating Software from Specifications
- Software Quality Assurance
- Web Engineering
- Model Checking
- Deductive Verification
- Software Safety
- Model-Driven Software Development (MDSD)
- Software Analysis
- Designing code analyses for large software systems
- Advanced Software Engineering: Methods, Architectures and Industrial Application

The classes are structured as follows:

- Generating Software from Specifications (in English):
  See module Languages and Programming Methods
- Software Quality Assurance (in English):
  1. Standards (ISO 9126, CMM-I, SPICE, ISTQB, …)
  2. Constructive Approaches (domain-specific languages, meta-modeling, architectural styles, patterns, MDA, …)

3. Analytical approaches (reviews, inspections, black-box, white-box-testing, Model-based Testing,…)

- Web Engineering (in English):
  1. Web Applications
     - Categories / Characteristics
     - Modeling approaches (WebML, UWE, …)
     - Web technologies (XML, CGI, JSP, JSF, PHP, AJAX, …)
  2. Web Services

     - Standards (WSDL, SOAP, UDDI)
     - Visual contracts
  3. Service-Oriented Architectures (SOA)
     - Concepts, notions
     - Development methods

- Model Checking (in English):

  See Module *Analytical Methods in Software Engineering*

- Deductive Verification (in English):

  See Module *Analytical Methods in Software Engineering*

- Software Safety (in English):

  See Module *Constructive Methods in Software Engineering*

– Model-Driven Software Development
  1. Models and meta-models
  2. Transformation types and languages
  3. Model-driven process models
  4. Distributed modeling
  5. Testing of artifacts in model-driven software development

- Software Analysis (in English)

  1. Intra- and interprocedural data flow analysis
  2. Abstract interpretation
  3. Predicate abstraction
  4. Points-to analysis
  5. Static single assignment
  6. Program slicing

- Designing code analyses for large software systems (in Englisch)

  Static code analysis has the goal of finding programming mistakes automatically, by searching for suspicious anti-patterns in a program’s code. This course will explain how to design static code analysis that are inter-procedural, i.e., consider the whole program, across procedure boundaries. Designing such analyses is challenging, as they need to handle millions of program statements efficiently and precisely. Example applications are drawn from the area of IT security.

The course Software Analysis is a recommended but not required prerequisite. A mature understanding of the Java programming languages and object-oriented programming will be helpful.

Structure of the class:

1. Intra-procedural data-flow analysis
2. Call-graph construction algorithms
3. Context-insensitive inter-procedural data-flow analysis
4. Context-sensitivity using the call-strings approach
5. Value-based contexts
6. Context-sensitivity using the functional approach
7. Efficiently solving distributed problems in the IFDS and IDE frameworks
8. Current challenges in inter-procedural static program analysis

- Advanced Software Engineering: Methods, Architectures and Industrial Application

    1. Advanced Software Engineering: Methods, Architectures, Industrial Application
    2. Architecture (Service-oriented Architecture, Microservices, Web Services, Mobile Architectures, Architectural Frameworks)
    3. Methods (Agile Software Development Methods, Situational Method Engineering, Requirements Engineering)
    4. Project Management (Effort Estimation, Economics of Software Projects)

Usability of the content

This module teaches knowledge and skills in the area of quality assurance in a model-based design as well as the design techniques themselves. In particular, the students learn to choose design as well as quality assurance techniques. The students can use the abilities gained in this module anywhere in their studies and professional work where the design of high-quality software is required.

Prerequisites and prior knowledge

- basic knowledge of software design, as taught in the classes Software Design and Model-based Software Design of the Bachelor Degree Course and /or knowledge of formal modeling techniques as taught in the classes Modeling or Software Design with Formal Methods of the Bachelor Degree Course.

Learning goals

The Students will learn to

Teaching of factual knowledge

- understand quality assurance techniques
- understand software design techniques
- understand design techniques for safety-critical systems

Teaching of methodological knowledge

- select and use languages and methods for developing (large, web-based) software applications

- apply industrial standards
- apply model-based software development techniques

Teaching of transfer skills

- apply the skills and abilities gained in this module in any domain, as well as ones different from those of the module

Teaching of normative evaluation skills

- evaluate and improve existing design processes
- evaluate the applicability of different techniques for a particular application domain

Key qualifications

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge: Combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work and prepare homework

Module assignment

Elective module in the area of software technology and information systems in the Master Degree Course of Computer Science.

Mode

Credit points per module (workload): 8

SWS (hours per week): V2 + Ü1, each (twice)

Frequency: Annually

Duration: 1 – 2 semesters (depending on the class that has been chosen)

Methods of implementation

- Introduce and explain methods and technologies using typical examples
- Apply in practice during the tutorials (all classes)

Organizational arrangements / media use / literature references

- Lectures with overhead presentation
- Tutorial classes in small groups
- Activities expected of the students:Participation in tutorial classes, preparation of homework, preparation and reviewing of lecture material
- Web-based lecture material

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Engels

III.1.2 Languages and Programming Methods

Role in the Computer Science Degree Course

Languages play various important roles in software technology. As programming languages, they are a means of expression for program development and are tailor-made for a specific programming method. As specification languages, computer scientists use them for formulating task descriptions in general, or as tailor-made description methods for specific application areas. Not only the methodically well-founded use, but also designing and implementing such languages by compilers or generators are important topics in software technology.

This module teaches knowledge and skills for the in-depth understanding, specifying, and implementing programming and specification languages. It offers the extension of this topic in a choice of two areas of language implementation or programming methods. This module enables the participants to:

- Acquire special methods for analyzing or synthesizing programs;
- Apply programming methods for object-oriented, parallel, functional, logical or web-based paradigms;
- Design and implement specification languages for application-specific software generators.

This class builds on the knowledge of calculi for describing language features and on methods for implementing languages.

Content structure of the module

For this module, students will select two advanced classes from the following list:

- Generation of Software from Specifications
- Compilation Methods
- Object-Oriented Programming
- Parallel Programming
- Functional Programming
- Prolog with Applications
- Model-Driven Software Development (MDSD)
- Language Based Security
- Compiler Construction
- Prolog and its Application in Interpreter Construction and Computational Linguistics

The classes are structured as follows

- Compilation Methods
    1. Optimization of intermediate code
    2. Code generation
    3. Register allocation
    4. Code parallelization

- Generation of Software from Specifications:
    1. Reuse and generators
    2. Generation of structured texts
    3. Constructing trees
    4. Computations on trees
    5. Names and attributes
    6. Language design
    7. Projects

- Object-Oriented Programming:
    1. Paradigms for the use of inheritance
    2. Separate design with design patterns
    3. Libraries and frameworks
    4. Design errors
    5. Beyond Java

- Parallel Programming in Java:
    1. Monitors and their systematic development
    2. Barriers: Application and implementation
    3. Loop parallelization
    4. Programming with asynchronous messages
    5. Programming with synchronous messages

- Functional Programming:
    1. Recursion paradigms
    2. Function schemes
    3. Type structures
    4. Functions as data
    5. Data streams and lazy evaluation
    6. Fixpoints, functional algebra

- Prolog with Applications:
  See Module Databases and Information Systems

– Model-Driven Software Development:
  See Module *Model-based Software Development*

– Language Based Security
  Structure of the class:
  run-time organization of programs
  code injection attacks and defences
    - buffer overflows and stack canaries
    - control-flow hijacking and control-flow integrity

  code re-use attacks and defences
    - return-oriented programming and software diversity

  data attacks
    - non-control data attacks and data-flow integrity/randomization

  current topics
    - theoretical limits of control-flow integrity

trends in software diversity

Relevant aspects of the lecture will be complemented by lab assignments.

– Compiler Construction
  While the predecessor lecture "Programming Languages and Compilers" (PLaC, Bachelor lecture) focuses on the frontend part of a compiler, this lecture focuses on the compiler backend. More precisely, we will discuss a variety of different optimizations, such as instruction scheduling, instruction selection, and register allocation. To complement the theoretical foundations of this course, we will study the concrete implementation in the LLVM compiler framework.

- Prolog and its Application in Interpreter Construction and Computational Linguistics

  This course views various concepts and techniques from computer science, artificial intelligence, and computational linguistics from a different perspective, i.e. the perspective of programming in logic. Programming in logic in general and the programming language Prolog in particular offer the ability to describe many concepts in logic, i.e. in a declarative way, and to have them tested and executed by an interpreter at a same time. This is in particular useful for puzzles and quizes, but also for self-defined or domain specific languages.

  Structure of the class:

  1. Introduction into logic programming using the Prolog language
  2. Constraint solvers, puzzles, and theorem provers
  3. Interpreters for term substitution systems
  4. Parsing programs, XML, and natural language
  5. Semantics construction, question answering systems, and text translation
  6. Meta interpreters, domain specific languages, and programming in "natural language"
  7. Feature term unification and applications in computer linguistics and ecommerce

Usability of the content
Students can apply the knowledge and abilities gained in this module anywhere in their studies and professional work where a deeper understanding of programming or specification languages is required. In this context, we have oriented the class Generating Software from Specifications more toward developing language-based tools, whereas the other classes teach methods for using languages.

Prerequisites and prior knowledge

Basic knowledge in a language suitable for software development as well as individual practical experience, in program development as it is taught in the Bachelor Degree Course classes *Foundations of Programming 1* and *2* and the *Software Engineering Lab*; understanding of general language features and of non-imperative programming paradigms as taught in the Bachelor Degree Course classes *Foundations of Programming Languages*; knowledge of fundamental methods for the specification and implementation of language features as they are taught in the lecture *Programming Languages and Compilers* of the Bachelor Degree Course.

Learning goals

The students will learn to

Teaching of factual knowledge

- understand advanced techniques for language implementation
- learn language constructs for specific programming paradigms and specification calculi

Teaching of methodological knowledge

- apply generators and standard solutions for language implementation
- systematically apply methods of specific programming paradigms

Teaching of transfer skills

- specify languages for new application tasks and implement them using generators
- transfer programming methods to future languages

Teaching of normative evaluation skills

- recognize the clarity and problem focus of functional program and data formulations
- recognize the value of systematic methods of program development and language implementation

Key qualifications

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work and prepare homework.

Module assignment

Elective module in the area of Software Technology.

Mode

Credit points: 4+4 ECTS (2 catalog classes)

SWS (hours per week): 2+1, 2+1

Frequency: 2-3 catalog classes per year in the winter and summer semesters

Methods of implementation

- We introduce and explain methods and technologies using typical examples which are then applied in practice during the tutorials (all classes)
- In the tutorials, students carry out projects in small supervised groups (GSS)

Organizational arrangements / media use / literature references

- Lectures with overhead presentation
- Tutorial classes in small groups
- Some supervised exercise classes at the computer
- Activities expected of the students: Participation in tutorial classes, preparation of homework, preparation and reviewing of lecture material

- Web-based lecture material

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Kastens

III.1.3 Databases and Information Systems

Role in the Computer Science Degree Course

The module Databases and Information Systems is a module of the Master Degree Course of Computer Science in the field of Software Technology and Information Systems (SWT&IS). The acquired knowledge and skills may be deepened in seminars and are an ideal base for the Master Thesis.

Content structure of the module's classes

The module consists of two of the following classes:

- Databases and Information Systems (in English)
- Prolog with Applications
- Propositional Proof Systems
- Web Engineering
- Advanced Software Engineering: Methods, Architectures and Industrial Application
- Semantic Web
- Language Based Security
- Compiler Construction
- Data Integration

At least one of the following classes has to be taken within the module:

- Databases and Information Systems (in English)
- Prolog with Applications

These classes are continuously updated with the latest content and are - at the time this Module Handbook was written - structured as follows:

- Databases and Information Systems (DBIS, in English)
- 1. Core database technology
- 1.1. Query optimization
- 1.2. Synchronization of transactions
- 1.3. Atomicity and recovery techniques
- 1.4. Distributed and mobile transactions
- 2. Text compression
- 2.1. Encodings
- 2.2. Lempel-Ziv compression techniques
- 2.3. Grammar-based compression
- 2.4. Burrows Wheeler transformation and IRT
- 2.5. Suffix trees and suffix arrays
- 2.6. Wavelet tree and wavelet trie

- Prolog with Applications
  1. Natural language access to information systems (question answering systems etc.)
  2. Applications of computational linguistics (text understanding, automated translation)
  3. Inference methods (constraint-solver, term-rewrite rule systems)

4. Parser and interpreter design (meta-interpreters and language extensions)
5. Search, puzzles, and strategy games.

- Web Engineering
  → see class description in the module of Model-Based Software Development
- Propositional Proof Systems
  → see class description in the module of Knowledge Based Systems
- Advanced Software Engineering: Methods, Architectures and Industrial Application
  → see class description in the module of Model-Based Software Development
- Semantic Web
  "Give me all family-friendly universities in NRW". All information necessary to answer this question can be found on the Web. Still, current search engines and question answering approaches can barely provide an answer to this question. This is due to the content of the Web being difficult to transform into a format that can be easily consumed by machines. The goal of the Semantic Web is to provide an extension of the current Document Web within which machines can easily access, assess and use the semantics of Web content. Therewith, the integration and efficient processing of knowledge on the Web is to be facilitated.

  The goal of this course is to present the fundamentals, technologies and applications of the Semantic Web. Over the last years, the World Wide Web Consortium (W3C) and a large community of scientists and companies (including Google, Yahoo! and the like) have developed standards and technologies for the exchange of knowledge across machines. These standards and technologies are being used within a growing number of applications including search engines, browsers and question answering systems. Some of these applications, such as IBM Watson, are even able to outperform humans at tasks that were considered to essentially favour humans due to their intrinsic complexity. In the lecture, we will present the basic concepts and standards behind semantic technologies. Moreover, the students will be enabled to use Semantic Web technologies for practical applications. The core of the practical part of the course will be the conception and development of semantic applications. To this end, frameworks and existing applications will be provided.

    1. Introduction
    2. The Resource Description Framework
    3. The Web Ontology Language
    4. The SPARQL Query Language
    5. Linked Data
    6. Applications

- Language Based Security
  → see class description in the module of Languages and Programming Methods

- Compiler Construction

→ see class description in the module of Languages and Programming Methods

- Data Integration
  Which trains should I not take given my allergy to pollen? Answering such complex questions is only one of the many domains within which data from different sources (Deutsche Bahn, mCloud, OpenStreetMap, etc.) has to be cleaned, normalized, integrated or even fused. Given the growing amount of data available across the Web and within organizations (companies, universities, government, etc.), there is a growing need for time-efficient and accurate algorithms that allow carrying out the tasks aforementioned.

  The goal of this course is to present the fundamentals of data integration with a focus on Semantic Web technologies. During the course of the lecture, we will define the data integration problem and studies its facets. We will the focus on two of the most important challenges behind data integration: efficiency and accuracy. We will begin by studying efficient approaches for computing simple measures. Thereafter, we will look into the execution of complex similarity measures. Machine-learning approaches for ensuring the accuracy of results with a high precision and a high recall will be presented. Finally, some of the state-of-the-art frameworks for data integration will be presented. After completing this course, the participants will be able to integrate large amounts of data efficiently for problems of their choice. Within the practical part of the course, we will develop efficient and accurate data integration approaches and evaluate them with real data.

  1. Introduction
  2. Formal Definition
  3. Efficient algorithms (HR3, ORCHID, etc.)
  4. Learning specifications (Genetic programming, refinement operators, etc.)
  5. Applications

Usability of the content

Databases play a central role in enterprises because the most important asset that a company has is its data. Therefore, industries significantly depend on methods to efficiently and safely exchange, search, and transform large data volumes, and to be able to adjust them to individual users' needs. This includes the requirement to develop algorithms, and search and storage techniques for flexible data exchange formats (e.g. based on the XML standard family) for a variety of architectures ranging from mobile, to web-oriented, to service-oriented and to cloud-based architectures. In other words, data integrity, data safety, controlled data access, and efficient data processing as provided by database systems are key issues of each company - and a "must" for computer scientists to know about. Furthermore, in modern information systems and in the World Wide Web, documents and semi-structured data are exchanged largely based on modern data exchange formats such as XML. The class *Data Bases and Information Systems* (DBIS2) gives the students a deep understanding of these topics and teaches the practical skills needed to manage projects involving XML technologies and databases.

Furthermore, knowledge and web information management as well as different ways to combine data in order to infer further knowledge or to derive strategies or plans, have become a major challenge for today's companies. These topics are learned in the other class offered in this module such as *Propositional Proof Systems, Web Engineering*, *Scripting Languages* and *Prolog with Applications*.

Students will apply the knowledge and skills acquired in this module in their professional practice in many companies.

The knowledge and skills are also consolidated in seminars that build directly on the module's central class (DBIS2) and form an ideal basis for the master thesis.

Prerequisites and prior knowledge

The contents of the classes *Foundations of Database Systems* and *XML Databases* of the Bachelor Degree Course as well as solid programming skills in Java as taught in the exercises belonging to the classes *Foundations of Programming 1 and 2* of the Bachelor Degree Course are prerequisites.

Learning goals of the module

The student will learn

Teaching of factual knowledge

After completing this module the student will have learned

- the principles of non-standard data models and concepts, architecture and the building blocks of database systems for non-standard data models (e.g. XML database systems, distributed databases)
- the basic concepts and structure of parsers, interpreters and natural language processing systems

Teaching of methodological knowledge

In small groups, during tutorial classes, the student will learn

- to understand and to compare algorithms for XML and text compression techniques,
- to design access control system components for database and information systems.

In practice exercises at the computer the student will learn

- to build their individual databases and information systems.
- to design and change interfaces of web and information systems.
- to use, design and implement small system components (e.g. in web-based systems or information systems) correctly and appropriately.

- work properly with important standards used in industry, i.e. SQL/XML, XPath, XSLT, DOM, SAX, XQuery, XML-Schema, web-services and others.

Teaching of transfer skills

After completing this module, the student will be able to

- transfer the acquired knowledge and skills to other data models or architectures, to other database or web-information systems, and to other middleware or software technologies.

Teaching of normative evaluation skills

After completing this module, the student will be able to

- decide about the suitability of various data models (relational, XML and others) and of various software-design models (rule-based, grammar-based and others) for different applications,
- estimate development times for key database, web, and information system technologies.

Key qualifications

In lab tutorials, students learn to work with important data and software standards used in the industry, for example, SQL/XML, XPath, XSLT, XQuery, XML Schema and SOAP. Through their individual computer exercises with these technologies, the students also acquire the necessary practical experience and are well prepared for learning a large variety of similar database and web-based technologies being used in the industry.

Module assignment

Elective module in the area of Software Technology and Information Systems of the Master Degree Course of Computer Science.

Mode

- Credit points for the complete module (workload): 8
- Credit points for this class: 8
- SWS (hours per week): two classes, each having 2hr lectures + 1hr tutorial per week
- Frequency of the module offered: Annually
- Duration: 1-2 semesters (depending on the class selected from the catalog)

Methods of implementation

- Theoretical concepts are deepened in tutorials in small groups.
- Lecture is combined with exercises using practical examples.
- Essential results are developed within the lecture, such that the student gets a feeling of how new results in this field are being “discovered” or “invented”.
- Practical experience is gained during exercises at the computer, the subject of which is to extended given example programs which have been explained in the lectures.
- Lecture (especially DBIS 2) is oriented with the latest research.

Organizational arrangements / media use / literature references

- Lecture with white board, PowerPoint presentations and the latest scientific publications.
- Lab examples are given as minimized working example programs that could be run on the student's own computer.
- The exercises are organized in different ways according to the individual classes DBIS 2, for example: About 40 % tutorial classes in small groups with exercise sheets and homework; about 60 % exercises applied in practice at the computer.
- Activities expected of the students: Participation in tutorial, preparation of homework and designing their own software at the computer.
- Recent literature will be announced during classes.

Examination modalities

Please see chapter "Grading Policies" at the beginning of this Module Handbook.

Person responsible for the module

Böttcher

III.1.4 Knowledge-Based Systems

Role in the computer science Master Degree Course

The module Knowledge-Based Systems includes classes from *Intelligent Data Processing for Solving Knowledge-Intensive Tasks*. Different knowledge representations and corresponding inference algorithms will be discussed. The classes should enable the students to model problems that are difficult to structure, and to make them accessible for efficient solution methods.

Knowledge-based methods are not a "stand alone technology". Rather, they intend to achieve a new quality in problem solving in combination with classic areas of computer science, or with engineering or business administration applications.

Content structure

The module consists of two additional classes from the following catalog:

- Distributed Problem Solving
- Machine Learning 1
- Machine Learning 2
- Propositional Proof Systems
- Heuristic Search
- Prolog with Applications
- Online Learning and Optimization
- Theorem Proving
- Prolog and its Application in Interpreter Construction and Computational Linguistics
- Semantic Web
- Data Integration

The lectures are structured as follows:

– Distributed Problem Solving

  1. Introduction to agent systems
  2. Representation and processing of knowledge
  3. Planning
  4. Interactions in agent systems
  5. Navigation

– Machine Learning 1

  Due to the ever increasing amount of data that is routinely produced in our information society, the topic of machine learning has become increasingly important in the recent years, not only as a scientific discipline but also as a key technology of modern software and intelligent systems. This lecture provides an introduction to the topic of machine learning, with a specific focus on supervised learning for classification and regression. The lecture covers theoretical foundations of generalisation as well as practical topics and concrete learning algorithms.

  Chapters of the class:

1. Introduction
2. The Learning Problem
3. Generalization
4. The Linear Model
5. Non-Linear Methods
6. Overfitting

– Machine Learning 2

This lecture, which is conceived as a continuation of the Machine Learning I, covers advanced topics in contemporary machine learning research, such as reinforcement learning, online learning and bandit algorithms, multi-task learning, multi-target and structured output prediction, preference learning, learning from weak supervision, and uncertainty in machine learning. The focus of the lecture will be on methods and algorithms, though theoretical issues and applications will be addressed, too.

Chapters of the class:

1. From binary to multi-class classification
2. Ordinal and hierarchical classification
3. Ensemble methods
4. Nonlinear models and kernel machines
5. Multi-target prediction
6. Semi-supervised learning
7. Active learning
8. Online learning
9. Multi-armed bandits
10. Reinforcement learning
11. Preference learning and ranking

– Propositional Proof Systems (in English)

1. Advanced concepts in propositional logic
2. Resolution calculus
3. Calculi and proof complexity
4. Modeling with quantified Boolean formulas
5. Decision problems for quantified Boolean formulas

– Heuristic Search

1. Representation of search spaces

2. Uniformed search procedures
3. Informed search procedures
4. Formal properties of A*
5. Relaxed models
6. View on game-tree-search

– Prolog with Applications

  Please see content description of module III.1.3. III.1.3 *Databases and Information Systems.*

– Theorem Proving

1. Complexity of predicate-logic decision problems
2. Predicate-logic resolution
3. Model elimination and Stickels PTTP
4. Tableau proves with non-classical extensions
5. Implementation of SAT-Solvers
6. Applications

– Online Learning and Optimization

  There are numerous learning scenarios where the training instances are not given in advance, but instead are observed in an online fashion, one after the other. Examples include online advertisement, which consists of deciding which ads to present on a web page, or stock market prediction. An online learning algorithm observes a stream of instances, and makes a prediction for each of them. The learner receives an immediate feedback about its prediction, which is then used to improve its performance on the subsequent predictions.

  In this course, online prediction problems and algorithms that optimize an online performance measure will be introduced with a special focus on algorithm design and theoretical analysis. The purpose of this course is to give an opportunity to glimpse into recent research targeting the emerging field of online learning. We will present different online learning problems including multi-armed bandits, PAC bandits, adversarial bandits, contextual bandits, online convex optimization, follow-the-perturbed-leader and online learning in Markov decision processes.

1. Markov's and Hoeding's inequality
2. Finite horizon / Racing
3. Finite horizon / PAC setting
4. Infinite horizon / Optimism in the face of uncertainty
5. Infinite horizon / ε-greedy, Thompson sampling + regret
6. Adversarial bandits, regret, EXP3, EXP3.P
7. Bandits with side information, Prediction with expert advice
8. Online multi-class classification
9. Online convex optimization / Follow-the-Leader / Follow-the-Regularized-Leader
10. Doubling trick
11. Online convex optimization / Online gradient descend / Perceptron
12. Online convex optimization / Online mirror descend / Winnow

13. Online convex optimization / Adversarial bandits / Gradient Descent Without a Gradient

– Prolog and its Application in Interpreter Construction and Computational Linguistics

  → the class is described in module III.1.2 Languages and Programming Methods, p. 74)

– Semantic Web
  → the class is described in module III.1.3 Databases and Information Systems

– Data Integration
  → the class is described in module III.1.3 Databases and Information Systems

Usability of the content

Students will apply the knowledge and skills acquired in this module in their professional practice where no standard problem solving methods exist, where they must account for aspects of uncertainty and vagueness, and where they need to emulate human problem solving behavior, etc.

Prerequisites and prior knowledge

Students should have an interest in algorithms, abstract modeling, and a sound knowledge and practical experience in a programming language.

Learning goals

Students should have a good command of a selection of problem solving techniques, be able to analyze complex problems independently and estimate the degree of possible automation realistically as well as develop an adequate solution based on this analysis.

In particular, the students should

Teaching of factual knowledge

- know a selection of task settings in the research area of knowledge-based systems,
- learn about modeling techniques,
- understand adequate inference algorithms and understand the differences and (dis)advantages of these algorithms.

Teaching of methodological knowledge

- model systems with different representation formalisms, apply suitable inference techniques and evaluate the results.

Teaching of transfer skills

- recognize the applicability of knowledge-based techniques in new task settings.
- acquire additional concepts and techniques independently.

Teaching of normative evaluation skills

- assess the relevance of knowledge-based techniques in new task settings.
- decide on the usability of problem solving techniques.

Key qualifications

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge: combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work and prepare homework.

Module assignment

Elective module in the area Software Technology and Information Systems

Mode

Credit points: 4+4 ECTS (2 catalog classes)

SWS (hours per week): 2+1, 2+1

Frequency: 2-3 catalog classes per year in the winter and summer semesters

Methods of implementation

- Introduce and explain methods, techniques, and their implementation in the lectures using typical examples (students will then practice them during the tutorials).
- Students will prepare and evaluate prototype implementations during some tutorials.

Organizational arrangements / media use / literature references

- Lectures with overhead presentation
- Tutorial classes in small groups
- Homework assignments, solutions partially presented in tutorials
- Activities expected of the students: Participation in tutorial, preparation of homework, preparation and reviewing of lecture material
- Lecture slides, homework assignments, and links to additional literature on the Web

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Kleine Büning

III.1.5 Analytical Methods in Software Engineering

Role in the Degree Course

This module consolidates the knowledge in the *Foundations of Software Technology* within the area of analytical methods. In contrast to *Constructive Methods of Software Design*, which aim at guaranteeing high quality by construction, analytical methods analyze software designs or code after construction. This module in particular looks at mathematical and formal methods in software design, trying to ensure correctness of the software.

We will discuss in detail concepts and methods of the semantics of programming languages, of semiautomatic and automatic verification techniques as well as classical software quality assurance. A command of these methods facilitates both a better understanding of the concepts of software technology, and scientific investigation, improvement and the foundation of new software technologies.

Depending on the selected focus within this module, the students should, after successfully completing this module, be able to

- evaluate, compare and apply different analytic quality assurance techniques, and choose between different techniques based on the application domain,
- apply semiautomatic and automatic verification techniques.

This module is a mandatory elective module in Software Technology and Information Systems.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Propositional Proof Systems
- Theorem Proving
- Model Checking
- Deductive Verification
- Compilation Methods
- Software Quality Assurance
- Software Analysis
- Designing code analyses for large software systems (DECA)
- Build It, Break It, Fix It

The classes are structured as follows:

- Propositional Proof systems
  See module *Knowledge-Based Systems*
- Theorem Proving
  See module *Knowledge-Based Systems*
- Model Checking
  1. Modeling and property specification for reactive systems
  2. Temporal logics: LTL and CTL
  3. Fairness
  4. LTL Model checking via Büchi automata
  5. BDDs, symbolic model checking

6. Reduction and abstraction techniques, bi-simulation

- Deductive Verification
  1. A programming language and its semantics
  2. Proof systems
  3. Partial/total correctness
  4. Safety and liveness
  5. Soundness and completeness of proof systems
- Compilation Methods
  See module *Languages and Programming Methods*
- Software Quality Assurance
  See module *Model-Based Software Design*
- Software Analysis
  See module *Model-Based Software Design*
- Designing code analyses for large software systems (DECA)
  See module *Model-based Software Development*
- Build It, Break it, Fix It
  This course aims at teaching basic principles of secure software development in a very practical fashion. It is based on the "Break It, Build It, Fix It" security contest by Ruef et al.
  The contest is separated into three phases that test the applicant's skills in the fields of building, breaking and fixing software products.
  In the "Build It" phase, students will be asked to gather in teams and develop small software projects based on a formal specification, also including security requirements. In the "Break It" phase, the developed software will be exchanged between development teams to break the implementation, i.e., find and exploit security vulnerabilities in code of other teams. Afterward, in the "Fix It" phase, teams will get the chance to fix found vulnerabilities and, hence, render their software product more secure.
  The course will contain a theoretical part in which basic strategies of secure software development and vulnerability discovery are presented. Furthermore, specific vulnerability classes and examples of their exploitation will be presented as stimulus at the beginning of the "Break It" phase. Nevertheless, the course is generally of a very practical nature and since securing a software product, as well as breaking it, demands a wide variety of skills and creativity, a quite high amount of motivation and self-organization is required.

Usability of the content

This module teaches knowledge and skills that permit students to understand, formulate, formalize and discuss complex relationships by using formal and mathematical models. Students will use these models specifically when developing safety critical systems and using reliable software.

Beyond these uses, the module offers a starting point for scientific work in formal methods, particulary on verification and model checking.

Prerequisites and prior knowledge

Prerequisite for this module is the ability to model and formalize facts using mathematical and computer science notations as taught in the module *Modeling* as well as in the class *Software Design* of the Bachelor Degree Course. Furthermore, the command of at least one programming language as it is taught in the module *Programming Techniques* of the Bachelor Degree Course is a prerequisite. In addition to this, the students should master the basic techniques for formal definitions and conclusions, as they are taught in the classes *Formal Methods in Software Design*, *Foundations of Knowledge-Based Systems* and in the modules of *Mathematics* of the Bachelor Degree Course.

Learning goals

The students will learn to

Teaching of factual knowledge

After completing this module, the student will

- know the techniques and mathematical structures for formalizing the semantics of programming and modeling languages,
- know and understand different analytical techniques and methods for quality assurance, starting from static analysis over testing up to verification
- know the differences and (dis)advantages of the different techniques

Teaching of methodological knowledge

After completing this module, the student will be able to

- formally model systems and formulate their features
- evaluate the suitability of techniques and methods for different purposes
- apply mathematics and logic correctly and appropriately
- analyze software systems with respect to quality aspects

Teaching of transfer skills

After completing this module, the student will be able to

- set up mathematical models independently and discuss their features
- acquire new concepts and techniques as well as assess and, if necessary, adapt them

Teaching of normative evaluation skills

After completing this module, the student will be able to

- recognize the relevance of the semantic foundation of techniques
- be aware that selecting suitable verification methods requires a detailed analysis of the features of the specific application area.

Key qualifications

For this module, we expect for you to

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge
- combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work, prepare homework and take part in class tutorials.

- evaluate and question new concepts
- discover and establish cross links and relations between similar concepts

Module assignment

Mandatory elective module in the field of Software Technology and Information Systems.

Mode

Credit points: 4+4 ECTS (4 for each class)

SWS (hours per week): 2 lectures + 1 tutorial, 2 lectures + 1 tutorial

Frequency: At least one class per term

Methods of implementation

- Introduce and discuss methods and techniques using typical examples
- Students practice the examples in practice during the tutorials; at times, computer tools are used.

Organizational arrangements / media use / literature references

- Lecture with overhead presentation or writing on the blackboard
- Materials complementing the lecture on the Internet
- Problems are solved collectively in the tutorials

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Wehrheim

III.1.6 Constructive Methods in Software Engineering

Role in the Computer Science Degree Course

This module consolidates the knowledge in software engineering within the area of constructive methods. Constructive methods aim to ensure a high software quality directly through the engineering process (in contrast to analytical methods which ensure quality with an analysis after construction).

After successfully completing this module the students should be able to evaluate, compare and apply different constructive quality assurance concepts. In particular they should be able to choose and apply those methods that are appropriate for a given application domain.

Content structure of the module

For this module, students will select two advanced classes from the following list:

- Web Engineering
- Generating Software from Specifications
- Prolog with Applications
- Compilation Methods
- Parallel Programming
- Objektorientierte Programmierung (Object-Oriented Programming)
- Funktionale Programmierung (Functional Programming)
- Software Safety
- Software Quality Assurance
- Databases and Information Systems 2 (DBIS 2)
- Model-Driven Software Development (MDSD)
- Advanced Software Engineering: Methods, Architectures and Industrial Application
- Language Based Security
- Compiler Construction
- Build It, Break It, Fix It

The classes are structured as follows:

- Web Engineering:
  See module *Model-Based Software Design*
- Generating Software from Specifications (in English):
  See module *Languages and Programming Methods*
- Prolog mit Anwendungen (Prolog with Applications)

  See module *Databases and Information Systems*
- Compilation Methods (in English):

  See module *Languages and Compilation Methods*
- Parallel Programming (in English):
  See module *Languages and Programming Methods*

- Objektorientierte Programmierung (Object-Oriented Programming):
  See module *Languages and Programming Methods*
- Funktionale Programmierung (Functional Programming):
  See module *Languages and Programming Methods*
- Software Safety :
  1. Properties of safety-critical systems
  2. Model-based methods and domain-specific architectures for safety-critical systems
  3. Hazard analysis and fault tolerance
  4. Designing reliable software

- Software Quality Assurance (in English):
  See module *Model-Based Software Design*

- Databases and Information Systems 2 (DBIS 2) (in English):
  See module *Databases and Information Systems*

– Model Driven Software Development:
  See module *Model-based Software Design*

– Advanced Software Engineering: Methods, Architectures and Industrial Application
  See module *Model-Based Software Development*

– Language Based Security
  See module *Languages and Programming Methods*

– Compiler Construction
  See module Languages and Programming Methods

– Build It, Break It, Fix It
  See module *Analytical Methods in Software Engineering*

Usability of the content

The contents of this module can be used in practice for design and implementation of complex software systems. Of particular importance is the knowledge of different design paradigms and the ability to choose a suitable method depending on a given domain and the system to be developed. Special attention will be directed to software-intensive and safety-critical systems.

Prerequisites and prior knowledge

Basic knowledge in software design, as taught in the classes *Software Design and Model-Based Software Design* of the Bachelor Degree Course and basic knowledge about programming languages plus abilities in programming, as can be learned in the classes *Foundations of Programming 1* and *2* and *Foundations of Programming Languages* of the Bachelor Degree Course.

Learning goals

The students will learn to

Teaching of factual knowledge

- know well-established software engineering paradigms.
- understand the applicability of these paradigms to different contexts.

Teaching of methodological knowledge

- choose and apply suitable methods for the design and maintenance of software systems.

Teaching of transfer skills

- design complex software systems with regard to domain-specific requirements.
- adapt and apply new software engineering methods.

Teaching of normative evaluation skills

- estimate the impact of design decisions on software systems.
- assess the applicability of design concepts for a given systems.
- grasp the importance of design decisions in the domain of safety-critical systems.

Key qualifications

- cooperate and work in teams during the tutorial classes
- apply strategies for acquiring knowledge: Combine lectures, prepare and review the lecture material, participate in tutorial classes with supervised group work and prepare homework.

Module assignment

Elective module in the area of Software Technology and Information Systems in the Master Degree Course of Computer Science.

Mode

Credit points per module (Workload): 8

SWS (hours per week): 2 each V2 + Ü1

Frequency: Annually

Duration: 1 to 2 semesters (depending on the catalog class that has been chosen).

Methods of implementation

- Introduce and explain methods and technologies using typical examples.
- Apply the examples in practice during the tutorials in small groups.

Organizational arrangements / media use / literature references

- Lectures with overhead presentation
- Tutorial classes in small groups
- Activities expected of the students: Participation in tutorial, preparation of homework, preparation and reviewing of lecture material
- Lecture slides on the Web

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Schäfer

III.2 Field: Models and Algorithms

III.2.1 Algorithms I

Role of the module in the Degree Course

Algorithms form the foundation of every hardware and software. A circuit converts an algorithm into hardware, a program makes an algorithm "understandable to the computer". Algorithms thus play a central role in computer science. An important goal of algorithm design is (resource) efficiency, for example, developing algorithms that solve a given problem as quickly as possible or with as little memory requirement as possible. In this module we discuss different methodical and application-specific algorithmic problems such as online algorithms, approximation and randomization, and present their applications in algorithms for graphs, coding problems and geometric problems.

Because of the breadth and significance of the field and its important role in the University of Paderborn Computer Science Department, most classes in this module also appear in the module *Algorithms II.*

Content structure of the module

The following classes are offered:

- Advanced distributed algorithms and data structure
- Algorithmic Coding Theory I
- Algorithmic Coding Theory II
- Algorithmic Game Theory
- Algorithmic Geometry
- Algorithmic Number Theory
- Algorithms for complex virtual scenes
- Approximation Algorithms
- Clusteringalgorithms
- Combinatorial optimization
- Computational Geometry
- Graph Algorithms
- Heuristic Search
- Machine Learning 1
- Machine Learning 2
- Online Algorithms
- Randomized Algorithms
- Routing and data management in networks

The module *III.1.4 Knowledge Based Systems* includes and describes the classes *Heuristic Search, Machine Learning 1 and Meachine Learning 2*. The class *Heuristic Search* may also be assigned to the modules described here (III.2.1 and/or III.2.2), if requested.

The class “Clusteringalgorithms” is structured as follows:

1. Introduction
2. k-Means
3. KLD-Clustering
4. k-Means++
5. Constant Factor k-Means

6. Agglomerative Clustering
7. dbscan
8. Johnson-Lindenstrauss
9. SVD
10. Mixture Models and the EM Algorithm

The class “Computational Geometry” deals with algorithms and data structures in the area of computational geometry. The basic elements and input are geometric data (points, lines, circles, polygons, volumes). The problems are formulated geometrically and the task is to find an algorithmic solution using special geometric data structures. The algorithms are theoretically analyzed. For this purpose, runtime and space is analyzed and correctness of the algorithms is proved.
The class is structured as follows:

- Sweep line method: closest pair, line segment intersection
- Geometric data structures: k-d-tree, range tree, priority search tree, fractional cascading
- Voronoi Diagrams
- Delaunay Triangulation
- Motion planning for robots
- Dynamization of static data structures

The students get to know fundamental techniques in the area of computational geometry. They can decide for which geometric problem which algorithm is most appropriate. They can adapt algorithms to a new situation.

Usability of the content

The ability to design not just any algorithms, but to design resource-conserving (that is, efficient) algorithms for specific problems and the ability to assess problems with regard to their inherent complexity is important for many subfields of computer science. Databases and information systems, computer graphics systems and scientific computation are important examples.

Prerequisites and prior knowledge

Prerequisites are the basic concepts from algorithm and complexity theory as they are taught in *Data Structures and Algorithms*, *Introduction to Computability*, *Complexity Theory* and *Formal Languages* and *Fundamental Algorithms* in the Bachelor Degree Course.

Apart from basic mathematical knowledge as it is taught during undergraduate study in the Bachelor Degree Course, students must have an interest in creative problem solving with mathematically exact methods.

Learning goals of the module

Teaching of factual knowledge – Content Competence

Students should know and understand selected algorithms and data structures along with their correctness and runtime analyses as well as essential concepts and methods of the development and analysis of algorithms. They should deepen this understanding in at least one algorithmic sub discipline.

Teaching of methodological skills – methodological competency

Students should be able to apply on their own adequate algorithmic techniques and suitable data structures in order to solve algorithmic problems and to carry out correctness and runtime analyses.

Teaching of transfer skills

Students should have insights into the application areas of different algorithmic sub disciplines and, vice versa, they should be able to assign application problems to the according sub disciplines.

Teaching of normative evaluation skills

Students should be able to assess algorithmic problems according to their complexity in order to detect the possibilities and limits of achievable solution quality and in order to be able to evaluate the quality of the solutions found.

Module assignment

Elective module in the field of Models and Algorithms.

Mode

- Credit points: 4+4
- SWS (hours per week): 2+1,2+1
- Frequency: Annually.

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Meyer auf der Heide

III.2.2 Algorithms II

Role of the module in the Degree Course

Algorithms form the foundation of every hardware and software. A circuit converts an algorithm into hardware, a program makes an algorithm "understandable to the computer". Algorithms thus play a central role in computer science. An important goal of algorithm design is (resource) efficiency, for example, developing algorithms that solve a given problem as quickly as possible or with as little memory requirement as possible. In this module we discuss different methodical and application-specific algorithmic problems such as online algorithms, approximation and randomization, and present their applications in algorithms for graphs, coding problems and geometric problems. Other classes in this module explore the inherent complexity of problems, i.e. the derivation of lower bounds and the complexity theoretic comparison of problems. Classes in the area of cryptography complement these courses. Here, the inherent complexity of problems is used in the design of secure cryptographic primitives, e.g. encryption schemes.

Content structure of the module

The following classes are offered:

- Advanced complexity theory
- Advanced distributed algorithms and data structure
- Algorithmic Game Theory
- Algorithmic Geometry
- Algorithms for complex virtual scenes
- Approximation Algorithms
- Clusteringalgorithms
- Combinatorial optimization
- Computational Geometry
- Computational models
- Concrete Complexity Theory
- Cryptographic protocols
- Graph Algorithms
- Online Algorithms
- Provable Security
- Randomized Algorithms
- Routing and data management in networks

The classes “Clusteringalgorithms” and “Computational Geometry” are described in module III.2.1 Algorithms 1.

Complexity Theory in general deals with determining the amount of resources (e.g. runtime, memory consumption) necessary and sufficient for solving a given algorithmic problem (e.g. Travelling Salesperson Problem (TSP)) on a given machine model (e.g. Turing machine). One approach is to define complexity classes like P, NP, PSPACE, in order to classify problem complexity by means of completeness in such classes, like the famous class of NP-complete problems. This gives conditional results like ”If NP is not equal P, then TSP is not solvable in polynomial time.” This branch of Complexity Theory is often referred to as Structural Complexity Theory.
In contrast, proving explicit lower bounds for given problems is the topic of the so-called Concrete Complexity Theory. As nobody is currently able to prove superlinear time bounds

for explicitly defined problems on general computation models like Turing machines, one considers somewhat restricted models like 1-tape Turing machines, monotone Boolean circuits, Boolean circuits with bounded depth, algebraic computation models, and several kinds of parallel computation models.
This lecture surveys approaches to prove such lower bound on various such models.
Topics:

1. Boolean Circuits: basics, some lower bounds
2. Algebraic computations: lower bounds for different sets of arithmetic operations
3. Lower bounds for parallel computations

Usability of the content

The ability to design not just any algorithms, but to design resource-conserving (that is, efficient) algorithms for specific problems and the ability to assess problems with regard to their inherent complexity is important for many subfields of computer science. Databases and information systems, computer graphics systems and scientific computation are important examples.

Prerequisites and prior knowledge

Prerequisites are the basic concepts from algorithm and complexity theory as they are taught in *Data Structures and Algorithms*, *Introduction to Computability*, *Complexity Theory* and *Formal Languages* and *Fundamental Algorithms* in the Bachelor Degree Course.

Apart from basic mathematical knowledge as it is taught during undergraduate study in the Bachelor Degree Course, students must have an interest in creative problem solving with mathematically exact methods.

Learning goals of the module

Teaching of factual knowledge – Content Competence

Students should know and understand selected algorithms and data structures along with their correctness and runtime analyses as well as essential concepts and methods of the development and analysis of algorithms. They should deepen this understanding in at least one algorithmic sub discipline. Alternatively, students deepen their understanding of important concepts of complexity theory or cryptography and their understanding of the interplay between complexity and cryptographic security.

Teaching of methodological skills – methodological competency

Students should be able to apply on their own adequate algorithmic techniques and suitable data structures in order to solve algorithmic problems and to carry out correctness and runtime analyses. In the classes belonging to complexity theory and cryptography students be able to examine and classify the inherent complexity of problems. They should also be able to exploit theses insights in the development of secure cryptographic primitives.

Teaching of transfer skills

Students should have insights into the application areas of different algorithmic sub disciplines and, vice versa, they should be able to assign application problems to the according sub disciplines. They should be able to apply and understand the interplay of complexity theoretic and cryptographic concepts.

Teaching of normative evaluation skills

Students should be able to assess algorithmic problems according to their complexity in order to detect the possibilities and limits of achievable solution quality and in order to be able to evaluate the quality of the solutions found. Students should be able to relate the security of cryptographic primitives to the inherent complexity of algorithmic problems. They should be able to assess the security of cryptographic primitives.

Module assignment

Elective module in the field of Models and Algorithms.

Mode

- Credit points: 4+4
- SWS (hours per week): 2+1,2+1
- Frequency: Annually.

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Meyer auf der Heide

III.2.3 Complexity and Cryptography

Role of the module in the Degree Course

At the core, this module deals with the question about the limitations of computability and the classification of problems with regard to their algorithmic complexity. We will use running time and memory requirements in particular as measures of complexity, but also for example, parallelizability. The module includes the proof of both un-decidability, for example, of arithmetic's, and the investigation of the problem-inherent complexity, that is, the proof of lower complexity bounds and the complexity comparison of problems. We also examine formal languages as aspects of complexity theory. The basics of algorithms and complexity are complemented by methods for the algorithmic treatment of very complex problems, for example, approximation algorithms.

In the field of cryptography, the module teaches fundamental tasks, methods and security concepts of cryptography. Furthermore, we examine the most important, partly standardized procedures of cryptography including their number-theoretical and complexity-theoretical basis. Another main emphasis of this field is the construction of secure cryptographic primitives from general complexity theory assumptions.

Content structure of the module

The following classes are offered:

- Complexity Theory II
- Concrete Complexity Theory
- Models of Computation
- Approximation Algorithms
- Cryptographic Protocols
- Provable Security
- Lattices in Computer Science
- Logic and Deduction

The class “Concrete Complexity Theory” is described in module III.2.2 Algorithms 2.

Usability of the content

This module enables the students to assess the fundamental limitations of algorithmic solvability of problems and those imposed by resource limits. It also teaches them how to apply this skill to actual problems. These skills are of advantage in all areas in models and algorithms as well as wherever algorithms for complex problems are developed. Furthermore, this model enables the students to assess the security of cryptographic methods. This ability is essential in the field of security when suitable cryptographic methods for the construction of secure systems have to be selected.

Prerequisites and prior knowledge

Prerequisites are the basic concepts from complexity theory, as they are taught in the lectures Introduction to Computability, Complexity, Formal Languages and Complexity Theory in the Bachelor Degree Course.

Apart from basic mathematical knowledge as it is taught during undergraduate study in the Bachelor Degree Course, an interest in creative problem solving with mathematically exact methods is required.

Learning goals of the module

Teaching of factual knowledge

- Goals, concepts and methods of complexity theory and cryptography.
- Fundamental technology to analyze the complexity of problems.
- Essential security concepts and technology of the security analysis in cryptographic procedures.
- Connection between complexity theory and cryptography.

Teaching of methodological skills

- Design of advanced complexity analysis
- Methods for security analysis of complex cryptographic procedures

Teaching of transfer skills

- Ability to independently develop advanced methods and concepts of complexity theory and to apply them to new problems

Teaching of normative evaluation skills

- Assessment of problems with regard to their algorithmic complexity
- Assessment of the security of cryptographic primitives

Module assignment

Elective module in Models and Algorithms.

Mode

- Credit points: 4+4 (per class)
- SWS (hours per week): 2+1,2+1
- Frequency of the class offered: Annually.

Methodological Implementation

- Methods and techniques are introduced and discussed by examples.
- The methods and techniques are applied to new, typical examples in exercises in small groups.

Organizational arrangements / media use / literature references

- Lecture with overhead slides
- Tutorial classes in small groups
- Expected contribution from the students: Collaboration in tutorial classes, homework, preparatory and touch up work of the lectures
- Literature will be announced at the beginning or the module

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module
Blömer

III.2.4 Algorithms in Computer Networks

Role of the module in the Degree Course

In recent years, the theory of parallel algorithms and architectures has enabled massively parallel computers with a thousand processors or more to be built and to be used efficiently. The big challenges for computer science consist of weather forecasting, ocean simulation, astro-physical simulations and drug design but difficult optimization problems also require using massively parallel supercomputers. In addition to supercomputers, using parallel computers in the form of multi-processor PCs or processor clusters is already standard in many scientific, commercial or industrial applications.

The Internet, which overall is also a parallel computer, is already used as such, for example, when grid computing applications are implemented.

Theoretical computer science has contributed significantly to the efficient use of parallel computers in many areas requiring high computing performance. This includes the model development for parallel computers and the development of efficient algorithms for these models. The basic aim of the module's classes is to achieve a general understanding for parallel processes by discussing analyzable parallel algorithms and architectures.

Content structure of the module

The module includes both classes that present efficient algorithms for problem solving with computer networks, and classes that present problem solutions that permit an efficient use of computer networks.

The module consists of the following classes:

- Algorithms for Synchronous Computer Networks
- Algorithmic Foundations of the Internet
- Algorithmic Problems in Wireless Networks
- Concrete Complexity Theory
- Resource Management in Computer Networks
- Routing and Data Management in Networks
- Advanced Distributed Algorithms and Data Structures

The class “Concrete Complexity Theory” is described in module III.2.2 Algorithms 2.

Usability of the content

The basic knowledge about parallel algorithms and architectures students acquire in this module is essential for anyone working with parallel computers in research, commerce or industry. The knowledge acquired has a promising future because of the continued heavy growth of application fields for parallel supercomputers, in particular in scientific computation, but also because of the increasing number of multi-processor PCs or PC clusters in a commercial or industrial setting.

In addition to these types of parallel computers, the Internet is also increasingly used as a parallel computer, because of the rising number of services offered. The knowledge gained in this module will also be of importance for this growing industry.

Prerequisites and prior knowledge

Students must know the basic concepts in the fields of algorithms, data structures, computability and complexity theory, as they are taught in the first four semesters of the

Bachelor Degree Course. Knowledge of algorithms and their analysis, as they are taught in the lecture *Efficient Algorithms* in the Bachelor Degree Course are beneficial.

Learning goals of the module

The class aims to introduce students to important parallel algorithmic techniques and architectures. One aim is to provide a base repertoire of parallel algorithms for problems that occur ever so often in applications. The other aim is to enable students to develop efficient parallel algorithms for new problem scenarios, and to implement them on actual parallel computers-- regardless of their type-- from supercomputers to the Internet.

Module assignment

Elective module in Models and Algorithms.

Mode

- Credit points: 4+4
- SWS (hours per week): 2+1,2+1
- Frequency of the module offered: Annually.

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Scheideler

III.3 Field: Embedded Systems and System Software

III.3.1 Distributed Computer Systems

Role of the module in the Degree Course

(The following text is currently being revised)

In a global economy based on a division of labor, distributed computer systems form part of the essential infrastructure in computing. Their secure and fast functioning is critical for business success. Computer networks and operating and distributed systems form the basic concepts of modern information systems. Distributed computer systems are based on the basic concepts of computer networks, operating systems, and distributed systems. The operating systems link the computer hardware with the software and provide an interface to the hardware resources. Computer networks transport data between separate, physically different devices. To this end, various communication channels are used (wired, fiber optics or wireless), devices of vastly different performance characteristics are connected, and various degrees of quality assurance are given (correct, dependable or efficient communication). Based on existing computer networks, distributed systems permit interactions beyond computer limits. This interaction allows, for example, connecting different and spatially separated departments in an enterprise or for implementing general web services. Systems for distributed processing are also used where processes are to be accelerated or where fault tolerance is to be provided. In all cases, however, it is necessary that the implementation is carried out, for the user, as transparently, reliably and securely as possible. Security aspects play a particularly important role as the processing is carried out via insecure network structures.

In this module, students will first develop basic, general principles that they need to realize such distributed computing systems. They then transfer the general principles to actual system software, computer resources and programming models, and demonstrate them by case studies.

We have designed this module for students who wish to specialize in the software-oriented part of Embedded Systems and System Software, but do not intend to enroll in further ESS modules. This module builds on the foundations presented in *Concepts and Methods of System Software* and *Embedded Systems and System Software*. Students must take either Introduction to Distributed Systems or Computer Networks from the second stage of the Bachelor Degree Course as a prerequisite.

Content structure of the module

To successfully complete this module, students will select two classes from the following list. However, students have to choose between "*Empirical performance evaluation*" and "*Network Simulation"* since they cannot take both classes.

- Architecture of Parallel Computer Systems
- Computersicherheit / Computer Security (in German)
- Empirical Performance Evaluation
- Future Internet
- High-Performance Computing
- Mobile Communication

- Network Simulation
- Operating Systems
- Vehicular Networking

The classes are structured as follows:

- Architecture of Parallel Computer Systems

  1. Parallel computing from the user’s point of view
  2. Programming of parallel computers
  3. Basics of computer architecure
  4. Architecture of parallel computer systems
  5. Memory connected systems
  6. Cache-coherence in scaled computer systems
  7. Cluster-Computing
  8. Energy efficiency

- Computersicherheit / Computer Security (in German)

  1. Secure user authentication
  2. Basic network attacks
  3. Cryptographic building blocks
  4. Application layer security: e-mail encryption and more
  5. Transport layer security
  6. Network layer security, TOR and Onion Routing
  7. Link layer security
  8. WiFi security
  9. Crypto-attacks in practice

- Empirical Performance Evaluation

  10. Introduction
  11. A Simple Queue
  12. Simulating a Queue
  13. A Complicated Queueing System
  14. Random Distributions in Simulations
  15. How to get data out of simulation runs
  16. Interpreting simulation results, comparing systems
  17. Factorial design

- Future Internet

  1. Switch architecture
  2. Congestion control, buffer management in IP networks
  3. Flow-based networking (MPLS, OpenFlow, Software-Defined Networking)
  4. Data-center networking
  5. Selected topics, e.g., Information-Centric Networking

- High-Performance Computing

  1. Introduction to High-Performance Computing
  2. Models and programming patterns for parallel computing
  3. Programming languages and libraries for HPC
  4. Performance analysis, optimization, and debugging
  5. Heterogeneous computing with hardware accelerators

6. Case studies

- Mobile Communication
  1. Wireless communication and wireless channel models
  2. Medium Access in wireless systems
  3. Cellular Systems
  4. Wireless local networks (esp. IEEE 802.11)
  5. Techniques to assess performance of such systems and their protocols
- Network Simulation
  1. Network Simulation
  2. OMNeT++
  3. Model Management with git
  4. Verification and Validation
  5. Design of Experiments
  6. Result Evaluation with R
- Operating Systems
  1. Parallelism
  2. Scheduling
  3. Synchronization
  4. Inter-Process Communication
  5. Memory Management
  6. Security
  7. Embedded OS
  8. Real-Time
- Vehicular Networking
  1. Overview, Use Cases, and Architectures
  2. Protocols: K-Line, CAN, and LIN
  3. Protocols: FlexRay, MOST, Ethernet
  4. Electronic Control Units
  5. Overview, Use Cases, and Architectures
  6. Technology
  7. Traffic Information Systems
  8. Routing, Flooding, Geocast
  9. Beaconing
  10. Privacy
  11. Simulation

Usability of the content

(The following text is currently being revised)

Students will apply what they learn in these classes to application development, system administration, and in designing and implementing special systems. They will apply the mechanisms presented for resource management, security, and cross-platform communication both in classic information systems and - in an adapted versions - in special hardware resources. Knowledge about the detailed functioning of computer networks also helps computer scientists satisfy the complex requirements of modern information systems and access new fields of application. Time-dependent processes often play an important role in commercial and technical systems. Finally, students gain useful basic knowledge for network administration.

The basic building blocks for developing distributed systems are required for Internet applications, web services, enterprise software, etc. The knowledge gained should enable students to assess, select and adapt various solution paths and components to a specific task. Last but not least, the knowledge about high-performance computing is also required in many related sciences in which complex and computing-intensive tasks are to be solved.

Prerequisites and prior knowledge

Distributed Systems and/or *Computer Networks* taught in the second stage of the Bachelor Degree Course in computer science are prerequisites. Students should also know the content of *Concepts and Methods of Systems Software* in the Bachelor Degree Course. Classes such as performance-oriented programming expect students to be willing to become familiar with system-level programming languages. Students must have a basic knowledge of the programming languages from the module *Programming Techniques* in the Bachelor Degree Course.

Learning goals of the module

The students should gain an understanding of the specific features of system software and computer networks and become familiar with the basic building blocks for developing operating and distributed systems. Students will recognize potential threats to the computer operation arising from unauthorized access to resources, and should be able to take the corresponding counter measures. They will learn to assess and evaluate possibilities, limitations and risks of open distributed systems and high-performance computers. Finally, students will comprehend the core methods for efficient processing and resource management and should apply them to concrete examples.

Teaching of factual knowledge – content competency

After completing this module, students will learn the

- Relationship between hardware and system software
- Structure, management and synchronization of processes
- Techniques for memory management and scheduling
- Techniques for securing critical areas
- Techniques for designing parallel and concurrent programs
- Techniques for efficient, problem- and requirements-adequate transmission of data in wired, wireless or mobile communication systems

Teaching of methodological knowledge – methodological skills

After completing this module, students will know the

- Methods for efficiently managing and allocating resources
- Methods for detecting and avoiding deadlocks
- Methods for cooperation between processes in distributed systems
- Process interaction methods
- Methods and approaches for performance analysis and optimization in communication systems and similar technical systems

Teaching of transfer skills

After completing this module, students will know how to transfer global strategies to specified individual situations, for example, as part of exercises.

Teaching of normative evaluation skills

After completing this module, students will

- Develop techniques for applying different strategies
- Recognize the practical value of the concepts and methods of system software
- Select a solution strategy adequate to a given task, its optimization goals and its constraints

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Activities expected of students: cooperation during class-based tutorials, homework and independent study of secondary literature.

Module assignment

Elective module in Embedded Systems and System Software.

Mode

- Credit points for each module (workload) : 8
- Credit points of the class: 4 each
- SWS (hours per week): (2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week)
- Frequency of the module offered: 2-4 classes per year during the winter and summer semesters
- Duration: 2 semesters

Methods of implementation

In addition to classical lectures and exercises, tutorials in small groups foster the practical application of the methods presented to selected examples. Students must adapt particular, parameters and strategies to a given situation, problem or case study. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling. This approach is reinforced by offering “project groups,” where real research problems are to be solved by students in realistically sized groups working over a realistic period of time. Seminars offer the option to delve deeply into a limited topic area.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups. These tutorials include demonstrating the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module
Karl

III.3.2 System Software

Role of the module in the Degree Course

(The following text is currently being revised)

Operating systems form the fundamental software layer that connects the computer hardware to the software. Together with other components of the system software, it enables developing applications and provides an interface to the hardware resources. Distributed systems, however, permit interactions beyond computer limits. This characteristic allows for connecting different and spatially separated departments in an enterprise and for implementing general web services. Systems for distributed processing are also used where processes are to be accelerated or where failure safety is to be achieved. In all cases, however, it is necessary that user implementation is carried out as transparently, reliably and securely as possible. Security aspects play a particularly important role as the processing is carried out via insecure network structures. Current developments lead to a convergence of operating systems and distributed systems such that many links become visible.

We have designed this module for students wishing to specialize in the SW-oriented part of *Embedded Systems and System Software* (ESS). The specific focus on operating and distributed systems permits combining the module with all other aspects of ESS, such as computer networks or embedded and real-time systems as part of the specialization area. This module builds on the foundations presented in *Concepts and Methods of System Software* and Computer Engineering and requires Distributed Systems 1 from the second stage of the Bachelor Degree Course as a prerequisite. The general principles are now transferred to actual system software, computer resources and programming models, and are demonstrated through case studies.

Content structure of the class

To complete this module successfully, students select two classes from the following list:

– Architecture of parallel computer systems

– Computersicherheit / Computersecurity (in German)

– High-Performance Computing

– Operating Systems

– Swarm Robotics

The classes are structured as follows:

– Architecture of parallel computer systems
  Please see description of the class in Module III.3.1 Distributed Computer Systems

– Computersicherheit / Computer Security (in German)
  Please see description of the class in Module III.3.1 Distributed Computer Systems

– High-Performance Computing
  Please see description of the class in Module III.3.1 Distributed Computer Systems

– Operating Systems
  Please see description of the class in Module III.3.1 Distributed Computer Systems

– Swarm Robotics

  1. Behavior-based robotics

2. Scenarios of swarm robotics
3. Modeling swarm systems
4. Local sampling
5. Collective decision-making

Usability of the content

(The following text is currently being revised)

Students will apply what they learn in these classes to application development, system administration, and in designing and implementing special systems. Students will apply the mechanisms presented for resource management, security and cross-platform communication both in classic information systems and -- in an adapted version -- in special hardware resources. Students must also know about high-performance computing for many related sciences in which they must solve complex and computing-intensive tasks. The basic building blocks for developing distributed systems are required for Internet applications, web services, enterprise software, etc. The knowledge gained should enable students to assess, select and adapt various solution paths and components to a specific task.

Prerequisites and prior knowledge

The content of the class *Distributed Systems*, taught in the second stage of the Bachelor Degree Course, is a prerequisite. Further prerequisites are the contents of *Foundations of Computer Engineering* and *Concepts and Methods of System Software* in the Bachelor Degree Course. Classes such as performance-oriented programming expect students to be willing to become familiar with system-level programming languages. Students must have a basic knowledge of the programming languages from the module *Programming Techniques* in the Bachelor Degree Course.

Learning goals of the module

Students will understand the specific features of system software and become familiar with the basic building blocks for developing operating and distributed systems. Students will recognize potential threats to the computer operation arising from unauthorized access to resources, and will be able to take the corresponding counter measures. They will learn to assess and evaluate possibilities, limitations and risks of open distributed systems and high-performance computers. Finally, students will comprehend the core methods for efficient processing and resource management and will apply them to concrete examples.

Teaching of factual knowledge – content competency

After completing this module, students will know the

- Relationship between hardware and system software
- Structure, management and synchronization of processes
- Techniques for memory management and scheduling
- Techniques for securing critical areas
- Techniques for designing parallel and concurrent programs

Teaching of methodological knowledge – methodological skills

After completing this module, students will have learned the

- Methods for efficiently managing and allocating resources
- Methods for detecting and avoiding deadlocks

- Methods for cooperating between processes in distributed systems
- Process interaction methods

Teaching of transfer skills

After completing this module, students will have learned how to transfer global strategies to specified individual situations, for example as part of exercises

Teaching of normative evaluation skills

After completing this module, students will be able to

- Develop techniques to apply different strategies
- Recognize the practical value of the concepts and methods of system software

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Activities expected of students: Cooperation during tutorial classes, homework and independent study of secondary literature.

Module assignment

Elective module in *Embedded Systems and System Software*

Mode

- Credit points for each module (workload) : 8
- Credit points of the class: 4 each
- SWS (hours per week): 2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week
- Frequency of the class offered: 2-4 classes per year during the winter and summer semesters
- Duration: 2 semesters

Methods of implementation

Tutorials in small groups foster the practical application of the methods to selected examples. In particular, students must adapt parameters and strategies to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups including students demonstrating the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Dressler

III.3.3 Computer Networks

Role of the module in the Degree Course

(The following text is currently being revised)

Transmitting data between separate devices using various communication media is a basic building block for almost all modern information systems. Such communication enables distributed systems; it makes mobile communication possible by using wireless transmission; it leads to varying requirements in different system architectures and is used in different forms – from highly reliable communication in small automation networks ranging via the Internet at large to small, self-organized, wirelessly communicating ad hoc networks. These topics are treated in Computer Networks.

We have designed Computer Networks for students wanting to specialize in *Embedded Systems and System Software* (ESS) and wanting to combine Computer Networks with one of the three other ESS core subjects: Operating and Distributed Systems, Embedded and Real-Time Systems, or HW/SW Codesign. This module builds on ESS, taught in the second stage of the Bachelor Degree Course and has Computer Networks as a prerequisite.

Content structure of the module

To complete this module successfully, students select two classes from the following list. However, students have to choose between "*Empirical performance evaluation*" and "*Network Simulation"* since they cannot take both classes.

- Computersicherheit / Computer Security (in German)
- Empirical Performance Evaluation
- Future Internet
- Mobile Communication
- Network Simulation
- Vehicular Networking

The classes are structured as follows:

- Computersicherheit / Computer Security (in German)
  Please see description of the class in Module III.3.1 Distributed Computer Systems
- Empirical Performance Evaluation
  Please see description of the class in Module III.3.1 Distributed Computer Systems
- Future Internet
  Please see description of the class in Module III.3.1 Distributed Computer Systems
- Mobile Communication
  Please see description of the class in Module III.3.1 Distributed Computer Systems
- Network Simulation
  Please see description of the class in Module III.3.1 Distributed Computer Systems
- Vehicular Networking
  Please see description of the class in Module III.3.1 Distributed Computer Systems

Usability of the content

(The following text is currently being revised)

Knowledge about the detailed functioning of computer networks helps computer scientists to satisfy the complex requirements of modern information systems and to access new fields of application. In addition to a well-founded theoretical discussion of communication system, we also teach practical competency in usage, planning, configuration, programming, and administration of networks, which are relevant skills to many computer scientists' professions. The detailed modeling of relevant aspects and processes in a computer network is also a basic foundation for a simulation-based performance assessment – in particular when assessing systems or protocols that do not yet exist. Students will use the formal specification of communication systems for the (semi-)automated implementation of protocols with the help of respective programming tools and for testing the systems. The implementation results in a performance assessment in the form of laboratory measurements. Finally, the knowledge gained serves as a first step toward system and network administration.

Prerequisites and prior knowledge

We expect students to have completed *Introduction to Computability, Complexity, and Formal Languages* taught in the second stage of the Bachelor Degree Course. A further prerequisite is the content of *Concepts and Methods of System Software* in the 1st part of the Bachelor Degree Course. Furthermore, students must have basic knowledge of the programming languages from the module *Programming Techniques* in the Bachelor Degree Course.

Learning goals of the module

Based on known foundations in computer science, students should get to know and comprehend basic concepts and different functionalities of computer networks and their use. Students specializing in this area should familiarize themselves with the core concepts and protocols of communication systems and understand the reasons why certain design decisions in these systems were made the way they were. Specialists must know the methods for modeling/formal specification of communication systems and for performance assessment via simulation/measurement. They must also be able to adapt these methods to a specific problem scenario.

Teaching of factual knowledge – content competency

After completing this module, students will have learned the

- Techniques for efficient, problem- and requirements-adequate transmission of data over various transmission media, especially in mobile and wireless communication systems
- Advanced and specialized methods and techniques of the Internet

Teaching of methodological knowledge – methodological skills

After completing this module, students will know the

- Methods of performance evaluation and optimization in communication systems and similar technical systems
- Specification of communication systems and their protocols
- Approaches for systematic protocol implementation

Teaching of transfer skills

After completing this module, students will be able to transfer global strategies to specified individual situations, for example as part of exercises.

Teaching of normative evaluation skills

After completing this module, students will know how to

- Develop techniques to apply different strategies
- Select a solution strategy adequate to a given task, its optimization goals, and its constraints

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams. Students will gain practical experience via lab classes and project groups as well as detailed knowledge in seminars.

Activities expected of students: Cooperation during class-based tutorials, homework and independent study of secondary literature.

Module assignment

Elective module in the field of Embedded Systems and System Software

Mode

- Credit points for each module (workload) : 8
- Credit points of the class: 4 each
- SWS (hours per week): 2 lectures + 1 tutorial per week, project groups, seminars
- Frequency of the module offered: 2-4 classes per year during the winter and summer semesters
- Duration: 2 semesters

Methods of implementation

Tutorials in small groups foster the practical application of the methods presented in selected examples. In particular, parameters and strategies must be adapted to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups including students demonstrating the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module
Karl

III.3.4 Embedded Systems

Role of the module in the Degree Course

(The following text is currently being revised)

Embedded systems play a central role because of the continuous computerization of all technical systems. Large parts of not only mechanical, automotive and aerospace engineering, but also of communication technology can no longer be implemented without embedded systems. Embedded systems refer to the information processing components in such systems. They usually consist of dedicated hardware and the software that builds on it. Both are designed using the fundamental methods of computer science, with the interaction between HW and SW playing a particularly significant role. An important special feature of embedded systems, however, is that the physical laws of the entire system play a dominating role and must be taken into account during the design. Apart from real-time requirements, designers must also consider resource limits (for example power consumption). This requirement results in a specific adaptation of all phases of the general design cycle of computer systems. During specifying and modeling, real-time constraints and resource limits must be describable, which leads to specific formalisms. The designer must validate and analyze the abstract models during interaction with the surrounding system components (that may in part also be modeled or that may actually exist). In embedded systems, the partitioning into hardware and software is carried out based on the constraints that the system must fulfill. Synthesis is also dominated by the requirement to respect these constraints. As embedded systems usually contain safety-relevant parts and sometimes even take care of the system's safety, particular stringent verification techniques must be applied here. These are especially complex, not least because of the mandatory consideration of real-time aspects.

We have designed this module for students wanting to specialize in those aspects of Embedded Systems and System Software (ESS) that examine the interaction with the physical systems. The specific focus on embedded systems permits combining the module with all other aspects of ESS, such as computer networks or operating systems and distributed systems as part of the specialization area. This module builds on the foundations presented in *Concepts and Methods of System Software* and *Foundations of Computer Engineering*. Students will transfer the general principles to real-time capable system software, mapped to hardware resources and application-specific programming models. They will demonstrate their knowledge through case studies.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Adaptive Hardware and Systems
- Advanced Computer Architecture
- Evolutionary Robotics
- Hardware/Software Codesign
- Intelligence in embedded Systems
- Reconfigurable Computing
- Swarm Robotics

The classes are structured as follows:

- Adaptive Hardware and Systems
    1. Optimization fundamentals

2. Gradient / Steepest Descent und Hill Climbing
3. Statistical analysis of metaheuristics
4. The Metropolis algorithm, simulated annealing, taboo search, variable neighborhood search
5. Genetic algorithms, evolutionary strategies, generic programming
6. Particle Swarm optimization, Ant Colony optimization
7. Multi criterial evolutionary algorithms
8. Neuronal networks

– Advanced Computer Architecture

1. Fundamentals of computer architectures
2. Memory hierarchy design
3. Instruction-level parallelism
4. Data-level parallelism: Vector, SIMD and GPU architectures
5. Thread-level parallelism
6. Warehouse-scale computers

– Evolutionary Robotics

1. Biological fundamentals
2. Evolutionary algorithms
3. Artificial neural networks
4. Reactive intelligence
5. Evolving morphology

– Hardware/Software Codesign

1. Introduction
2. Target Architectures
3. Introduction to Compilers
4. Architecture Synthesis
5. Partitioning
6. Design space exploration
7. Estimation of HW/SW parameters
8. Case Studies

– Intelligence in embedded Systems

1. Application scenarios and architectures
2. Computer vision
3. Sensor fusion
4. Maps and navigation
5. Reactive agents/behavior based computing, affective computing
6. Planning and foundation of cooperative actions
7. Machine learning

– Reconfigurable Computing

1. Introduction & Motivation
2. Reconfigurable Devices
3. Reconfigurable Systems
4. Computer-Aided Design for FPGAs
5. Compilation from High-level Languages
6. System-level Design Methods

7. Application Domains and Examples

– Swarm Robotics
  Please see description of the Module III.3.2 System Software

Usability of the content

(The following text is currently being revised)

Students will apply what they learn in these classes to application development, technical systems, and in designing and implementing special systems. The methods presented for the specification, modeling, analysis, synthesis and verification are required in all application areas of embedded systems, that is, in the entire field of technical systems. However, real-time applications are also applied in non-technical environments, for example, for weather forecasts or for the strategic planning of financial services. Beyond the application reference, the investigation of embedded systems also provides non-negligible insight because one is forced to abandon the fiction of idealism in Plato's sense and to deal with physical boundary conditions.

Prerequisites and prior knowledge

We expect students to have completed *Embedded Systems and System Software* or HW/SW Codesign taught in the second stage of the Bachelor Degree Course. Further prerequisites are the contents of the modules *Foundations of Computer Engineering and Foundations of Computer Architecture* and *Concepts and Methods of System Software* in the Bachelor Degree Course. In addition, students must have a basic knowledge of modeling principles from the module *Modeling* and of programming languages from the module *Programming Techniques* in the Bachelor Degree Course. Students should also be willing to become familiar with system-level programming languages. In some classes, in particular in HW/SW Codesign, we expect students to familiarize themselves with hardware description languages.

Learning goals of the module

Students should understand the specific features of embedded systems and become familiar with the basic concepts for the design of such systems. Students will also recognize potential dangers in the case of faulty design of embedded systems. They should also have a command of the instruments used to avoid such errors. Furthermore, they should be able to assess the specific restrictions that result from the physical laws of the surrounding system and learn to include them systematically into the design process. Finally, students should comprehend the core methods for the precise, predictable use of scarce resources and should apply them to concrete examples.

Teaching of factual knowledge – content competency

After completing this module, students will have learned the

- Relationship between computer and physical system components
- Architecture variants for embedded systems
- Techniques of real-time management
- Techniques for validation and verification
- Methods for designing embedded systems

Teaching of methodological knowledge – methodological skills

After completing this module, students will know the

- Methods for predictable resource planning
- Methods for interacting with physical systems
- Methods for verifying time-dependent systems
- Methods for the targeted partitioning of tasks in HW and SW

Teaching of transfer skills

After completing this module, students will be able to transfer global strategies to specified individual situations, for example as part of exercises.

Teaching of normative evaluation skills

After completing this module, students will

- Develop techniques to apply different strategies
- Recognize the practical value of the concepts and methods of embedded systems

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Activities expected of students: Cooperation during tutorial classes, homework and independent study of secondary literature.

Module assignment

Elective module in *Embedded Systems and System Software*.

Mode

- Credit points for each module (workload) : 8
- Credit points of the classes: 4 each
- SWS (hours per week): 2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week
- Frequency of the module offered: 2-4 classes per year during the winter and summer semesters
- Duration: 2 semesters

Methods of implementation

Tutorials in small groups foster the practical application of the presented methods to selected examples. In particular, students must adapt parameters and strategies to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups in which students will demonstrate the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module
Platzner

III.3.5 HW/SW Codesign

Role of the module in the Degree Course

(The following text is currently being revised)

Embedded systems are important because they play a central role in the continuous computerization of all technical systems. Embedded systems refer to the information processing components in such systems. They usually consist of dedicated hardware and software that builds on it. Both are designed using the fundamental methods of computer science, with the interaction between HW and SW playing a particularly significant role. An important special feature of embedded systems is, how the physical laws, of the entire system, play a dominating role and designers must account for them during the design. Apart from real-time requirements, designers must also consider resource limits (for example, relating to power consumption or available chip space) this context. These requirements result in specific adaptation of all phases of designing computer systems. During specifying and modeling, real-time constraints and resource limits must be describable, which leads to specific formalisms. Designers must validate and analyze the abstract models in interaction with the surrounding system components (that may in part also be modeled or that may actually exist). In embedded systems, hardware and software is partitioned primarily with regard to the specific constraints rather than toward general optimization. The process of hardware and software synthesis is also dominated by the specification of having to respect these restrictions. As embedded systems usually contain safety-relevant parts and sometimes even take care of the system's safety, designers must apply particularly stringent verification techniques here. These techniques are especially complex, non the least because of the mandatory consideration of real-time aspects. However, because the systems under consideration are usually predefined and finite, designers may use methods from hardware verification as a basis for their work.

We designed this module for students wanting to specialize in those aspects of *Embedded Systems and System Software* (ESS) that deal not only with the interaction between hardware and software components, but also with the physical systems. The specific focus on HW/SW codesign permits combining the module with all other aspects of ESS, such as computer networks, operating systems and distributed systems, or embedded and real-time systems as part of the specialization area. This module builds on the foundations presented in the modules *Concepts and Methods of System Software* and *Computer Engineering*. Students will transfer the general principles to the overall design of mixed HW/SW systems and demonstrate them in case studies. The special consideration paid to the physical laws of the surrounding non-computer system components poses specific challenges in this context.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Adaptive Hardware and Systems
- Advanced Computer Architecture
- Hardware/Software Codesign
- High-Performance Computing
- Reconfigurable Computing

The classes are structured as follows:

- Adaptive Hardware and Systems
  Please see description in Module III.3.4 Embedded Systems

- Advanced Computer Architecture
  Please see description in Module III.3.4 Embedded Systems
- Hardware/Software Codesign
  Please see description in Module III.3.4 Embedded Systems
- High-Performance Computing
  Please see description of the class in Module III.3.1 Distributed Computer Systems
- Metaheuristics for Hardware Evolution
  Please see description in Module III.3.4 Embedded Systems
- Reconfigurable Computing
  Please see description in Module III.3.4 Embedded Systems

Usability of the content

(The following text is currently being revised)

Students will apply what they learn in these classes to developing applications and technical systems, and in designing and implementing special systems. The methods presented for the specification, modeling, analysis, HW/SW partitioning, synthesis and verification are required in all application areas of embedded systems, that is, in the entire field of technical systems. Solutions in the traditional environment of information processing can also be optimized in a task-specific way by clever partitioning into HW and SW components. In general, an algorithm can not only be implemented in SW, but can also be implemented by a dedicated HW solution. This approach represents a non-negligible insight for students.

Prerequisites and prior knowledge

We expect students to have completed HW/SW Codesign taught in the second stage of the Bachelor Degree Course. Further prerequisites are the contents of the module *Foundations of Computer Engineering* and *Foundations of Computer Architecture* in the Bachelor Degree Course. In addition, students need a basic knowledge of modeling principles from the module *Modeling* and of the module *Programming Techniques* in the Bachelor Degree Course. Students are also assumed to be willing to become familiar with system-level programming languages and hardware description languages.

Learning goals of the module

The students will understand the specific features of embedded systems and become familiar with the basic concepts for the design of such systems as mixed HW/SW implementations. Students will get to know partitioning criteria for HW/SW and will be able to carry out that partitioning. They will assess the specific restrictions that result from the physical laws of the surrounding system and learn to include them systematically into the design process. Finally, students will learn how to combine specific methods from both software technology and hardware design to achieve a powerful design methodology.

Teaching of factual knowledge – content competency

After completing this module, students will have learned the

- Relationship between computer and physical system components
- HW/SW architecture variants for embedded and real-time systems
- Techniques of HW/SW partitioning
- Techniques for validation and verification

- Techniques for the integrated design of mixed HW/SW systems

Teaching of methodological knowledge – methodological skills

After completing this module, students will know the methods for

- characterizing algorithms with respect to the implementation technique
- technical interaction with physical systems
- verification of time-dependent HW/SW systems
- the targeted design of dedicated HW architectures

Teaching of transfer skills

After completing this module, students will be able to transfer global strategies to specified individual situations, for example as part of exercises.

Teaching of normative evaluation skills

After completing this module, students will

- Develop techniques to apply different strategies
- Recognize the practical value of the concepts and methods of embedded systems

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Activities expected of students: co-operation during tutorial classes, homework and independent study of secondary literature.

Module assignment

Elective module in *Embedded Systems and System Software*

Mode

- Credit points for each module (workload) : 8
- Credit points of the classes: 4 each
- SWS (hours per week): 2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week
- Frequency of the module offered: 2-4 classes per year during the winter and summer semesters
- Duration: 2 semesters

Methods of implementation

Tutorials in small groups foster the practical application of the methods presented to selected examples. In particular, students must adapt parameters and strategies to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups which includes the demonstration of the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module
Platzner

III.3.6 Embedded and Real-Time Systems

Role of the module in the Degree Course

(The following text is currently being revised)

Embedded systems play a central role because of the continuous computerization of all technical systems. Large parts of not only mechanical, automotive and aerospace engineering, but also communication technology can no longer be implemented without embedded systems. Embedded systems refer to the information processing components in such systems. They usually consist of dedicated hardware and software that builds on it. Both are designed using the basic methods of computer science. An important special feature of embedded systems, however, is that the physical laws of the entire system play a dominating role and designers must consider them during the design. This consideration is especially true for the real-time requirements. The real-time aspect can also play an important role in non-technical applications. This aspect results in a specific adaptation of all phases of the general design cycle of computer systems. During specifying and modeling, real-time constraints and resource limits must be describable, which leads to specific formalisms. Designers must validate and analyze the abstract models in interaction with the surrounding system components (that may in part also be modeled or that may actually exist). The synthesis is dominated by the requirement to respect these restrictions. As embedded systems usually contain safety-relevant parts and sometimes even take care of the system's safety, designers must apply particularly stringent verification techniques here. These techniques are especially complex, not only because of the mandatory consideration of real-time aspects.

We have designed this module for students wanting to specialize in those aspects of *Embedded Systems and System Software* (ESS) that deal with the interaction with physical systems. In addition, we intend to teach general issues of real-time processing. The specific focus on embedded and real-time systems permits combining the module with all other aspects of ESS, such as computer networks, operating systems and distributed systems, or HW/SW Codesign as part of the specialization area. This module builds on the foundations presented in *Concepts and Methods of System Software* and *Computer Engineering*. The students will transfer the general principles to real-time capable system software and application-specific programming models, and demonstrate their knowledge with case studies. The special consideration paid to the physical laws of the surrounding non-computer system components poses specific challenges in this context.

Content structure of the module

To complete this module successfully, students select two classes from the following list:

- Evolutionary Robotics
- Hardware/Software Codesign
- Intelligence in embedded Systems
- Operating Systems
- Reconfigurable Computing
- Swarm Robotics
- Vehicular Networking

The classes are structured as follows:

- Evolutionary Robotics
  Please see description of the class in Module III.3.4 Embedded Systems.

- Hardware/Software Codesign
  Please see description of the class in Module III.3.4 Embedded Systems.
- Intelligence in embedded Systems
  Please see description of the class in Module III.3.4 Embedded Systems.
- Operating Systems
  Please see description of the class in Module III.3.1 Distributed Computer Systems.
- Reconfigurable Computing
  Please see description of the class in Module III.3.4 Embedded Systems.
- Swarm Robotics
  Please see description of the class in Module III.3.2 System Software
- Vehicular Networking
  Please see description of the class in Module III.3.1 Distributed Computer Systems.

Usability of the content

(The following text is currently being revised)

Students will apply what they learn in these classes to developing applications and technical systems, and in designing and implementing special systems. Students must know the methods presented for the specification, modeling, analysis, synthesis and verification in all application areas of embedded systems, that is, in the entire field of technical systems. However, real-time applications are also applied in non-technical environments, for example, for weather forecasts or for the strategic planning of financial services. Beyond the application reference, investigating embedded and real-time systems also provides non-negligible insight because one is forced to abandon the fiction of idealism in Plato's sense and to deal with physical boundary conditions, particularly that of a temporal development predetermined by the environment.

Prerequisites and prior knowledge

Prerequisites for *Embedded and Real-Time Systems* are Embedded Systems or HW/SW Codesign taught in the module *Embedded Systems and System Software* in the Bachelor Degree Course. A further prerequisite is the content of the modules *Foundations of Computer Engineering* and *Foundations of Computer Architecture* and *Concepts and Methods of System Software* in the Bachelor Degree Course. In addition, students must possess basic knowledge of modeling principles from the module *Modeling* and of programming languages from the module *Programming Techniques* in the Bachelor Degree Course. We also expect students to be willing to become familiar with system-level programming languages.

Learning goals of the module

The students will acquire an understanding of the specific features of embedded systems and become familiar with the basic concepts for the design of such systems. The students will recognize potential dangers in the case of faulty design of embedded systems. They will also have a command of the instruments used to avoid such errors. They will be able to assess the specific restrictions that result from the physical laws of the surrounding system and learn to include them systematically into the design process. Finally, students will comprehend the core methods for ensuring a precise and predictable system behavior and should apply these methods to concrete examples.

Teaching of factual knowledge – content competency

After completing this module, students will have learned the

- Relationship between computer and physical system components
- Implementation variants for embedded and real-time systems
- Techniques of real-time management
- Techniques for validation and verification
- Methods for designing embedded and real-time systems

Teaching of methodological knowledge – methodological skills

After completing this module, students will know the methods for

- predictable resource planning
- logical interaction with physical systems
- verifying time-dependent systems
- designing systems with inherent intelligence

Teaching of transfer skills

After completing this module, students will transfer global strategies to specified individual situations, for example as part of exercises.

Teaching of normative evaluation skills

After completing this module, students will be able to

- Develop techniques for applying different strategies
- Recognize the practical value of the concepts and methods of embedded systems

Key qualifications

The tutorials in small groups foster the ability to cooperate and work in teams.

Activities expected of students: cooperation during tutorial classes, homework and independent study of secondary literature.

Module assignment

Elective module in *Embedded Systems and System Software*.

Mode

- Credit points for each module (workload) : 8
- Credit points of the classes: 4 each
- SWS (hours per week): 2 lectures + 1 tutorial, 2 lectures + 1 tutorial per week
- Frequency of the module offered: 2-4 classes per year during the winter and summer semesters
- Duration: 2 semesters

Methods of implementation

Tutorials in small groups foster the practical application of the methods presented in selected examples. In particular, students must adapt parameters and strategies to the actual situation. The tasks are worked on in groups of three students, which fosters the ability to work as part of a team. The structure of the exercise sheets maps the structure of the system software, starting with the hardware via processes to resource management and scheduling.

Organizational arrangements / media use / literature references

- Lectures with overhead slides, writing on the blackboard in case of examples, additional explanations and topics to be elaborated on
- Weekly tutorials in small groups including demonstrating the calculations for tutorial exercises and the sample solutions for the homework
- Overhead script is available on the class homepage

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Dressler

III.4 Field: Human-Machine Interaction

III.4.1 Computer Graphics and Visual Computing

Role of the module in the Degree Course

Computer Graphics and Visual Computing focuses on generating images at the computer via scene descriptions, simulated, measured or empirical data, and the capture, analysis, interaction and exchange of image data. It belongs to the modules in Human-Machine Interaction (MMWW).

Content structure of the module:

The module consists of

- the basic class *Advanced Rendering* with the following content:
  - real-time rendering (pipeline technique)
  - Shader languages
  - ray tracing
  - radiosity
  - volume rendering
  - advanced modeling (Bezier, B-splines)
  - texture mapping
  - image-based rendering
  - image-based effects for animation and games
  - non-photorealistic rendering
  - animation
- a range of additional classes, where students select one of the following:
  - Digital Image Processing (Data and Information Visualization)
  - A seminar selected from the field *Human-Machine Interaction* which is eligible for this module.

Topics for additional classes are as follows:

- Computer-Generated Visualization (Data and Information Visualization) with the following content:
  - Intro to visualization
  - Visualization process and data
  - User and task
  - From data to pictures
  - Visual representation
  - Visual analytics
  - Visualization of 3d scalars
  - Visualization of vector fields
  - Systems and tools for visualization

The contents of the other classes are determined as required

Usability of the content

The methods of photorealistic rendering are a recent and dynamic area of computer science. The class *Advanced Rendering* provides knowledge in state-of-the-art interactive real time rendering, and builds the necessary foundations for computer animations and computer games. Because the quantity of data continuously increases (for example medical data, data from space missions, statistical data, monitoring data, etc.) and because it is to be interpreted quickly and correctly by humans (for example surgeons, geologists, environmentalists, etc.), systematic strategies for converting data into expressive and effective images (or image series collections) are required. Computer Generated Visualization concerns itself with these issues. In order to characterize the image data generated in this way, with respect to quality and quantity, and to use transformations and operations for image improvement, image manipulation and image transfer, Digital Image Processing provides the necessary basics for understanding the respective algorithms.

Prerequisites and prior knowledge

Listening to the lecture *Foundations of Computer Graphics* in the Bachelor Degree Course, alternatively: Knowledge from the textbook “Interactive Computer Graphics” written by Ed Angel and OpenGL Programming.

Learning goals of the module

Teaching of factual knowledge – Content Competency

- see Content structure of the module (above)

Teaching of methodological knowledge – methodological skills

After completing this module, students will have learned

- methodical foundations of the algorithms
- efficient algorithms vs. photorealistic algorithms
- practical application of the methods at the computer
- strategic procedures for converting data into images with respect to human interpretation
- transformation into different image spaces
- compression algorithms
- practical implementation of the algorithms at the computer: a fundamental step in order to understand the problem of applying the theory to practice

Teaching of transfer skills

Knowledge in computer graphics and visualization permits generating effective visualizations for application areas such as medicine, biology and chemistry. Furthermore, real time rendering becomes more and more important as a building block of virtual product development.

Teaching of normative evaluation skills

After completing this module, students will understand the

- efficiency assessment of computer graphics algorithms
- quality assessment of a graphics card

- image quality assessment for a specific target group and a specific visualization goal
- assessment of quality loss in image compression

Key qualifications

- Ability to use modern information and communication technologies
- Subject-specific foreign language skills because accompanying literature is in English (for non-native speakers of English)
- Ability to cooperate and work in teams is fostered in group projects
- Activities expected of students: Willingness to reactivate mathematical knowledge from the past; independent programming; cooperation during tutorial classes

Module assignment:

Elective module in Human-Machine Interaction (MMWW).

Mode

- Credit points: 4 + 4 ECTS (2 catalog classes)
- SWS (hours per week): 2+1, 2+1 (or 2+1, 2 if a seminar is selected
- Frequency of the module offered: 2-3 catalog classes per year in the winter and summer semesters

Methods of implementation

Students will expand their knowledge of the theoretical concepts in small groups during tutorial classes and test the methods in lab tutorials.

Organizational arrangements / media use / literature references

- For classes: one two-hour lecture per week and one two-hour tutorial class every second week, or solving of programming tasks in one's own time to a similar extent
- For seminars: either during the semester or as a block seminar, as announced
- Materials used: PowerPoint overhead slides for downloading and exercise sheets
- Literature references for the class Computer Graphics: Angel, *Interactive Computer Graphics*, Addison-Wesley; or Watt, Three Dimensional Computer Graphics, Addison-Wesley; or Foley et al., *Computer Graphics,* Addison Wesley Publishing
- Literature references may be found on the lecturer's website

Examination modalities

Please see chapter "Grading Policies" at the beginning of this Module Handbook.

As a partial performance we require:

- Independent programming of (parts of) the rendering pipeline or tasks adapted to the additional topics
- Project work
- Presentation of results
- Written examinations may be required as a partial examination

The weighting of the partial performances is announced at the beginning of each semester

Person responsible for the module

Domik

III.4.2 Computer Science and Society

Role of the module in the Degree Course

Computer scientists develop products that are based on characters (programs, specifications, documentations, etc.). In contrast to other engineering products that are manufactured from materials such as steel, plastics or glass, software maps social reality into a wide range of shapes. Through its use, this reality changes. This change leads to many different interactions between computer systems and their application environment; by use of the system, the behavior of people that is modeled or anchored in terms of assumption in the system, is changed. The application environment reacts upon the product – followed by revision, adjustments and expansions. We have to recognize these interactions as soon as possible in order to avert danger and to anticipate future adjustments.

Content structure of the module

The module consists of classes studying interactions between computer systems and their application environment. The goal is to examine the interaction of specific technologies with cognitive, social, economic and political factors and, while doing so, to identify risks as well as chances. The class *Computer Science and Society* teaches the corresponding theoretical and conceptual basics which are completed and deepened in further classes. Either two of the following lectures are mandatory to complete this module:

- Computer Science and Society
- Concepts of Digital Media
- Assistive Technologies, Accessibility

You will find a description of the class *Assistive Technologies, Accessibility* in module III.4.3.

Alternitively, the student may also replace one of the classes by a seminar from the *Field Human-Machine-Interaction* which is eligible for this module.

Usability of the content

Students gain basic knowledge about the possibilities and limitations of designing computer systems and of formalization. These insights are necessary both for assessing technical potentials and for managerial positions when handling software projects. The study of interactions also provides an advanced understanding of problems and potentials of IT in different application contexts. Considerations regarding the history of data processing integrate current concepts of computer science into a wider cultural and historic framework.

Prerequisites and prior knowledge

The basic knowledge from the Bachelor Class is a prerequisite.

Learning goals of the module

Students learn to examine interactions between computer science systems and their application environment. For this reason, students need to be able to differentiate between technical and non-technical problems and to adequately relate these to each other. Furthermore, they should be able to evaluate and compare actual technological developments and computer science systems as well as to assess new innovation potentials in the media field.

Teaching of factual knowledge – Content Competency

Students learn theoretical and conceptual basics in order to examine the interaction between computer science and its application context. For this reason, the class covers the corresponding cognitive-psychological, sociological and economic basics. On this basis, aspects of the history of data processing as well as the latest developments in computer science are examined and evaluated.

Teaching of methodological skills

The students learn to be able to evaluate the chances and risks of the application of computer science systems. Here, the focus lies on the ability to separate technical potential and usage potential in order to determine the risks and factors of embedding for a successful application.

Teaching of transfer skills

In principle, the basic concepts and techniques are also applicable to other fields of technical design and evaluation. The ability to be able to adjust technical and non-technical vectors is fundamental for all communicative and cooperative actions in computer science system development.

Teaching of normative evaluation skills

The class should teach basics of human-machine interactions to the point that the students are able to solve standard problems as well as to identify fields that require other scientific skills. The students will be enabled to work on ethical considerations and on a value-conscious design of technical artifacts.

Key qualifications

Expected contribution of the module to teaching key qualifications

- Competency in working out scientific results on the basis of original literature from other disciplines, too.
- Ability to analyze and evaluate modern IT-technologies.
- Ability to present and discuss scientific approaches.
- Connecting knowledge for interdisciplinary cooperation.

Module assignment

Elective module in Human-Machine Interaction (MMWW).

Mode

- Credit points: 4 + 4 ECTS (2 catalog classes)
- SWS (hours per week): 2+1, 2+1 (or 2+1, 2 if a seminar is selected)
- Frequency of the module offered: Annually

Methods of implementation

– Basics are introduced during lecture
– Students contribute the results that they have worked out to the discussion in small groups
– Concepts and techniques are deepened by tutorial classes in small groups

Organizational arrangements / media use / literature references

- Lecture with script/overhead slides and accompanying literature
- Exercises: Tutorial classes in small groups with exercise sheets and presentation of work results
- Expected activities from the students: Collaboration in tutorial classes, reading and presentation
- Web-based lecture material, complementing literature

Examination modalities

Please see chapter “Grading Policies” at the beginning of this Module Handbook.

Person responsible for the module

Selke

III. 4.3 Accessible Human-Computer Interaction

Role of the module in the Degree Course

So called information and communication technologies (ICT) are used by a constantly increasing user group for many different interests and purposes. However, people with disabilities / impairments – a significant part of the population (about 20%) – are either excluded or are having serious problems in using current main-stream technologies. They would benefit from proper solutions improving their quality of life, including them better in society and providing accessibility not given by main-stream technology. The module addresses design, evaluation and individual assignment of technological solutions – mainly interactive systems and devices – for people with impairments. The central focus is on understanding people / users by knowledge and findings of neuropsychology regarding motor, sensory and cognitive capabilities needed for using systems. Are capabilities impaired or lost then solutions demanding alternative, still existing, capabilities are providing accessibility. The module can be equally seen as a theoretical and practical deepening of existing knowledge on human-computer interaction as well as a special introduction to theory and methods of human-computer interaction.

Content structure of the module

The module contains a central class on understanding users and their motor, sensory and cognitive processes based on models and findings in neuropsychology, on understanding the range of impairments and on understanding the human capabilities technology is demanding. Design principles and criteria for special solutions are reflected on this understanding of the user. Hence, interdisciplinary knowledge and approaches are taught. The module consists of the following classes:

- *Assistive Technologies, Accessibility*
- *Usability Engineering Practice*
- *Modelling User Interfaces*
- *Web Modelling*
- *Computer Science and Society*
- *Seminar in the field of Human-Machine Interaction*

In the mandatory class *Assistive Technologies, Accessibility* the following topics (among others) are covered

- range of motor, sensory and cognitive impairments
- impact of these impairments to the use and usability of main-stream technologies
- multi-modal alternatives of human-computer interaction for people with impairments
- design criteria for technologies useable for people with impairments
- special solutions for special impairments (e.g. blindness, vision impairments, motor impairments, memory problems) like alternative input devices and methods, brain-computer interfaces, haptic interfaces, speech input, voice output, or magnification.

The very first topic in the class is the role of people with disabilities / impairments in the society so awareness and understanding of the needs of special user groups are guaranteed. Examples of typical scenarios of use demonstrate the importance of special solutions (assistive technologies, accessible artifacts enabling accessibility) for people with impairments for their self-determination, empowerment and inclusion. Critical discussion of current practice and trends like guidelines for web accessibility, guidelines for universal design or solutions available on the market (e.g. screen reader) round off the class.

Details on the class *Usability Engineering Practice* can be found in module III.4.5, details on the class *Computer Science and Society* can be found in module III.4.2, details on *Modelling User Interfaces* and *Web Modelling* can be found in module III.4.6.

Usability of the content

Detailed considerations of human motor, sensory and cognitive capabilities needed for the interaction with technological artifacts provide basic psychological knowledge that is of general importance for design and evaluation of systems to be used by people. The special focus on users with different impairments enables decisions on design and use of systems for special user groups. The additional classes are either focusing on the social relevance and aspects of systems (*Computer Science and Society*) or on general methods of system design and evaluation (*Usability Engineering Practice*). Both topics are highly relevant for the design and evaluation of technological solutions for people with impairments.

Prerequisites and prior knowledge

The prerequisite for *Assistive Technologies, Accessibility* is fundamental knowledge of human-computer interaction as taught in the bachelor class *Basics of Human-Machine Interaction.*

Learning goals of the module

Students will learn how interrelations between users and systems are analyzed. Systems are the product of construction through a design process containing assumptions and decisions on intended users. In order to use them effectively, efficiently and with positive attitudes, people need certain capabilities (capability-demand relationship). Through examples of different impairments students will learn to analyze systems with respect to their motor, sensory and cognitive demands and to identify alternative technological solutions for people with impairments.

Teaching of factual knowledge – content competency

Students will acquire knowledge of theoretical and conceptual foundations describing the relationship between a user and an artifact used to accomplish tasks. On the user's side, these are cognitive resources and processes like memory with its different forms, perception (visual, auditory, haptic), attention and language. Typical impairments are discussed for each of these capabilities. Similar knowledge of relevant motor and sensory capabilities will be also acquired. Based on that knowledge, alternative technological solutions like different forms of input and different forms of representing information are taught and evaluated. Those solutions can, under certain circumstances, also be relevant for users without significant impairments. Examples are eye tracking, speech input, brain-computer interfaces or haptic interfaces. Students will also learn current laws, conventions and guidelines addressing accessibility and design for people with impairments.

Teaching of methodological knowledge – methodological skills

After completing this module, students will be able

- to apply methods for identifying the demands of a system on a user's capabilities so a design or an existing system can be evaluated,

- to apply methods for identifying motor, sensory and cognitive processes, relevant for the use of a system, and for inclusion in a user’s profile.

Teaching of transfer skills

The content of all three lectures is in many ways multi-disciplinary. This holds especially for the mandatory class *Assistive Technologies, Accessibility* with its particular stress on neuropsychology and its impact on the design of accessible systems.

Teaching of normative evaluation skills

The mandatory class is especially suited to provide the students with numerous methods and hints to assess the suitable of a particular system for an individual user profile described by given impairments and alternative capabilities. The class *Usability Engineering Practice* enables students to assess the usability of a system.

Key qualifications

It is expected that this module supports the transfer of key qualifications in the following ways:

- Working through the classes contents by delivering solutions to written homework exercises
- Ability to co-operate in a team through collaborative work on small homework projects
- Competence in presenting, as every student is forced to present his/her solutions
- Ability to assess accessibility and usability of a system designed for people with particular impairments
- Ability to assess the degree of inclusion / exclusion a system is providing

Module assignment

Elective module in the area of *Human-Machine Interaction* (MMWW)

Mode

- Credit points: 4 + 4 ECTS (2 catalog classes)
- SWS (hours per week): 2+1, 2+1 (or 2, if a seminar is selected as additional class)
- Frequency of the module offered: 2-3 catalog classes per year in the winter and summer semesters

Methods of implementation

The classes of the module are intensely supported by *koaLA*, the e-Learning system of the university. Slides are published there, written exercises are distributed from here, and students need to submit their solutions to koaLA. During the exercise hours, students present their solutions to the group, which includes the creation of adequate PowerPoint slides, concepts and methods taught in the lecture are deepened. Discussions and questions are encouraged which also holds for the lectures.

Organizational arrangements / media use / literature references

- Lectures with presentation of slides
- Exercise hours: attendance is mandatory, students present their solutions which are discussed

Examination modalities

See the paragraph on "Evaluation of modules" in the beginning of this module handbook

Person responsible for the module
Tauber

III.4.4 Computer-Supported Cooperative Work and Learning

Please note that from SS2015 on, the basic lecture in this module *Cooperation Support Systems* is not offered any more. For this reason, this module cannot be chosen any more.

III.4.5 User Interface Development

Role of the module in the Degree Course

This module deals with the aspects of user interface development. Emphasis here is on a selection of methods for usability engineering – hence a class on those is mandatory for passing this module. Based on this, the students can either deepen their knowledge about users with special needs in the class *Assistive Technologies, Accessibility*, or alternatively the class *Modeling User Interfaces* can be chosen to complement the practical usability aspects with model-based development concepts. All three classes emphasize a design-oriented approach, less so the analytic viewpoint. Hence, this class can be seen in conjunction with other classes dedicated to a constructive way in the fields of *Software Technology* or *Embedded Systems and System Software*.

Content structure of the module

The module consists of two of the following classes:

- *Usability Engineering Practice*
- *Assistive Technologies, Accessibility*
- *Modelling User Interfaces*
- *Web Modelling*
- *Seminar in the field of Human-Machine Interaction*

The lecture *Usability Engineering Practice* is a sequel to the Bachelor class *Usability Engineering*. It treats with much more detail in theory and practice selected methods, such as Cognitive Walkthrough, Card Sorting, or Value-Centered Design. Generally spoken, the class deals with concepts and methods which widen the view on the term "Usability Engineering" in different directions, including among others User Experience, Extreme Usability, Esthetics, Health and Security aspects.

Details about the class *Assistive Technologies, Accessibility* can be found in Module III.4.3; *Modeling User Interfaces* and *Web Modelling* are introduced in Module III.4.6.

Usability of the content

A detailed and deepened insight into usability concepts and methods is the core of this module. The mandatory class is always updated to the most current new methods and ideas in the field. Both additional classes cover constructive aspects – either towards special needs, or towards properly model-based abstraction concepts.

Prerequisites and prior knowledge

The prerequisite for *Usability Engineering Practice* is fundamental knowledge of technology from Usability Engineering as taught in the bachelor class *Usability Engineering*.

Learning goals of the module

Teaching of factual knowledge – content competency

The students will learn

- current usability methods and approaches
- current concepts and techniques for the support of users with special needs
- current modeling techniques from the field of model-based user interface development

Teaching of methodological knowledge – methodological skills

After completing this module, students will be able to

- apply innovative methods of Usability Engineering, including the supporting software tools
- apply technologies adequately to adapt user interface to the user's special needs
- apply modelling concepts, techniques, and tools

Teaching of transfer skills

The content of all three lectures is in many ways multi-disciplinary. This holds especially for the mandatory class *Usability Engineering Practice*, but also for *Assistive Technologies, Accessibility*.

Teaching of normative evaluation skills

The mandatory class is especially suited to provide the students with numerous methods and hints, to assess the usability of user interfaces. The class *Assistive Technologies, Accessibility* uses large parts of its presentation to enable the students to learn about assessment of technology for a user's special needs. The modeling class deals explicitly with practical usefulness of and reasoning behind model-based techniques; hence, the students will be able to assess model-based approaches for practical projects.

Key qualifications

It is expected that this module supports the transfer of key qualifications in the following ways:

- Working through the classes contents by delivering solutions to written homework exercises
- Ability to co-operate in a team through collaborative work on small homework projects
- Competence in presenting, as every student is forced to present his/her solutions
- Ability to asses usability, assistance, and modelling concepts
- Subject-specific foreign language skills because the whole class, tutorials, and exams are held in English

Module assignment:

Elective module in the area of *Human-Machine Interaction* (MMWW).

Mode

- Credit points: 4 + 4 ECTS (2 classes)
- SWS (hours per week): 2+1, 2+1 (or 2, if a seminar is selected as additional class)
- Frequency of the module offered: 2-3 catalog classes per year in the winter and summer semesters

Methods of implementation

The classes of the module are intensely supported by *koaLA*, the e-Learning system of the university. Slides are published there, written exercises are distributed from here, software (such as modeling tools) is provided. During the classes, interactive group work is applied wherever appropriate. During the exercise hours, students present their solutions to the group, which includes the creation of adequate PowerPoint slides. The solutions are assessed and honored with points (or not), and every student has to reach a minimum level of points. During the classes discussions and questions are encouraged.

Organizational arrangements / media use / literature references

- For classes: one two-hour lecture and a one hour tutorial class per week (or one two-hour tutorial class every second week). In the exercise hours students present their solutions, which frequently contain solutions of programming tasks, development of modeling documents, or results from usability tests.
- References regarding activities expected of students: independent solving of the exercises, sometimes programming, usability evaluations, etc.
- Materials used: exercise sheets, usability tools, assistive technologies, modeling tools

Examination modalities

See the paragraph on "Evaluation of modules" in the beginning of this module handbook

Person responsible for the module

Szwillus

III.4.6 Model-Based Development of User Interfaces

Role of the module in the Degree Course

This module introduces the students to current approaches from the field of model-based development of user interfaces. The term 'model' is in the focus, and a clear separation of aspects, artifacts, and corresponding tasks is provided. If possible, these general considerations are supported by appropriate tools (modeling software, simulators). This module establishes a link to corresponding model-based software development as treated in the field *Software Technology*.

Content structure of the module:

The module consists of a number of classes; two of these classes have to be taken:

- *Modelling User Interfaces*
- *Web Modelling*
- *Usability Engineering Practice*
- Seminar in the field of *Human-Machine Interaction*

The two classes are structured as follows:

- *Modelling User Interfaces*

    - Foundations of Modeling
    - The Model-based Development Process
    - Task Analysis and Task Modeling
    - Dialogue Modeling
    - User Interaction Modeling
    - Control Modeling

- *Web Modelling*
  This class covers current technology and concepts for the model-based development of web sites. If possible, the corresponding tools are provided and used in practical exercises. Goal of this class is to provide an overview of problems and advantages of model-based concepts in contrast to conventional, popular development environments. Especially, we deal with the web modeling approaches WebML and UWE as most prominent examples.

- The class *Usability Engineering Practice* is described in module III.4.5.

Prerequisites and prior knowledge

The prerequisite for the *Web Modeling* class is *Modeling User Interfaces*; hence it can only be heard as second class.

Learning goals of the module

Teaching of factual knowledge – content competence

Both modeling classes deal explicitly with practical aspects of the modeling process. This enables a clear separation of the different aspects, playing an important role during user interface development, and which aspects on which level can be treated with which modeling approach. The mandatory class lays the ground, while the optional class covers specialized modeling approaches for web site development.

Teaching of methodological skills – methodological competency

In both classes of the module the students learn to apply the modelling approaches. We consider academic as well as commercial modelling tools and use them practically during the exercises.

Teaching of transfer skills

It is the nature of modelling approaches to support abstraction concepts. Hence, techniques introduced within the field of user interface or web site development – such as State Charts, Petri nets, Finite Automata – can be transferred to other fields.

Teaching of normative evaluation skills

The students experience the value and positive outcomes, but also the problems and disadvantages of the single modelling approaches and the general model-based approach. As a result they are enabled to evaluate the worthiness of applying model-based approaches and which potential risks and benefits are linked to it.

Key qualifications

It is expected that this module supports the transfer of key qualifications in the following ways:

- Working through the classes contents by delivering solutions to written homework exercises
- Ability to co-operate in a team through collaborative work on small homework projects
- Competence in presenting, as every student is forced to present his/her solutions
- Ability to asses usability, assistance, and modelling concepts
- Subject-specific foreign language skills because the whole class, tutorials, and exams are held in English

Module assignment:

Elective module in the area of Human-Machine Interaction (MMWW).

Mode

- Credit points: 4 + 4 ECTS (2 classes)
- SWS (hours per week): 2+1, 2+1 (or 2, if a seminar is selected as additional class)
- Frequency of the module offered: 2-3 catalog classes per year in the winter and summer semesters

Methods of implementation

The classes of the module are intensely supported by *koaLA*, the e-Learning system of the university. Slides are published there, written exercises are distributed from here, software (such as modeling tools) is provided. During the classes, interactive group work is applied wherever appropriate. During the exercise hours, students present their solutions to the group, which includes the creation of adequate PowerPoint slides. The solutions are assessed and honored with points (or not), and every student has to reach a minimum level of points. During the classes discussions and questions are encouraged.

Organizational arrangements / media use / literature references

- For classes: one two-hour lecture and a one hour tutorial class per week (or one two-hour tutorial class every second week). In the exercise hours students present their

solutions, which frequently contain solutions of programming tasks, development of modeling documents, or results from usability tests.
- References regarding activities expected of students: independent solving of the exercises, sometimes programming, usability evaluations, etc.
- Materials used: exercise sheets, usability tools, assistive technologies, modeling tools

Examination modalities

See the paragraph on "Evaluation of modules" in the beginning of this module handbook

Person responsible for the module

Szwillus

III.5 Cross Area Matters

III.5.1 Project Group

In this module, we describe the project group regardless of any actual content aspects of the class.

Role of the module in the Degree Course
The project group is an essential part of the Master Degree Course. As a two semester class, the student will usually start in the 1st or 2nd semester of the Master Degree Course and finish in the following 2nd or 3rd semester. On the one hand, the project group is intended to support the personality development of the participating student while, on the other hand, it is intended to achieve the goals according to the contents of the work of the project group.

Within the project, teamwork and organization of a project is practiced and learned; this prepares the participating student for future jobs in industry. The student will get to know extensive development processes from his/her own view within the team. Due to the explicit need for division of labor, the student is forced to report on his/her work in the group and defend the results.

With regard to the content, the project groups introduce the student to actual research areas which typically belong to the interest area of the organizer. Therefore, project groups contribute – not primarily but also – to university research. This means that the individual student, after having completed the project group, he/she will generally be predestined to continue with a thesis in the corresponding field.

Content structure
The top organization principle of the project group should be to realize **self-organization** as far as possible. This may be achieved by

- discussion at the beginning of the project group on the goals set or to be set together with the organizer;
- preparation of knowledge on the subject and choosing the systematic procedures, methods and tools that are relevant for this subject – typically during an initial **seminar phase**;
- consequently delegating jobs, i. e. **spreading responsibilities** within the group;
- revealing and supporting of special talents which exist within the group or which result from seminar lectures or from delegation of tasks;
- developing a **process-oriented personnel structure** similar to an industrial development team; delegating sub-tasks to smaller groups who will report afterwards;
- **regular reports** on the work of individual students and small groups;
- a written interim and final report contributed on by all group members.

Usability of the content
The usability of the content depends on the specific subject of the project group. In any case, the project group will take the participants near some parts of the latest computer science

research and may, therefore, be an important preparation for a Master Thesis close to research.

Prerequisites and prior knowledge
With regard to the **contents**, the **prerequisites** depend on the subject chosen. The **formal prerequisites** ensure that a project group can only be started after having successfully completed the Bachelor Degree Course. In case of a consecutive Bachelor/Master Degree Course at the University of Paderborn, an overlap of the project group and writing of the Bachelor Thesis is avoided despite of the flexible interim regulations in between Bachelor and Master Degree Course, considering the existing work load of the student.

Learning goals
Regardless of the work field and specific subject of a project group, the participating student should learn to proceed methodically and systematically according to each individual work field within the project. Should software design be the main goal of the project, the methods and techniques of software design should be applied systematically just as they are taught in the Computer Science Course. Apart from the contents, we mainly teach methodological skills and normative evaluation skills with regard to the aspects of cooperative work on computer science problems and, more specifically, on cooperate software design.

Key qualifications
The project group contributes considerably to the achievement of key qualifications by its large scale and its high grade of self-organization. The participating student is introduced intensively to the lab project work within an industry-similar setting with regard to team size, project complexity, project duration as well as communication skills. Furthermore, presentation and moderation skills are efficiently learned because even interim results are being presented and defended over and over as well as discussed within self-organized meetings. The literature research for each subject has to be done systematically which teaches the student to apply learning strategies. In fact, the force to cooperate and work not only with German speaking students supports the intercultural competencies and trains – parallel to mainly English literature studies – special foreign language skills.

Module assignment
The Project Group is a mandatory module. The subject can be chosen from a wide spectrum due to the extensive range – there are about 3 to 5 new starting project groups in each semester. In particular, this module is not part of a certain area and therefore does not serve to cover a certain area of the Master Degree Course.

Mode

- Credit points: 30 ECTS

- Range and form: Typically, the participants and the organizer of the project group meet weekly; there will be additional meetings with part of the group depending on the work distribution. An interim/final report has to be presented at the end of each semester.

- Frequency: Project groups start off newly in each semester. Around the end of term, the latest planned project groups for the following semester are being introduced and advertised in a public presentation (“Projektgruppenvorstellung”) to the Master students. Afterwards, the students have to apply for the project groups.

- Duration: 2 semesters

Methods of implementation, organizational arrangements

- The number of participants is limited to approximately 12 persons.
- There are **plenum meetings** (all participants and the organizer) to especially teach common necessary knowledge (seminar phases at the beginning of both semesters) and to plan future work.
- Fixed **responsibilities** are distributed among all participants which may remain this way for the entire project time or just for a short duration (ad hoc tasks).
- **Sub-groups** are established for each subject; the sub-groups will work independently and in time while also having to defend their work in front of the plenum.
- Typically, each project group will present its work on a **website.**
- A **repor**t has to be written at the end of each of the two semesters, filled with contents and created in every aspect by the participating students.

Examination modalities
The teacher supervises and evaluates the performance of the student within the project group during the entire project period. For the final grade, he/she will consider the contribution to the **project group result** (implementation, for example), the contribution to **project group reports** as well as the result of the final **oral discussion** which is normally about as long as an oral examination. The grade of the entire module will be a summary of all above listed points.

Person responsible for the module
Fischer

III.5.2 Master Thesis

This module describes aspects of the final thesis of the Master`s Degree Course disregarding the subject.

Role of the module in the Degree Course
A Master Thesis consists of working on a subject, including a written report and an oral presentation of the results. With the Master Thesis, the student shows his/her ability to work independently and scientifically on a demanding subject which also enables him/her to create his/her own ideas. On a "state-of-the-art" basis, the methods of computer science should be implemented systematically. The writing of a Master Thesis is a full-time work (30 ECTS-points) in the curricular of the 4th semester.

Content structure of the module
A Master Thesis consists of four phases: Determination of a subject, planning of the work (this will take about one month), writing the thesis (fix term of five months and formally supervised by the Examination Office) and the presentation of the results.

- **Determination of a subject**: Every professor and every scientific employee with a doctorate of the Institute of Computer Science who has already taught on his own is entitled to assign a subject. Subject proposals of the student may be considered. Furthermore, a subject may result from the participation of a student in a project group.
- After informally having agreed on a subject, the student will draw up a **work plan** in coordination with the supervisor. The designated time for this period is one month of full-time work (ECTS 5). For this reason, the planning of the work should explore the subject adequate and thoroughly, and the student should carefully plan it. The work plan should reflect this preparation accordingly by the depth of the contents and completeness of the relevant aspects and it should include the following elements:

    - Description of the **subject**
    - **Motivation** for the thesis
    - Explicit formulation of the goal
    - Considered approaches of the strived contribution to **computer science research**
    - Description of the necessary work in order to reach the goal, including the corresponding **time table**
    - A table of the preliminary **structure** of the written thesis

- **Writing the thesis:** After the work plan has passed, the student has to register formally with the Examination Office and with the supervisor in order to notify both of the beginning of the fixed duration. The knowledge level of the Master Thesis is adjusted to the class contents of a complete Bachelor Degree Course as well as to preparatory deepening classes on the subject and, if necessary, it may even be adjusted to the contents of a preceded project group on this subject. The supervisor guarantees that the Master Thesis can be finished appropriately within the given period. He is available for the entire working period and checks regularly on the progress of the work. In case there are any problems - if it turns out that a subject may not be worked

on in the intended way, or that the duration of the work may exceed the fixed term according to the regulations of the conduct of examination - the supervisor intervenes and directs the student accordingly.
The supervisor also helps with the written report and points out mistakes.

- **Presentation**: Typically - but not mandatory – the student will present his/her results in a public lecture with a discussion at the university at the end of or after having finished the written work. This presentation is also considered for the evaluation of the Master Thesis.

Prerequisites and prior knowledge
With regard to the **content**, the Master Thesis is based on fundamental knowledge and skills of computer science (as they are obtained in the Bachelor Degree Course) –; sometimes it may also be based on some deepening classes of the Master Degree Course that the student has taken before (1. - 3. semester) and it is often based on a preceding project group.

Formally, a Master Thesis can only be started with after having successfully completed an essential part of the Master Degree Course (54 ECTS-Points). This guarantees that the Master Thesis is done during the later classes of the Degree Course and therefore it may deal with more demanding subjects.

Learning goals
With the Master Thesis, the student shows his/her ability to work independently in a scientific way on a suitable demanding subject which also enables him/her to create his/her own ideas. On a “state-of-the-art” basis, the methods of computer science should be applied systematically.

The student should show that he/she can handle a subject of computer science on the basis of scientific methods within a given period of time. This includes the proof of the ability to use skills and good working knowledge in literature studies, but also to use previous results and/or relevant development tools. The student has to prove the ability to edit and present a result as well as show how he/she achieved this result. He/she has to present the goals set and, furthermore, the necessary basics have to be processed in a structured and stylistically correct way and on an appropriate abstraction level. For the Master Thesis, the development of a research-relevant contribution of the student himself/herself is mandatory. The publication of the results or parts hereof is definitely wanted and supported in some cases of extraordinary performance.

Key qualifications
Through the obligatory work with literature, the Master Thesis promotes explicitly the development of knowledge acquiring strategies as well as in most cases acquisition of specific foreign language skills. Furthermore, the presentation of working results in computer science in a written and oral form is explicitly demanded and therefore promoted.

Module assignment
The Master Thesis is a mandatory module. However, its contents may be chosen from a wide spectrum and - as described above –the student may influence parts of it himself/herself.

Mode

- Credit points per module: Planning of the work 5 ECTS; writing and presentation 25 ECTS)

- Content structure of the module: The period of time designated for writing a Master Thesis by the *Regulations for the Conduct of an Examination (Prüfungsordnung)* is 5 months of full-time (about one semester).

- Frequency: To be freely defined between student and supervisor

- Duration: Typically, 1 month planning + 5 months writing

Methods of implementation
As described above, an adequate supervision of the student is essential for the support of the student. Besides this, the student has to become active in an extensive independent way and with adequate initiative on himself/herself with regard to the definition of a subject as well as with regard to the work itself. The student will listen to presentations of Master Theses of fellow students when attending the "Oberseminar" of the working group, which is usually requested by the supervisor, and this may also serve as an orientation for himself/herself.

Examination modalities
The module III.5.2 Master Thesis is being examined and graded according to §18, Abs. 2 of the *Regulations for the Conduct of Examination (Prüfungsordnung).*

Person responsible for the module
Szwillus