Advanced Software Technology

Master of Science

Subject-specific Examination Regulations for Advanced Software Technology

The subject-specific examination regulations for Advanced Software Technology are defined by this program handbook and are valid only in combination with the General Examination Regulations for Master degree programs ("General Master Policies").

This handbook also contains the program-specific Study and Examination Plan in chapter 2.2.

Upon graduation students in this program will receive a Master of Science (MSc) degree with a scope of 120 ECTS credit points (for specifics see chapter 2 and 5 of this handbook).

Valid for all students starting their studies in Fall 2023.

Version	Valid as of	Decision	Details
Fall 2023 – V1.2	Sep 01, 2023	June 28, 2023	Academic Senate approval of study program name change from "Data Science and Software Development" to "Advanced Software Technology"
Fall 2023 – V1.1		May 26, 2023	Editorial changes in all handbooks by Program Support and Development
Fall 2023 – V1		May 24, 2023	Originally approved by Academic Senate

Contents

1 Program Overview

1.1 Concept

The Master of Science in Advanced Software Technology at Constructor University is a consecutive master program that prepares students to become the next generation of experts in the field of advanced software technology. The program offers a unique opportunity to gain a solid education in software development, data science and programming languages, which are at the forefront of digitalization and are driving the digital transformation of industry and society. The program is designed to provide students with a solid foundation in Mathematics and basic programming skills, a comprehensive understanding of the latest research and technology in these areas, as well as essential management and leadership skills, so that they can become technology leaders in research and industry.

The program offers three tracks: Data Science, Software Development, and Programming Languages, allowing students to specialize in the area of their choice. The program also includes common modules for all students, such as Architectural Strategy, Programming Languages in Software Development, Big Data Software Engineering, Capstone Project, Product Innovation & Marketing, Quality Engineering, Kotlin Ecosystem, Data Analytics, and Agile Product Development & Design.

The special modules for the Data Science track include Advanced Deep Learning, Recommender Systems, Computer Vision, Machine Learning in Software Engineering, and Bayesian Methods in Machine Learning. The special modules for the Software Development track include Static Program Analysis, Mobile Application Development, System Security, Distributed Ledge Technology Smart Contracts and Cryptography, Network Security and IDE Development. For the Programming Languages track, the special modules are Advanced Functional Programming, Weak Memory Models, Virtual Machines, Metacomputations, Dependent Types, Homotopy Type Theory, and Category Theory for Programmers.

The program will be taught by distinguished experts in the field from Constructor University and JetBrains, guaranteeing excellent teaching competence and hands-on experience from the forefront of the state of the art in research and industry. In addition, students will have access to real-world applications and the IT job market via JetBrains' excellent international network, and will be supported by the Constructor University Student Career Support.

The program will also make use of contemporary blended e-learning techniques, flipped classroom teaching, and team-based work on software projects, allowing for a student-centric and hands-on experience. Together with the availability of state-of-the-art software and hardware at Constructor University and the support of JetBrains, the program allows seamless collaboration among students and instructors of different institutions, and adapts to conditions that may arise from pandemic emergencies.

Students will acquire the core expertise of digital leaders, with a solid technological backbone developed along three complementary tracks, with additional core management and leadership skills. They will acquire the essential soft skills for an active digital technology leadership in the

contemporary global and multiethnic society, thanks to the international environment that characterizes Constructor University and JetBrains. Overall, this education will enable them to enter research via Ph.D. programs and to succeed in the job market in high profile roles.

1.2 Qualification Aims

1.2.1 Educational Aims

The MSc Advanced Software Technology program at Constructor University aims to provide students with an in-depth understanding of the essential aspects of designing and development of software products with a focus on Data Science. The program comprises three main tracks: Data Science, Software Development, and Programming Languages Tools. Students will acquire the skills necessary to apply methods and tools to successfully and responsibly engineer software, with a special emphasis on the use of JetBrains tools.

The program seeks to expand the participant's competencies and capabilities in the subject areas of Data Science, Software Development and Programming Languages, which play a dominant role in industries and research. Each student will select one of these areas as their main specialization, and the curriculum will provide them with modern cross-disciplinary leadership and management competencies to become tomorrow's digital leaders.

Throughout the program, students will be introduced to practical and research-oriented work through a Capstone project, an elective research project, and a thesis, which will be supported by frequent individual feedback sessions and personal guidance. This will facilitate and quicken the students' career development and help them to become valuable assets in industries and research within a short period of time.

Constructor University programs are offered in a highly intercultural environment. Students will acquire intercultural competence as part of their education through everyday group work, class participation, and extracurricular activities. In this way, students will gain practical intercultural competencies and build their confidence in an English-speaking work and study environment.

To summarize, graduates of the MSc Advanced Software Technology program will have obtained the following competences and skills:

1. Subject-matter competence in a Data Science, Software Development or Programming Languages specialization

Graduates will have an in-depth knowledge of one of the fields of Data Science, Software Development or Programming Languages. They will be able to define and interpret the doctrine of the field, and will have also developed a detailed and critical understanding at the cutting edge of knowledge in the field.

2. Advanced Software Technology Competency

Graduates will have a broadened and deepened knowledge in their formal, algorithmic, and applied competencies in Advanced Software Technology. This will enable them to develop independent ideas as digital experts.

3. Learning, transfer, and research skills

The Program will enable students to apply problem solutions in new and unfamiliar situations. They will integrate learned skills in complex and multidisciplinary contexts, as it is more and more necessary in industry and research. In particular, graduates will be able to design research questions, select appropriate methods, and document and interpret research results.

4. Management and Leadership Skills

Recognizing the ever-increasing need for management and leadership skills in business, industry and research, graduates will have a broad and integrated knowledge and understanding of the fundamentals from management and leadership. Their knowledge corresponds to the standard literature in the field. In particular, they will be able to solve related problems in the field of Advanced Software Technology with professional plausibility.

5. Teamwork and communication skills

Graduates will be proficient in the specialized exchange of ideas in a group setting with the goal of collaborative development of a digital software or hardware system. This will be reinforced by effective and reflective practice of communication and collaboration on both academic and non-academic topics.

6. Personal and Professional Competence

Graduates will be able to make, justify and reflect on decisions based on theoretical and professional knowledge. They will be able to critically examine their own behavior and assess social consequences. In doing so, they will act appropriately to the situation. Thus, they will be able to develop a professional profile both in and out of academia.

1.2.2 Intended Learning Outcomes

Upon completion of this program, students will be able to

1. critically assess and creatively apply technological possibilities and innovations in the fields of data science, software development and programming languages;
2. critically assess and apply software engineering methodologies considering real life situations, organizations and industries;
3. use, adapt and improve modern techniques in data science, such as deep learning, recommender systems, computer vision, and machine learning in software engineering;
4. apply cross-disciplinary management methodologies to solve academic and professional problems in the context of software development and data science;
5. critically assess and integrate a consistent tool set of leadership abilities into a professional work environment;
6. plan, conduct and document small research projects in the context of data science, software development and programming languages;
7. independently research, document and present a scientific topic with appropriate language skills;

8. use scientific methods as appropriate in the field of data science and software engineering such as defining research questions, justifying methods, collecting, assessing and interpreting relevant information, and drawing scientifically-founded conclusions that consider social, scientific and ethical insights;
9. develop and advance solutions to problems and arguments in their subject area and defend these in discussions with specialists and non-specialists;
10. engage ethically with academic, professional and wider communities and to actively contribute to a sustainable future, reflecting and respecting different views;
11. take responsibility for their own learning, personal and professional development and role in society, evaluating critical feedback and self-analysis;
12. apply their knowledge and understanding of data science, software development, and programming languages to a professional context;
13. take on responsibility in a diverse team;
14. adhere to and defend ethical, scientific and professional standards;
15. apply data analytics techniques;
16. understand and utilize agile product development and design methodologies;
17. understand and apply principles of quality engineering.

1.3 Target Audience

The MSc Advanced Software Technology Program at Constructor University is designed for students of diverse backgrounds, with a focus on those who have completed an undergraduate program in Computer Science or a related field. The program is tailored for graduates who are interested in gaining advanced knowledge and skills in the fields of Data Science, Software Development and Programming Languages.

This program is particularly suitable for candidates who are dedicated to and interested in gaining theoretical and application-oriented knowledge in the fields of Data Science, Machine Learning, Software Engineering, Cybersecurity, Artificial Intelligence, and Programming Languages.

The program prepares students for key roles in the IT industry, as well as for entering research in the subject fields. Additionally, the program provides students with additional educational opportunities in management and leadership, which can prepare them to develop their own start-up. The program's educational approach encourages exchange and discussion within the student community, making the willingness to interact, appreciate different teaching and learning formats, accept challenges and develop professionally during study, important requirements for successful participation in the program.

1.4 Career Options

The field of Advanced Software Technology is rapidly growing and in high demand as more and more companies are recognizing the value of data-driven decision making. Graduates of the MSc Advanced Software Technology program at Constructor University will be well-equipped to enter a variety of exciting and rewarding careers in the IT industry.

Graduates of this program will be well-prepared for roles in data analysis and software development, such as data scientists, software engineers, and machine learning engineers. They will also be able to work in a wide range of industries, including finance, healthcare, education, and technology. The program's focus on advanced software technology provides students with a versatile skill set that will be highly valued by employers.

Constructor University's Student Career Services and Alumni Association, as well as the university's partnerships with leading technology companies such as JetBrains, Acronis, Alemira, Virtuozzo and Rolos, will provide students with valuable support and opportunities for professional growth. The Student Career Services offers high-quality training and coaching in application and interview preparation, effective presenting, business etiquette, and employer research, while the Alumni Association helps students establish a long-lasting worldwide network. These resources, along with the university's industry connections, will help graduates succeed in their chosen careers.

1.5 Admission Requirements

The Advanced Software Technology graduate program requires students to have completed an undergraduate program in computer science, data science, software development, information technology or another discipline with at least 60 ECTS of computer science-related topics (such as mathematics, programming, design, software architecture).

Admission to Constructor University is selective and based on a candidate's university achievements, recommendations and self-presentation. Students admitted to Constructor University demonstrate exceptional academic achievements, intellectual creativity, and the desire and motivation to make a difference in the world.

The following documents need to be submitted with the application:

- Letter of motivation
- Curriculum vitae (CV)
- Official or certified copies of university transcripts
- Bachelor's degree certificate or equivalent
- Language proficiency test results (minimum score of 90 (TOEFL), 6.5 (IELTS) or 110 (Duolingo)).
- Copy of Passport
- Letter of recommendation (optional).

Formal admission requirements are subject to higher education law and are outlined in the Admission and Enrollment Policy of Constructor University.

For more detailed information about the admission visit:

https://constructor.university/admission-aid/application-information-graduate.

1.6 More information and contacts

For more information on the study program please contact the Study Program Coordinator:

Prof. Dr. Alexander Omelchenko

Professor of Applied Mathematics, Data Science and Computing

Email: aomelchenko@constructor.university

or visit our program website: Advanced Software Technology | Constructor University

For more information on Student Services please visit:
https://constructor.university/student-life/student-services

2 The Curriculum

2.1 The Curriculum at a Glance

The Advanced Software Technology graduate program is composed of foundational lectures, specialized modules, and applied project work, leading to a master thesis that can be conducted in research groups at Constructor University, at external research institutes or in close collaboration with a company. The program takes four semesters (two years). The following table shows an overview of the modular structure of the program. The program is sectioned into two areas (AST and Management modules) and the Master Thesis. All credit points (CP) are ECTS (European Credit Transfer System) credit points. In order to graduate, students need to obtain 120 CP. See Chapter 3 "Modules" of this handbook for the detailed module descriptions or refer to CampusNet.

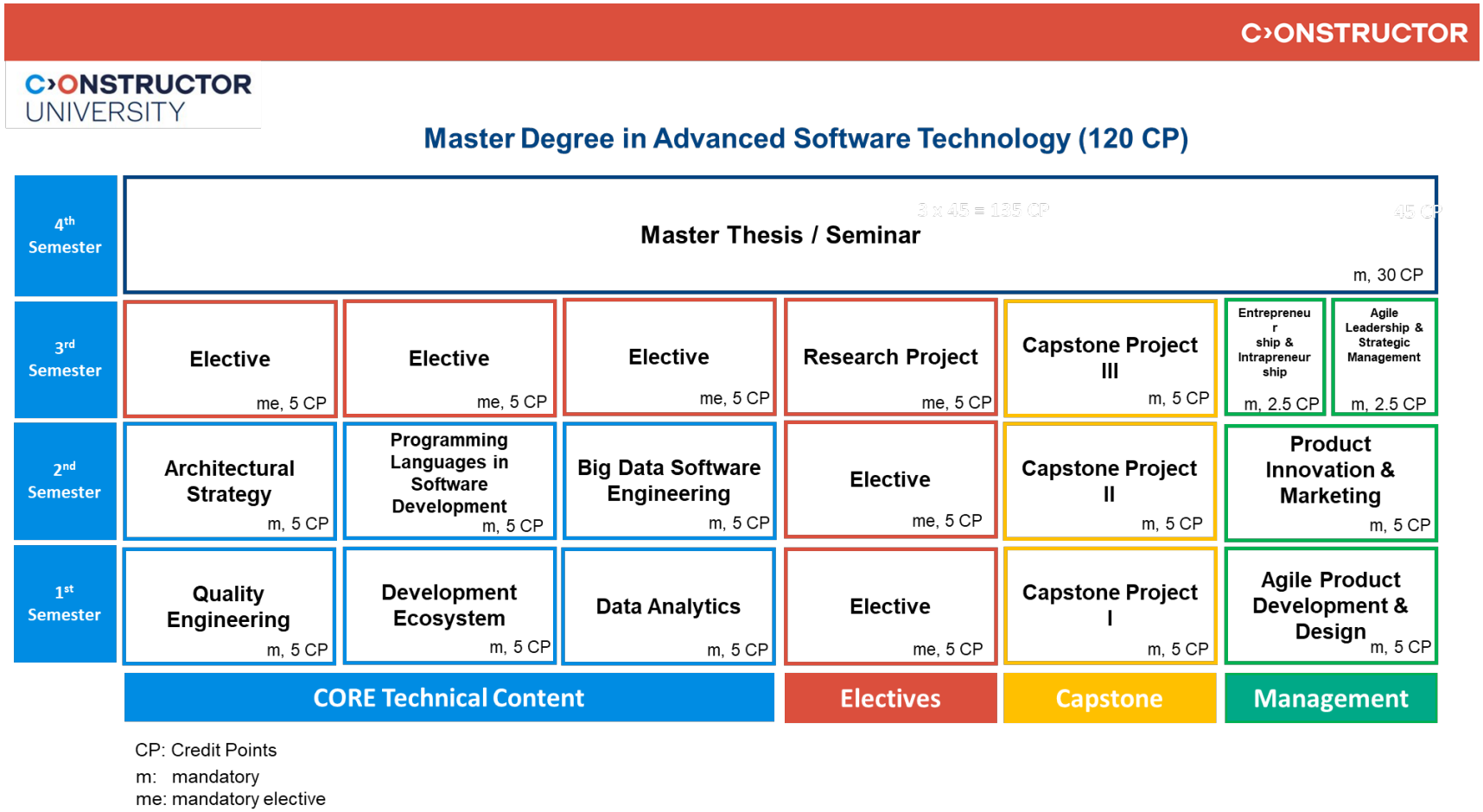

Figure 1: Schematic Study Scheme

2.2 Study and Examination Plan

MSc Degree in Advanced Software Technology Matriculation Fall 2023							
Module Code	**Program-Specific Modules**	**Type**	**Assessment**	**Period**[1]	**Status**[2]	**Semester**	**CP**
Semester 1							**30**
	Unit: CORE modules						**20**
MCSSE-SE-02	**Module: Quality Engineering**				**m**	**1**	**5**
MCSSE-SE-02	Quality Engineering	Lecture	Portfolio	During semester			
MAST-101	**Module: Development ecosystem**				**m**	**1**	**5**
MAST-101-A	Development ecosystem	Lecture	Written examination	Examination period			
MDE-CO-02	**Module: Data Analytics**				**m**	**1**	**5**
MDE-CO-02	Data Analytics	Lecture	Project report	During semester			
	Further CORE modules				**me**	**1**	**5**
	- students choose 1 module from those listed below						
	Unit: Capstone Project						**5**
MCSSE-CAP-01	**Module: Capstone Project 1**				**m**	**1**	**5**
MCSSE-CAP-01	Capstone Project 1	Project	Project Assessment	During semester			
	Unit: Management and Leadership Modules						**5**
MCSSE-MGT-01	**Module: Agile Product Development & Design**				**m**	**1**	**5**
MCSSE-MGT-01	Agile Product Development & Design	Lecture	Presentation	Examination period			
Semester 2							**30**
	Unit: CORE modules						**20**
MCSSE-SE-03	**Module: Architectural Strategy**				**m**	**2**	**5**
MCSSE-SE-03	Architectural Strategy	Lecture	Portfolio	During semester			
MAST-102	**Module: Programming Languages on Software Development**				**m**	**2**	**5**
MAST-102-A	Programming Languages on Software Development	Lecture	Written examination	Examination period			2.5
MAST-102-B	Programming Languages on Software Development Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-103	**Module: Big Data Software Engineering**				**m**	**2**	**5**
MAST-103-A	Big Data Software Engineering	Lecture	Written examination	Examination period			2.5
MAST-103-B	Big Data Software Engineering Tutorial	Tutorial	Practical assessment	During semester			2.5
	Further CORE modules				**me**	**2**	**5**
	- students choose 1 module from those listed below						
	Unit: Capstone Project						**5**
MCSSE-CAP-02	**Module: Capstone Project 2**				**m**	**2**	**5**
MCSSE-CAP-02	Capstone Project 2	Project	Project Assessment	During semester			
	Management Modules						**5**
MCSSE-MGT-02	**Module: Product Innovation & Marketing**				**m**	**2**	**5**
MCSSE-MGT-02	Product Innovation & Marketing	Lecture	Presentation	During semester			
Semester 3							**30**
	Unit: CORE modules						**20**
	Further CORE modules				**me**	**3**	**20**
	- students choose 4 modules from those listed below. One CORE module can be replaced by the Research Project module.						
	Unit: Capstone Project						**5**
MCSSE-CAP-03	**Module: Capstone Project 3**				**m**	**3**	**5**
MCSSE-CAP-03	Capstone Project 3	Project	Project Assessment	During semester			
	Unit: Management and Leadership Modules						**5**
MCSSE-LAS-03	**Module: Agile Leadership and Strategic Management**				**m**	**3**	**2,5**
MCSSE-LAS-03	Agile Leadership and Strategic Management	Lecture	Presentations	During semester			
MCSSE-LAS-01	**Module: Entrepreneurship & Intrapreneurship**				**m**	**1**	**2.5**
MCSSE-LAS-01	Entrepreneurship & Intrapreneurship	Lecture	Presentations	During semester			
Semester 4							**30**
	Master Thesis						**30**
MAST-300	**Module: Master Thesis MSc DSSD**				**m**	**4**	**30**
MAST-300-T	Master Thesis DSSD	Thesis					
Total CP							**120**

[1] Each lecture period lasts 14 semester weeks and is followed by reading and examination days. Written examinations are centrally scheduled during weeks 15 and 16. For all other assessment types, the timeframes indicated in the above table stipulate the period during which module work has to be handed in or presented. Specific information on dates of topic announcement as well as submission deadlines is communicated in the syllabus which is made available to the students at the beginning of each semester. Academic dates are published in the university-wide Academic Calendar (see http://www.jacobs-university.de/academic-calendar).

[2] m = mandatory, me = mandatory elective

Further CORE modules							
Module Code	**Program-Specific Modules**	**Type**	**Assessment**	**Period[1]**	**Status[2]**	**Semester**	**CP**
Data Science Track							
MAST-105	**Module: Advanced Deep Learning**				**me**	**1**	**5**
MAST-105-A	Advanced Deep Learning	Lecture	Written examination	Examination Period			2.5
MAST-105-B	Advanced Deep Learning Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-202	**Module: Recommender Systems**				**me**	**3**	**5**
MAST-202-A	Recommender Systems	Lecture	Written examination	Examination Period			2.5
MAST-202-B	Recommender Systems Tutorial	Tutorial	Practical assessment	During semester			2.5
NEW	**Module: Computer Vision**				**me**	**3**	**5**
NEW	Computer Vision	Lecture	Written examination	Examination Period			
MAST-203	**Module: Machine Learning in Software Engineering**				**me**	**3**	**5**
MAST-203-A	Machine Learning in Software Engineering	Lecture	Written examination	Examination Period			2.5
MAST-203-B	Machine Learning in Software Engineering Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-204	**Module: Bayesian Methods in Machine Learning**				**me**	**1**	**5**
MAST-204-A	Bayesian Methods in Machine Learning	Lecture	Written examination	Examination Period			2.5
MAST-204-B	Bayesian Methods in Machine Learning Tutorial	Tutorial	Practical assessment	During semester			2.5
Software Development Track							
MAST-205	**Module: Static Program Analysis**				**me**	**1**	**5**
MAST-205-A	Static program Analysis	Lecture	Written examination	Examination Period			2.5
MAST-205-B	Static program Analysis Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-108	**Module: Mobile Application Development**				**me**	**1 or 3**	**5**
MAST-108-A	Mobile Application Development	Lecture	Written examination	Examination Period			2.5
MAST-108-B	Mobile Application Development Tutorial	Tutorial	Practical assessment	During semester			2.5
MCSSE-CYB-01	**Module: Cryptography**				**me**	**1**	**5**
MCSSE-CYB-01	Cryptography	Lecture	Written examination	Examination Period			
MCSSE-CYB-02	**Module: System Security**				**me**	**2**	**5**
MCSSE-CYB-02	System Security	Lecture	Written examination	Examination Period			
MAST-206	**Module: Distributed Ledger Technology and Smart Contracts**				**me**	**3**	**5**
MAST-206-A	Distributed Ledger Technology and Smart Contracts	Lecture	Written examination	Examination Period			2.5
MAST-206-B	Distributed Ledger Technology and Smart Contracts Tutorial	Tutorial	Practical assessment	During semester			2.5
MCSSE-CYB-03	**Module: Network Security**				**me**	**3**	**5**
MCSSE-CYB-03	Network Security	Lecture	Written examination	Examination Period			
CA-S-RIS-802	**Module: Human-Computer Interaction**				**me**	**1 or 3**	**5**
CA-S-RIS-802	Human Computer Interaction	Lecture	Written examination	Examination Period			
MAST-207	**Module: IDE Development**				**me**	**1**	**5**
MAST-207-A	IDE Development	Lecture	Written examination	Examination Period			2.5
MAST-207-B	IDE Development Tutorial	Tutorial	Practical assessment	During semester			2.5
Programming Languages Track							
MAST-104	**Module: Advanced Functional Programming**				**me**	**1**	**5**
MAST-104-A	Advanced Functional Programming	Lecture	Written examination	Examination Period			2.5
MAST-104-B	Advanced Functional Programming Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-208	**Module: Weak memory Models**				**me**	**1**	**5**
MAST-208-A	Weak memory Models	Lecture	Written examination	Examination Period			2.5
MAST-208-B	Weak memory Models Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-106	**Module: Virtual Machines**				**me**	**2**	**5**
MAST-106-A	Virtual Machines	Lecture	Written examination	Examination Period			2.5
MAST-106-B	Virtual Machines Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-107	**Module: Metacomputations**				**me**	**2**	**5**
MAST-107-A	Metacomputations	Lecture	Written examination	Examination Period			2.5
MAST-107-B	Metacomputations Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-209	**Module: Dependent Types**				**me**	**3**	**5**
MAST-209-A	Dependent Types	Lecture	Written examination	Examination Period			2.5
MAST-209-B	Dependent Types Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-210	**Module: Homotopy Type Theory**				**me**	**3**	**5**
MAST-210-A	Homotopy Type Theory	Lecture	Written examination	Examination Period			2.5
MAST-210-B	Homotopy Type Theory Tutorial	Tutorial	Practical assessment	During semester			2.5
MAST-211	**Module: Category Theory for Programmers**				**me**	**3**	**5**
MAST-211-A	Category Theory for Programmers	Lecture	Written examination	Examination Period			2.5
MAST-211-B	Category Theory for Programmers Tutorial	Tutorial	Practical assessment	During semester			2.5
	Research Project						**5**
MCSSE-RP-01	**Module: Research Project**				**me**	**3**	**5**
MCSSE-RP-01	Research Project	Project	Project Report	Examination period			

Figure 2: Study and Examination Plan

2.3 Core Area (45 CP)

This area is the centerpiece of the Advanced Software Technology program. The six mandatory modules in the Core Area cover essential methods of Advanced Software Technology. They provide the foundations for further, more advanced modules and applied projects by introducing the fundamental concepts, methods and technologies used in Advanced Software Technology. The modules are intensive courses accompanied by hands-on tutorials and labs.

To pursue an AST master, the following CORE modules (45 CP) need to be taken as mandatory modules (m):

- CORE Module: Quality Engineering (m, 5 CP)
- CORE Module: Development Ecosystem (m, 5 CP)
- CORE Module: Data Analytics (m, 5 CP)
- CORE Module: Architectural Strategy (m, 5 CP)
- CORE Module: Programming Languages in Software Development (m, 5 CP)
- CORE Module: Big Data Software Engineering (m, 5 CP)

2.4 Elective Area (30 CP)

The Advanced Software Technology program attracts students with diverse career goals, backgrounds, and prior work experience. Therefore, modules in this area can be chosen freely by students depending on their prior knowledge and interests. Students can choose to strengthen their knowledge by following one of suggested focus tracks and electing the modules offered therein: Data Science, Software Development, and Programming Languages.

Students may choose any combination of the modules listed below. Each track may be followed completely and/or complemented with other modules). In addition to the modules offered within these focus tracks, 3rd year modules from the undergraduate curriculum or other graduate programs at Constructor University can be taken with the approval of the program coordinator. Please see CampusNet for current offerings.

To pursue an AST master, students choose the following Electives modules (30 CP) as mandatory elective modules (me):

Data Science Track:

- Electives Module: Advanced Deep Learning (me, 5 CP)
- Electives Module: Recommender Systems (me, 5 CP)
- Electives Module: Machine Learning in Software Engineering (me, 5 CP)
- Electives Module: Bayesian Methods in Machine Learning (me, 5 CP)

Software Development Track:

- Electives Module: Static Program Analysis (me, 5 CP)
- Electives Module: Mobile Application Development (me, 5 CP)
- Electives Module: Cryptography (me, 5 CP)
- Electives Module: System Security (me, 5 CP)

- Electives Module: Distributed Ledger Technology and Smart Contracts (me, 5 CP)
- Electives Module: Network Security (me, 5 CP)
- Electives Module: IDE Development (me, 5 CP)

Programming Language Track:

- Electives Module: Advanced Functional Programming (me, 5 CP)
- Electives Module: Weak Memory Models (me, 5 CP)
- Electives Module: Virtual Machines (me, 5 CP)
- Electives Module: Metacomputations (me, 5 CP)
- Electives Module: Dependent Types (me, 5 CP)
- Electives Module: Homotopy Type Theory (me, 5 CP)Electives Module: Category Theory for Programmers (me, 5 CP)

2.5 Management Area (15 CP)

To equip students with market-relevant management skills they take modules in the fields of product development, marketing and change management. All modules are mandatory for the program.

To pursue an AST master, the following Management modules (15 CP) need to be taken as mandatory modules (m):

- Management Module: Agile Product Development & Design (m, 5 CP)
- Management Module: Product Innovation & Marketing (m, 5 CP)
- Management Module: Entrepreneurship & Intrapreneurship (m, 2.5 CP)
- Management Module: Agile Leadership and Strategic Management (m, 2.5 CP)

2.6 Capstone project, Research project and Master Thesis (45 CP)

To explore the full development process of a software application with relation to the areas of specialization of the program, all students take the three modules of the Capstone Project. It is highly recommended to take the three modules in their numerical order, to gain full experience of the project. Students with a strong drive towards academic research can replace in their third semester one Technical CORE Module by the Research Project, which is carried out in cooperation with JetBrains. The JetBrains researcher will provide research topics for the students. In the fourth semester, students conduct research and write a master thesis guided and supported by their academic advisor.

To pursue an AST master, the following modules (15 CP) need to be taken as mandatory modules (m):

- Capstone Module: Capstone Project 1 (m, 5 CP)
- Capstone Module: Capstone Project 2 (m, 5 CP)
- Capstone Module: Capstone Project 3 (m, 5 CP)

Students can replace in their third semester one Technical CORE Module by the Research Project (5 CP):

- Research Project Module: Research Project (me, 5 CP)

To pursue an AST master, the following Master Thesis module need to be taken as mandatory module:

- Thesis Module: Master Thesis (m, 30 CP)

3 Advanced Software Technology Modules

3.1 Quality Engineering

<table>
<tr><td colspan="3">Module Name
Quality Engineering</td><td>Module Code
MCSSE-SE-02</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP</td></tr>
<tr><td>MCSSE-SE-02</td><td colspan="3">Quality Engineering</td><td>Lecture / Tutorial</td><td>5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Alexander Omelchenko</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills:
• Programming skills in an imperative language at CS bachelor level
• Algorithms and data structure at CS bachelor level
• Basic skills in software testing: structural testing, Junit
• Basic knowledge of software engineering and IDEs at CS bachelor level
• Discrete math at CS bachelor level</td><td>Frequency

Annually (Fall)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Tutorial (35 hours)
• Private study (55 hours)

Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation</td></tr>
<tr><td colspan="6">Content and Educational Aims

Software quality can be defined as the degree of satisfaction of the requirements; it represents an essential part of the software development and cannot be guaranteed a-priori, but most be verified both during and after the development. This course introduces the main testing and analysis techniques that can be used to identify failures and verify the quality of software systems. The course introduces the general testing and analysis principles and the basic techniques, shows how to apply them to solve relevant quality problems, illustrates complementarities and differences among the different techniques, and presents the organization of a coherent quality process. The course provides the elements needed to understand principles, techniques and process that comprise the basic background of test designer, quality manager and project manager. At the end of the course, the students will be able to define and implement quality plans for complex software systems. The student will have the basic knowledge of a project and a quality manager.

Students will know in the first session which assignments will be part of the portfolio examination.</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. manage a software quality process.
2. select and implement a suitable set of testing and analysis activities to certify the quality of software systems.
3. understand the core principles of software testing and program analysis.
4. master the basic techniques underlying software testing and program analysis.
5. choose the suitable approaches to address the different testing and analysis programs.
6. design and monitor a suitable quality process.

Indicative Literature

Usability and Relationship to other Modules

Examination Type: Module Examination

Assessment: Portfolio (Individual Assignments, Group Assignments) Weight: 100 %

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.2 Development Ecosystem

<table>
<tr><td colspan="3">Module Name

Development Ecosystem</td><td>Module Code

MAST-101</td><td>Level (type)

Year 1</td><td>CP

5.0</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP</td></tr>
<tr><td>MAST-101-A</td><td colspan="3">Development Ecosystem</td><td>Lecture</td><td>5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Timofey Bryksin</td><td colspan="3">Program Affiliation

● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status

Mandatory for AST</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ expected to have practical knowledge of everything described in Kotlin documentation (https://kotlinlang.org/docs/) up to Annotations</td><td>Frequency

Annually

(Fall)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

● Lecture and Tutorials (35 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)

Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Before diving into the ecosystem, it's important to have a solid understanding of the Kotlin language itself. You can start by reading through the official Kotlin documentation and working through some of the tutorials and examples provided there.</td></tr>
<tr><td colspan="6">Content and Educational Aims

A programming language is only the first tool you need to develop applications. After knowing the syntax and the execution environment come tooling and essential libraries. In the end, to develop a non-trivial and practical application one should know a lot about the programming language ecosystem. This course covers some software development practices in Kotlin and some must-know libraries and tools for Kotlin.

Content:
• Gradle
• Testing
• Profiling
• DSL
• Networking in JVM
• Ktor
• Reflection
• Data Science
• Interoperability
• Annotations</td></tr>
</table>

- IntelliJ Platform SDK
- Compose

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. write different Kotlin applications from scratch
2. use Kotlin for web-development, data science, IntelliJ Platform plugins
3. deploy and maintain Kotlin applications in production environments
4. understand deeply how the Kotlin compiler works and how Kotlin works with different platforms

Indicative Literature

Dmitry Jemerov and Svetlana Isakova: "Kotlin in Action", Manning Publications, 2017.

Antonio Leiva: "Kotlin for Android Developers", Packt Publishing, 2017.

Stephen Samuel and Stefan Bocutiu: "Programming Kotlin", O'Reilly Media, 2018.

Ashish Belagali and Hardik Trivedi: "Kotlin Blueprints", Packt Publishing, 2018.

Alexey Soshin: "Kotlin Cookbook", O'Reilly Media, 2018.

Usability and Relationship to other Modules

- Kotlin is a general-purpose programming language that is designed to be fully interoperable with Java. This means that it can be used in a wide variety of contexts, including web development, Android development, and server-side development. One of the main advantages of Kotlin is its improved readability and expressiveness over Java. It has a more compact and expressive syntax, which makes it easier to write and maintain code. Additionally, Kotlin has a number of features that make it more suitable for functional programming, such as support for lambda expressions and higher-order functions. Another advantage of Kotlin is that it is fully compatible with Java, which means that developers can easily integrate it into existing Java projects, and use Java libraries and frameworks with Kotlin. This also makes it easy for Java developers to start using Kotlin, as they can continue to use the tools and libraries that they are already familiar with. For Android development, Kotlin has become the preferred language for Android development by Google since 2019, and it is supported by Android Studio, the official IDE for Android development. This make the transition from Java to Kotlin very smooth.

Examination Type: Module Examination

Assessment: Written examination — Duration: 120 min
Weight: 100%

Scope: All intended learning outcomes of the module

Completion: To pass this module, the examination has to be passed with at least 45%.

3.3 Data Analytics

Module Name	**Module Code**	**Level (type)**	**CP**
Data Analytics	MDE-CO-02	Year 1 (CORE)	5

Module Components

Number	Name	Type	CP
MDE-CO-02	Data Analytics	Lecture	5

Module Coordinator	**Program Affiliation**	**Mandatory Status**
Prof. Dr. Adalbert F.X. Wilhelm	▪ MSc Data Engineering (DE)	Mandatory for AST and DE Mandatory elective for DSSB and CSSE

Entry Requirements			**Frequency**	**Forms of Learning and Teaching**
Pre-requisites	Co-requisites	Knowledge, Abilities, or Skills	Annually (Fall)	▪ Lecture (17.5 hours) ▪ Tutorials (17.5 hours) ▪ Private study (90 hours)
☒ None	☒ None	☒ None	**Duration**	**Workload**
			1 semester	125 hours

Recommendations for Preparation

Read the Syllabus.
Take the free online course: Introduction to Data Science at https://cognitiveclass.ai/courses/data-science-101/

Content and Educational Aims

This module introduces concepts and methods of data analytics. The objective of the module is to present methods for gaining insight from data and drawing conclusions for analytical reasoning and decision-making. The module comprises a broad spectrum of methods for modelling and understanding complex datasets. Comprising both descriptive and predictive analytics, the standard portfolio of supervised and unsupervised learning techniques is introduced. Automatic analysis components, such as data transformation, aggregation, classification, clustering, and outlier detection, will be treated as an integral part of the analytics process.
As a central part of this module, students are introduced to the major concepts of statistical learning such as cross-validation, feature selection, and model evaluation. The course takes an applied approach and combines the theoretical foundation of data analytics with a practical exposure to the data analysis process.

Intended Learning Outcomes

By the end of this module, students will be able to

1. explain advanced data analytics techniques in theory and application;
2. apply data analytics methods to real-life problems using appropriate tools;
3. evaluate and compare different data analytics algorithms and approaches;
4. apply statistical concepts to evaluate data analytics results.

Indicative Literature

G. James, D.Witten, T. Hastie, Rob Tibshirani: Introduction to Statistical Learning with R by Springer, 2013 (ISLR)
A. Telea, Data Visualization: Principles and Practice, Wellesley, Mass.: AK Peters, 1st edition, 2008.(DV)
M. Ward, G. Grinstein, D. Keim, Interactive Data Visualization: Foundations, Techniques, and Applications. AK Peters, 1st edition, 2010. (IDV)

Usability and Relationship to other Modules

In this module students will learn concepts and various techniques for data analysis. They will be rigorously applied in MDE-CS-03 as well as in the applied projects MDE-DIS-02 and MDE-DIS-03, and typically also in the master thesis.

Examination Type: Module Examination

Assessment Type: Project Report — Length: 20 pages
Weight: 100%

Scope: All intended learning outcomes of this module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.4 Architectural Strategy

<table>
<tr><td colspan="2">Module Name
Architectural Strategy</td><td>Module Code
MCSSE-SE-03</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="5">Module Components

Number | Name | Type | CP
MCSSE-SE-03 | Architectural Strategy | Lecture / Tutorial | 5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Alexander Omelchenko</td><td colspan="2">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="2">Entry Requirements

Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills</td><td>Frequency

Annually (Spring)</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Tutorial (35 hours)
• Private study (55 hours)</td></tr>
<tr><td colspan="2"></td><td>Duration</td><td colspan="2">Workload

125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation</td></tr>
<tr><td colspan="5">Content and Educational Aims

The course “Architectural Strategy” focuses on Software Architectures, the key element for systematically developing large and complex software systems. During the course, we study how to design, recover, analyze, and document Software Architectures and understand how the main design decisions comprising them influence the quality attributes of the resulting systems.

Students will know in the first session which assignments will be part of the portfolio examination.</td></tr>
<tr><td colspan="5">Intended Learning Outcomes

Upon completion of this module, students will be able to
1. understand methods for designing large software systems
2. design complex and large software systems using components and connectors
3. use UML as modeling language to represent the main concepts of software systems
4. document their main design decisions and motivate them in terms of quality attributes</td></tr>
<tr><td colspan="5">Indicative Literature

R.N. Taylor, N. Medvidovic, E.M. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley, January (2009)

Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. Addison Wesley 2013

C. Pautasso, Software Architecture, 2020 (Visual Lecture Notes)</td></tr>
<tr><td colspan="5">Usability and Relationship to other Modules</td></tr>
<tr><td colspan="5">Examination Type: Module Examination</td></tr>
</table>

Assessment: Portfolio (Individual Assignments, Group Assignments)	Weight: 100 %
Scope: All intended learning outcomes of the module.	
Completion: To pass this module, the examination has to be passed with at least 45%.	

3.5 Programming Languages in Software Development

<table>
<tr><td colspan="3">Module Name
Programming Languages in Software Development</td><td>Module Code
MAST-102</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-102-A</td><td colspan="3">Programming Languages in Software Development</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-102-B</td><td colspan="3">Programming Languages in Software Development - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Timofey Bryksin</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Spring)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td colspan="3"></td><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
Before taking the course, it's important to have a solid understanding of at least one programming language, as the course will cover a wide range of languages and paradigms.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module aims to provide students comprehensive understanding of the different types of programming languages and their characteristics, to familiarize them with the syntax and semantics of a variety of programming languages, including low-level, high-level, functional, logic, concurrent, parallel, scripting, and domain-specific languages, to teach students how to analyze and compare different programming languages, and to understand the trade-offs between different language features, to train students to use different programming languages for different types of software development tasks, such as web development, data science, and mobile app development, to develop students' problem-solving skills by applying the programming languages to solve real-world problems.
.
Content:
● Overview of programming languages: history, classification, and trends.
● Low-level languages: assembly, machine code, and C.
● High-level languages: Java, C#, Python, JavaScript, and Kotlin.
● Functional languages: Haskell, Lisp, and Scala.
● Logic and constraint programming languages: Prolog, and MiniZinc.
● Concurrent and parallel programming languages: Erlang, and Go.
● Scripting languages: Perl, Ruby, and Shell.
● Domain-specific languages: SQL, and XML.</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. understand the history and classification of programming languages, and be able to analyze and compare different languages based on their characteristics.
2. write, read, and understand code written in a variety of programming languages, including low-level, high-level, functional, logic, concurrent, parallel, scripting, and domain-specific languages.
3. use different programming languages for different types of software development tasks, such as web development, data science, and mobile app development.
4. evaluate the trade-offs between different language features and choose the appropriate language for a given task.
5. apply their knowledge of programming languages to solve real-world problems, and develop their problem-solving skills.

Indicative Literature

Terrence W. Pratt and Marvin V. Zelkowitz: "Programming Languages: Design and Implementation", Prentice Hall, 2004.

Michael L. Scott: "Programming Language Pragmatics", Morgan Kaufman Publishers, 2009.

Carl A. Gunter: "Introduction to the Theory of Programming Languages", Cambridge University Press, 1996.

Robert W. Sebesta: "Concepts of Programming Languages", Addison-Wesley, 2010.

David A. Watt and Deryck F. Brown: "Programming Languages and Paradigms", Pearson, 2008.

Usability and Relationship to other Modules

- The course content is designed to provide students with a comprehensive understanding of different programming languages, including low-level, high-level, functional, logic, concurrent, parallel, scripting, and domain-specific languages. It covers the history, classification and trends of programming languages. This would give students the ability to analyze and compare different languages based on their characteristics, and choose the appropriate language for a given task.

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight:50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.6 Big Data Software Engineering

<table>
<tr><td colspan="3">Module Name
Big Data Software Engineering</td><td>Module Code
MAST-103</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-103-A</td><td colspan="3">Big Data Software Engineering</td><td>Lectures</td><td>2.5</td></tr>
<tr><td>MAST-103-B</td><td colspan="3">Big Data Software Engineering - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Timofey Bryksin</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory for AST</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ Basic knowledge Kotlin or Java</td><td>Frequency
Annually
(Spring)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (35 hours)
● Independent study (52.5 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
Before taking the course, it's important to have a solid understanding of the concepts and techniques of software engineering and data science, as the course will cover big data technologies and how to use them for data analysis and modeling. Minimal knowledge of Docker, PostgreSQL and basics of working with relational databases will be a big plus.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module aims to provide students with the principles of building a scalable distributed software system for storing and processing big amounts of data. The course will look at the production solutions where such principles are implemented and will try to write our own distributed key-value storage.

Content:
• Data partitioning/sharding
• Data replication
• Distributed data processing
• Consistency in distributed systems

Assignments assume writing code, tests, configuration files, doing peer code reviews, deploying code in a cloud environment and running benchmarks.</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. understand the general principles and challenges of building a distributed data storage system
2. implement data partitioning, replication and consensus algorithms in their own systems
3. use data partitioning, replication and consensus features of the existing database systems

Indicative Literature

"Big Data: A Revolution That Will Transform How We Live, Work, and Think" by Viktor Mayer-Schönberger and Kenneth Cukier, Houghton Mifflin Harcourt, 2013.

"Data Management for Researchers: Organize, Maintain and Share Your Data for Research Success" by Kristin Briney, CreateSpace Independent Publishing Platform, 2017.

"Big Data: Understanding How Data Powers Big Business" by Bernard Marr, John Wiley & Sons, 2015.

"Real-Time Big Data Analytics: Emerging Architecture" by Tejaswini Mandar Jog, Apress, 2016.

"Big Data Analytics with R and Hadoop" by Vignesh Prajapati, Packt Publishing, 2016.

Usability and Relationship to other Modules

- The module provides a comprehensive coverage of the tools and technologies used for storing, managing, and processing big data. It also covers the important topic of data quality, governance and security. The course is suitable for students who want to learn about the challenges and opportunities of big data and how to use the technologies to process and analyze big data. The course is also beneficial for students who want to pursue a career in data science, software engineering, or big data analytics.

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.7 Static Program Analysis

<table>
<tr><td colspan="3">Module Name
Static Program Analysis</td><td>Module Code
MAST-205</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-205-A</td><td colspan="3">Static Program Analysis</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-205-B</td><td colspan="3">Static Program Analysis Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="3">Program Affiliation
● MSc Data Science and Software Engineering (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td colspan="3"></td><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
It is important to have a solid understanding of the concepts and techniques of software engineering and programming languages, as the course will cover program analysis techniques and how to use them to improve software quality and security. Understanding compilers, formal languages or semantics of programming languages would make parts of the course easier to grasp, but it is not a hard pre-requisite.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module aims to provide students with a comprehensive understanding of the kinds of program analysis and their applications; to familiarize students with the techniques and algorithms used for type analysis, data- and control-flow analyses, intra- and interprocedural analyses, alias analysis, bounded model checking; to develop students' skills in using program analysis to detect bugs, optimize code and perform security analysis; to train students to use program analysis tools and frameworks such as Soot, LLVM, and Frama-C; to give students an opportunity to apply their knowledge of program analysis to solve real-world problems.

Content:
• Introduction to program analysis: Types of program analysis, applications, and challenges.
• Type analysis: Definition, kinds and algorithms.
• Monotone framework: Definition, kinds and algorithms.
• Interval analysis: Definition, kinds and algorithms.
• Path sensitive analysis: Definition, kinds and algorithms.
• Bounded model checking: Definition, kinds and algorithms.
• Interprocedural analysis: Definition, kinds and algorithms.
• Alias analysis: Definition, kinds and algorithms.
• Applications of program analysis: Bug detection, code optimization, and security analysis.</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. understand the different kinds of program analysis, their applications, and challenges;
2. to design and implement program analysis algorithms for type analysis, data- and control-flow analyses, intra- and interprocedural analyses, alias analysis, bounded model checking;
3. use program analysis tools and frameworks such as Soot, LLVM, and Frama-C;
4. understand the results of program analyses, and use them to improve software quality and security;
5. apply program analysis techniques to solve real-world problems in the field of software engineering.

Indicative Literature

"Principles of Program Analysis" by Hanne Riis Nielson, Flemming Nielson, Springer, 1999.

“Introduction to Static Analysis: An Abstract Interpretation Perspective” by Xavier Rival, Kwangkeun Yi, The MIT Press, 2020

“Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities” by Axel Simon, Springer, 2008

“Introduction to Lattices and Order” by B.A. Davey, H.A. Priestley, Cambridge University Press, 2022

“WYSINWYX: What You See Is Not What You Execute” by Gogul Balakrishnan, University Of Wisconsin–Madison, 2007

Usability and Relationship to other Modules

- This module belongs to the Software Engineering Track in the MSc AST
- The course provides a comprehensive coverage of different types of program analysis, their applications and challenges. The course is suitable for students who want to learn about the different types of program analysis and how to use them to improve software quality and security. The course is also beneficial for students who want to pursue a career in software engineering, software testing or software security.

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.8 Mobile Application Development

<table>
<tr><td colspan="3">Module Name
Mobile Application Development</td><td>Module Code
MAST-108</td><td>Level (type)
Year 1 and 2</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-108-A</td><td colspan="3">Mobile Application Development</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-108-B</td><td colspan="3">Mobile Application Development - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Kirill Krinkin</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td colspan="3"></td><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
To master the module students need the basic knowledge in the field of Software Development and Programming languages, e.g., Kotlin or Java.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module aims to provide students with theoretical knowledge and practical skills in the fundamentals of mobile development. As part of the module, students will gain an understanding of the main stages of the application life cycle (including publishing and promotion). During practical classes, students will go through all stages of development from UI to functionality.
Content:
● Introduction to mobile development
● UI design principles
● Architecture and development tools
● The life cycle of mobile application</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will:
1. Know the major advances in mobile development. Be able analyze their own and others' computer code to develop applications for mobile devices. Use different forms of feedback to analyze problems and improve the performance of the mobile applications being created.
2. Know basic frameworks for software development for mobile operating systems. Select effective tools for solving practical software development problems. Select optimal libraries and algorithms for writing effective code.</td></tr>
</table>

3. Know the basics of mobile software development and operation. Audit the security of mobile devices. Have the ability to control inter-process and network interactions of mobile applications.

Indicative Literature

"Android Programming: The Big Nerd Ranch Guide" by Bill Phillips and Brian Hardy, Big Nerd Ranch Guides, 2016.

"iOS Programming: The Big Nerd Ranch Guide" by Christian Keur and Aaron Hillegass, Big Nerd Ranch Guides, 2016.

"Cross-Platform Mobile Development in C#" by Jonathan Peppers, Apress, 2014

"Mobile Application Development: Building Applications for the iPhone and Android" by John W. Carter, Addison-Wesley Professional, 2012

"Mobile Application Development: A Complete Guide" by Ahmed K. Elmagarmid, Moustafa Youssef and Mohamed M. Eltoweissy, CRC Press, 2019

Usability and Relationship to other Modules

- This module belongs to the Software Engineering Track in the MSc AST
- The course provides a comprehensive coverage of the tools and technologies used for developing mobile apps. This can include topics such as mobile app design, user interface, different mobile operating systems, software development kits, mobile security, and app deployment and distribution. The course is suitable for students who want to learn about the challenges and opportunities of mobile app development and how to use the technologies to develop mobile apps. The course is also beneficial for students who want to pursue a career in mobile app development, software engineering or mobile development.

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.9 Cryptography

<table>
<tr><td colspan="2">Module Name

Cryptography</td><td>Module Code

MCSSE-CYB-01</td><td>Level (type)

Year 1</td><td>CP

5</td></tr>
<tr><td colspan="5">Module Components

Number | Name | Type | CP
MCSSE-CYB-01 | Cryptography | Lecture | 5</td></tr>
<tr><td>Module Coordinator

Prof. Dr.
Jürgen Schönwälder</td><td colspan="2">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory elective for AST and CSSE</td></tr>
<tr><td colspan="2">Entry Requirements

Pre-requisites

☒ none

Co-requisite
☒ none

Knowledge, Abilities, or Skills</td><td>Frequency

Annually (Fall)

Duration
1 semester</td><td colspan="2">Forms of Learning and Teaching
• Lectures (35 hours)
• Private study (70 hours)
• Exam preparation (20 hours)

Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation

Students are expected to have a solid mathematical foundation. Students should review basic concepts of number theory, probability theory, and complexity theory as preparation for this module.</td></tr>
<tr><td colspan="5">Content and Educational Aims

Information security requires techniques to protect information and to secure communication. Cryptography studies the design of cryptographic algorithms that can ensure the confidentiality, the integrity, and the authenticity of data and messages exchanged in a secure communication protocol. This module focuses on the mathematical and algorithmic foundations of cryptography, and it covers the application of basic primitives to solve common information security challenges. Students familiar with the foundations of cryptographic algorithms will be able to judge the applicability and limitations of different cryptographic algorithms.</td></tr>
<tr><td colspan="5">Intended Learning Outcomes

Upon completion of this module, students will be able to

1. understand the mathematical problems on which cryptographic algorithms are built
2. describe pseudo random number generators and pseudo random functions
3. evaluate the strengths, weaknesses, and the applicability of cryptographic algorithms
4. select from a set of symmetric block cipher, message integrity, and authenticated encryption algorithms
5. contrast different asymmetric ciphers (finite field based, elliptic curve based, lattice based, hash based)
6. explain the notion of quantum resistant cryptographic algorithms
7. analyze the properties of cryptographic protocols such as key exchange mechanisms
8. apply techniques to analyze cryptographic protocols and their implementations
9. explain homomorphic encryption schemes and differential privacy</td></tr>
<tr><td colspan="5">Indicative Literature

• Bruce Schneier: Applied Cryptography, 20th Anniversary Edition, Wiley, 2015
• Wm.Arthur Conklin, Gregory White: Principles of Computer Security, 5th Edition, McGraw-Hill, 2018
• Simon Singh: The Code Book: Science of Secrecy from Ancient Egypt to Quantum Cryptography, Anchor Books, 2000
• Dan Boneh, Victor Shoup: A Graduate Course in Applied Cryptography, version 0.5, online, 2020</td></tr>
<tr><td colspan="5">Usability and Relationship to other Modules</td></tr>
</table>

- The module serves as the foundational module in the cyber security specialization in MSc CSSE. Other modules related to cyber security build on this module.
- This module belongs to the Software Engineering Track in the MSc AST

Examination Type: Module Examination

Assessment: Written examination

Duration: 120 min
Weight: 100%

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.10 System Security

<table>
<tr><td colspan="3">Module Name

System Security</td><td>Module Code

MCSSE-CYB-02</td><td>Level (type)

Year 1</td><td>CP

5</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MCSSE-CYB-02</td><td colspan="3">System Security</td><td>Lecture</td><td>5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Jürgen Schönwälder</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory elective for AST and CSSE</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements

Pre-requisites

☒ Cryptography

Co-requisites

☒ none

Knowledge, Abilities, or Skills</td><td>Frequency

Annually

(Spring)</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Private study (70 hours)
• Exam preparation (20 hours)</td></tr>
<tr><td>Duration

1 semester</td><td colspan="2">Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Students are expected to be familiar with how programs are executed at the system and machine level. Students should have a good understanding of computer architecture and operating systems at the level of typical undergraduate modules covering these topics. Students who have not taken an undergraduate course on computer architecture or operating systems yet may consider taking a remedial course or an online course to obtain a fundamental understanding how computer systems function.</td></tr>
<tr><td colspan="6">Content and Educational Aims
This module focuses on system level security aspects of computing systems. The module starts with investigating attacks on the microarchitecture of computing systems, such as attacks to gain information from side channels targeting caches. It then introduces trusted execution environments that use hardware isolation mechanisms to provide protected storage for keys and to bootstrap the integrity of bootloaders and the loaded operating systems. Students learn about the different levels of isolation that can be achieved using various types of hypervisors or sandboxing mechanisms. Techniques that can be used to protect a system against misbehaving code and malware are introduced. Students will gain knowledge how protected data storage components can be provided at the system level and how systems can offer support for collections of (distributed) authentication mechanisms. Finally, the module will discusses how authorization mechanisms are realized in the different system software components and how they can be used to define effective security policies.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. describe microarchitectural attacks and computer components and suitable counter measures
2. illustrate trusted execution environments and how they can be used to bootstrap security
3. compare the isolation achieved by hypervisors and operating system mechanisms
4. assess application layer isolation and sandboxing mechanisms
5. explain how systems can identify misbehaving code and protection themselves against malware
6. outline how protected data storage can be implemented
7. recommend authentication methods suitable for different kinds of applications
8. compose authorization mechanisms to define effective security policies</td></tr>
</table>

Indicative Literature

- William Stallings, Lawrie Brown: Computer Security: Principles and Practice, 4th edition, Pearson, 2018
- Swarup Bhunia: Hardware Security: A Hands-on Learning Approach, Morgan Kaufmann, 2018

Usability and Relationship to other Modules

- The module serves as a mandatory elective module in the cyber security specialization. Parts of the module require an understanding of cryptographic algorithms.
- This module belongs to the Software Engineering Track in the MSc AST

Examination Type: Module Examination

Assessment: Written examination

Duration: 120 min
Weight: 100%

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.11 Distributed Ledger Technology and Smart Contracts

<table>
<tr><td colspan="2">Module Name
Distributed Ledger Technology and Smart Contracts</td><td>Module Code
MAST-206</td><td>Level (type)
Year 2</td><td>CP
5.0</td></tr>
<tr><td colspan="5">Module Components

Number | Name | Type | CP</td></tr>
<tr><td>MAST-206-A</td><td colspan="2">Distributed Ledger Technology and Smart Contracts</td><td>Lecture/ Tutorial</td><td>5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="2">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="2">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)

Duration
1 semester</td><td colspan="2">Forms of Learning and Teaching
● Lecture/Tutorial attendance (35 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)

Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation
A solid understanding of computer science (data structures, algorithms, and networks) and mathematics is essential to understanding blockchain technology.</td></tr>
<tr><td colspan="5">Content and Educational Aims
The module aims to provide students with theoretical knowledge and practical skills in fundamental concepts of blockchain technology and distributed ledger, the cryptographic principles that underpin blockchain technology, gain knowledge of various blockchain platforms and use cases, keep updated with current trends and future developments in blockchain technology.

Content:
• Introduction to blockchain technology and distributed ledger
• Pioneering ones: how Bitcoin works
• How Cryptography can been applied to DLT
• Transactions in Bitcoin
• Scalability issues and famous triangle
• Permissioned/enterprise blockchain - is it a living animal?
• Blockchain 2.0 - smart contracts on the run
• Blockchain platforms and use cases - how to choose the right one?
• What is the next step for DLT?</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. understand the fundamental concepts of blockchain technology and distributed ledger systems, and be able to explain them to others.
2. understand the cryptographic principles that underpin blockchain technology, and be able to evaluate the security of different blockchain systems.
3. develop simple smart contracts.
4. keep updated with current trends and future developments in blockchain technology, and be able to evaluate the potential impact of these developments on different industries and sectors.
5. apply the knowledge of blockchain technology to real-world problems, and be able to evaluate the potential benefits and drawbacks of different solutions.

Indicative Literature

"Bitcoin: A Peer-to-Peer Electronic Cash System" by Satoshi Nakamoto, 2008

"Mastering Bitcoin" by Andreas Antonopoulos, 2017

"The Basics of Bitcoins and Blockchains " by Anthony Lewis, 2018

"Mastering Ethereum: Building Smart Contracts and DApps" by Andreas M. Antonopoulos and Gavin Wood, 2018

"Blockchain Fundamentals for Web 3.0" by Mary C. Lacity (Author), Steven C. Lupien, 2022

Usability and Relationship to other Modules

- This module belongs to the Software Engineering Track in the MSc AST
- Familiarity with basic computer science concepts such as data structures, algorithms and networks is fundamental for almost all advanced modules in computer science and technology. This module additionally introduces advanced concepts of blockchain technology, distributed ledger, and smart contracts that are needed in advanced programming-oriented modules in the 2nd year of the MSc program, as well as for developing decentralized applications and research purposes.

Examination Type: Module Examination

Assessment Type: Written examination Duration: 120 min
Weight: 100%

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.12 Network Security

<table>
<tr><td colspan="3">Module Name

Network Security</td><td>Module Code

MCSSE-CYB-03</td><td>Level (type)

Year 2</td><td>CP

5</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MCSSE-CYB-03</td><td colspan="3">Network Security</td><td>Lecture</td><td>5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Jürgen Schönwälder</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory elective for AST and CSSE</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements

Pre-requisites

☒ Cryptography

Co-requisites

☒ none

Knowledge, Abilities, or Skills</td><td>Frequency

Annually

(Fall)</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Private study (70 hours)
• Exam preparation (20 hours)</td></tr>
<tr><td>Duration

1 semester</td><td colspan="2">Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Students are expected to have a general understanding of computer networks, as provided by typical undergraduate modules on computer networks. Students who have not taken an undergraduate course on computer networks yet may consider taking a remedial course or an online course to obtain a fundamental understanding how computer networks function.</td></tr>
<tr><td colspan="6">Content and Educational Aims
Computer networks such as the Internet connect millions of computing systems, enable a fast exchange of information, and provide the technological basis on which large parts of the modern online economy are built. Computer networks, however, also expose an infrastructure that can be used by criminals or nation states to attack computing systems, to control the flow of messages, or to distribute malicious programs to potentially large numbers of targeted systems. This module educates students about how computer networks can be used to obtain information about remote systems, to manipulate the flow of data traffic, to disrupt access to remote services, or to control malicious software using botnets and distributed command and control channels. The module also covers technologies that help to protect the integrity of computer networks and that provide generic security services that can be used by applications requiring secure communication.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. describe techniques to obtain information about networked computing systems
2. contrast mechanisms in the different network protocol layers for traffic manipulation and redirection
3. explain how distributed denial of service attacks are executed and how botnets are constructed
4. evaluate security mechanisms such as firewalls and anomaly / intrusion detection systems
5. analyze generic security protocols such as IPsec, TLS, SSH and how they have evolved
6. compare protocols aiming to secure the network infrastructure (name resolution, routing)
7. evaluate the security properties of modern software-defined network architectures
8. design scalable solutions for protecting communication in distributed applications</td></tr>
</table>

Indicative Literature

- William Stallings: Cryptography and Network Security: Principles and Practice, 7th edition, Pearsons, 2018
- Chris McNab, Network Security Assessment, O'Reilly, 2017
- James Forshaw: Attacking Network Protocols, A Hacker's Guide to Capture, Analysis, and Exploitation, no starch press, 2017

Usability and Relationship to other Modules

- The module serves as a mandatory elective module in the cyber security specialization. It builds on the cryptography module, which provides the necessary knowledge of cryptographic primitives that are used to protect data exchanged over computer networks and to authenticate communicating peers.
- This module belongs to the Software Engineering Track in the MSc AST

Examination Type: Module Examination

Assessment: Written examination

Duration: 120 min
Weight: 100%

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.13 IDE Development

<table>
<tr><td colspan="3">Module Name
IDE Development</td><td>Module Code
MAST-207</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components
Number | Name | Type | CP</td></tr>
<tr><td>MAST-207-A</td><td colspan="3">IDE Development</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-207-B</td><td colspan="3">IDE Development - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Timofey Bryksin</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td colspan="3"></td><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
The students should have a strong foundation in programming concepts and practices.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module is designed to introduce students to a modern approach to creating integrated development tools. The course covers the main IDE modules: lexer, parser, code analyzer, local and global caches, code navigation, code modification and refactorings. In addition to the theoretical review, students will gradually develop their own IDE during the course. The course was created and implemented with the support of JetBrains. It is an elective module .

Content
● Development tools. An introduction to the history and architecture of the IDE.
● Data structures for working with text. Text editor and document markup.
● Virtual file system, the concepts of the PSI model and the design model.
● Introduction to the theory of formal languages.
● Lexical analysis.
● Parsing, abstract syntax trees.
● Semantic analysis, symbol tables and link resolution.
● Introduction to type systems and type inference.
● Introduction to static analysis.
● Abstract interpretation, control flow analysis and data flow analysis.
● Interprocedural analysis and call graph.
● Help with typing and code completion. Search and navigation through the code.
● Modification of the abstract syntax tree. Code generation based on the abstract syntax tree. Auto-formatting. Automatic refactoring.
● Debugger and debugging symbols, expression evaluation during debugging.
● Instrumentation, profiling and tracing.</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. demonstrate a thorough understanding of the algorithms, data structures, and methods underlying the operation of modern IDEs and static analysis tools.
2. conduct research in the field of IDE development by identifying, analyzing, and developing new specific algorithms necessary to solve problems that arise during the development process.
3. apply practical skills to address applied problems that emerge during the development of an IDE, such as designing user interfaces, optimizing performance, and implementing advanced features.
4. evaluate and compare various IDEs and static analysis tools, considering factors such as usability, efficiency, and extensibility.
5. collaborate effectively with a team to design, implement, and refine an IDE or a static analysis tool, leveraging version control systems, project management tools, and communication skills

Indicative Literature

"Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin

"Refactoring: Improving the Design of Existing Code" by Martin Fowler

"The Pragmatic Programmer: From Journeyman to Master" by Andrew Hunt and David Thomas

Usability and Relationship to other Modules

- The course provides a comprehensive coverage of the tools and technologies used for developing integrated development environments (IDEs). This can include topics such as IDE architecture, plugin development, debugging, code refactoring, version control integration and software testing. The course is suitable for students who want to learn about the challenges and opportunities of IDE development and how to use the technologies to develop IDEs. The course is also beneficial for students who want to pursue a career in software development, software engineering or software testing.
- This module belongs to the Software Engineering Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.14 Advanced Deep Learning

<table>
<tr><td colspan="2">Module Name
Advanced Deep Learning</td><td>Module Code
MAST-105</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="5">Module Components

Number | Name | Type | CP</td></tr>
<tr><td>MAST-105-A</td><td colspan="2">Advanced Deep Learning</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-105-B</td><td colspan="2">Advanced Deep Learning - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="2">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)

Duration
1 semester</td><td>Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)

Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation
Good understanding of the fundamental concepts of deep learning, such as neural networks, backpropagation, and gradient descent.</td></tr>
<tr><td colspan="5">Content and Educational Aims
The module aims to provide students with theoretical knowledge and practical skills in understanding the advanced architectures and models for deep learning, the optimization techniques for deep learning, the regularization and regularization methods for deep learning, the transfer learning and multi-task learning for deep learning, the generative models for deep learning, being familiar with the application of deep learning in different fields, to conduct research in deep learning.

Content:
● Advanced architectures and models for deep learning
● Optimization techniques for deep learning
● Regularization and regularization methods
● Transfer learning and Multi-task learning
● Generative Models
● Applications of deep learning
● Research in deep learning</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. implement advanced architectures and models for deep learning, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and generative models.
2. optimization techniques for deep learning, such as stochastic gradient descent (SGD), Adagrad, and Adam.
3. implement regularization methods for deep learning, such as dropout, weight decay, and early stopping.
4. apply transfer learning and multi-task learning techniques.
5. implement generative models such as GANs, VAEs and Autoencoders.
6. apply deep learning to different fields such as computer vision, natural language processing and speech recognition.
7. conduct research in deep learning by reading and understanding recent papers and be able to critically evaluate the results.

Indicative Literature

"Deep Learning" by Yoshua Bengio, Ian Goodfellow, and Aaron Courville, MIT Press, 2016

"Neural Networks and Deep Learning: A Textbook" by Charu Aggarwal, Springer, 2018

"Generative Adversarial Networks" by Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press, 2017

"Deep Learning for Computer Vision" by Rajalingapuram Kannan and Sridevi Sarma, Springer, 2018

"Deep Learning for Natural Language Processing" by Li Deng and Dong Yu, Cambridge Press, 2019

Kevin P. Murphy, “Probabilistic Machine Learning: Advanced Topics”, MIT Press, 2023, http://probml.github.io/book2

Usability and Relationship to other Modules

- Familiarity with basic concepts of machine learning, probability, and statistics is fundamental for almost all advanced modules in artificial intelligence and data science. This module additionally introduces advanced concepts of deep learning, such as advanced architectures, optimization techniques, and generative models, that are needed in advanced AI and data science-oriented modules in the 2nd year of the MSc program, as well as for research purposes.
- This module belongs to the Data Science Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination Duration: 60 min

Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.15 Recommender Systems

<table>
<tr><td colspan="2">Module Name
Recommender Systems</td><td>Module Code
MAST-202</td><td>Level (type)
Year 2</td><td>CP
5.0</td></tr>
<tr><td colspan="5">Module Components
Number | Name | Type | CP</td></tr>
<tr><td>MAST-202-A</td><td colspan="2">Recommender Systems</td><td>Lectures</td><td>2.5</td></tr>
<tr><td>MAST-202-B</td><td colspan="2">Recommender Systems - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Kirill Krinkin</td><td colspan="2">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)</td><td>Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (35 hours)
● Independent study (52.5 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td colspan="3"></td><td>Duration
1 semester</td><td>Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation
Basic concepts of machine learning, such as supervised and unsupervised learning, and the different types of models and algorithms.</td></tr>
<tr><td colspan="5">Content and Educational Aims
As part of the study of the module, students will get acquainted with the principles of recommender systems and consider issues related to the design features and use of such systems. After completing the course, students will navigate the methods of building and evaluating recommender systems from basic non-personalized approaches, recommendations based on content characteristics (content-based), collaborative filtering, to adaptive and advanced ones based on machine learning methods. To master the module, students need knowledge of probability theory and mathematical statistics, linear algebra, basic concepts of machine learning.

Content
● Introduction to recommender systems.
● non-personalized models. Models based on content information.
● Collaborative filtering
● Advanced Techniques for Building Factorization Models
● Accounting for contextual information in models</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. choose appropriate algorithms for building models
2. use summary statistics
3. explain the key concepts behind recommendations
4. explain the difference between user-based and item-based approaches
5. create a profile of personal interests
6. create product association recommendations
7. combine collaborative filtering and content-based recommendations
8. build recommendations based on collaborative filtering

Indicative Literature

"Recommender Systems" by Jannach, Dietmar and Zanker, Markus and Felfernig, Alexander and Friedrich, Gerhard and Loos, Peter. Springer, 2017

"Programming Collective Intelligence" by Toby Segaran, O'Reilly Media, 2007

"Deep Learning for Recommender Systems" by Balázs Hidasi, Google AI, 2019

"Recommender Systems Handbook" by Francesco Ricci, Lior Rokach and Bracha Shapira, Springer, 2011

"Matrix Factorization Techniques for Recommender Systems" by Yehuda Koren, Robert Bell and Chris Volinsky, AT&T Labs, 2009

Usability and Relationship to other Modules

- This module additionally introduces advanced concepts of recommender systems, such as collaborative filtering, matrix factorization, and deep learning-based approaches, that are needed in advanced AI and data science-oriented modules in the 2nd year of the MSc program, as well as for research purposes.
- This module belongs to the Data Science Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.16 Machine Learning in Software Engineering

<table>
<tr><td colspan="3">Module Name
Machine Learning in Software Engineering</td><td>Module Code
MAST-203</td><td>Level (type)
Year 2</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-203-A</td><td colspan="3">Machine Learning in Software Engineering</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-203-B</td><td colspan="3">Machine Learning in Software Engineering - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Timofey Bryksin</td><td colspan="3">Program Affiliation
• MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills:
☒ Understanding of machine learning and deep learning approaches used for natural language processing.
☒ Experience in programming in Python.</td><td>Frequency
Annually
(Fall)

Duration
1 semester</td><td colspan="2">Forms of Learning and Teaching
• Lecture attendance (35 hours)
• Project (35 hours)
• Independent study (35 hours)
• Exam preparation (20 hours)

Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
Fundamental concepts of machine learning such as supervised and unsupervised learning, and the different types of models and algorithms.</td></tr>
<tr><td colspan="6">Content and Educational Aims
Machine learning is actively used in a variety of areas, software engineering in this sense is no exception. This course offers for consideration one and a half dozen practical problems from the field of programming and software development, as well as the scope of machine learning to solve them: what data and methods are used for this, what difficulties arise, what is the current progress in these tasks and what are the problems in general now relevant in the field of machine learning in SE. The course deals with the most relevant scientific articles of recent years, and in order to receive an assessment, students must complete a group practical project on one of the proposed topics.
Content
• machine learning problem statement
• using machine learning for prediction and estimation
• using machine learning for code synthesis problems
• using machine learning to optimize code architecture
• using machine learning to find duplicates
• using natural language processing techniques
• using machine learning to analyze code</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. know the areas of expedient application of the machine learning method, including the development of software projects. Read their own and other people's code, and debug the program. Determine the appropriateness of applying machine learning methods for the selected task.
2. know the main approaches and methods of machine learning, understand their strengths and weaknesses, the limits of applicability. Able to measure the effectiveness of the constructed models.
3. develop models and prototypes of applications for the selected task in common programming languages.
4. formulate an algorithm for solving a problem in the form of a sequence of actions based on machine learning methods. Implement algorithms for solving the selected problem in suitable programming languages and using appropriate libraries.

Indicative Literature

"Applied Machine Learning for Software Engineering" by Markus Helfert and Michael Sheng, Springer, 2020

"Machine Learning for Software Engineers" by David C. Anastasiu and Zoran Duric, O'Reilly Media, 2018

"Machine Learning for Software Developers" by David C. Anastasiu, Zoran Duric and Rishi Shah, O'Reilly Media, 2019

"Machine Learning for Software Quality" by Juergen Rilling, Springer, 2020

"Machine Learning in Software Engineering" by Jörg Kienzle and Wojciech Cellary, Springer, 2018

Usability and Relationship to other Modules

- Familiarity with basic concepts of machine learning and software engineering is fundamental for almost all advanced modules in artificial intelligence and software engineering. This module additionally introduces advanced concepts of machine learning applied to software engineering, such as applying machine learning techniques to software development, testing and maintenance that are needed in advanced AI and software engineering-oriented modules in the 2nd year of the MSc program, as well as for research purposes.
- This module belongs to the Data Science Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.17 Bayesian Methods in Machine Learning

<table>
<tr><td colspan="3">Module Name
Bayesian Methods in Machine Learning</td><td>Module Code
MAST-204</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-204-A</td><td colspan="3">Bayesian Methods in Machine Learning</td><td>Lectures</td><td>2.5</td></tr>
<tr><td>MAST-204-B</td><td colspan="3">Bayesian Methods in Machine Learning - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="3">Program Affiliation
• MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)

Duration
1 semester</td><td colspan="2">Forms of Learning and Teaching
• Lecture attendance (17.5 hours)
• Tutorial attendance (35 hours)
• Independent study (52.5 hours)
• Exam preparation (20 hours)

Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
Bayesian methods are based on probability theory, so a solid understanding of probability and statistics concepts such as probability distributions, Bayes' theorem, and statistical inference is essential.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module focuses on the application of Bayesian methods to machine learning, providing students with theoretical knowledge and practical skills to incorporate probabilistic modeling and Bayesian techniques in their machine learning projects. The course will cover key Bayesian concepts, Bayesian inference methods, the use of Bayesian approaches in various machine learning algorithms, and the advantages of Bayesian techniques in handling uncertainty and modeling complex data.

Content:
• Introduction to Bayesian methods: Bayesian probability theory, conjugate priors, and Bayesian decision theory.
• Bayesian inference: Markov Chain Monte Carlo (MCMC) methods, Gibbs sampling, and Metropolis-Hastings algorithm.
• Bayesian linear regression and classification: Bayesian model selection, regularization, and hierarchical models.
• Bayesian non-parametric methods: Gaussian processes, Dirichlet processes, and their applications in machine learning.
• Bayesian approaches in various machine learning algorithms: Bayesian neural networks, Bayesian clustering, and Bayesian mixture models.
• Advantages and challenges of Bayesian methods in machine learning: handling uncertainty, modeling complex data, and computational efficiency."</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. Own heuristics to speed up the work of neuro-Bayesian algorithms and to reduce the dispersion of stochastic gradients
2. Know different variations of Bayesian inference methods
3. analyze the theoretical properties of the considered machine learning algorithms
4. select and train generative models from the GAN family
5. select and train generative models from the VAE family
6. compress neural networks based on the Bayesian approach

Indicative Literature

"Pattern Recognition and Machine Learning" by Christopher M. Bishop, Springer, 2006

"Bayesian Data Analysis" by Andrew Gelman, John Carlin, Hal Stern, David Dunson, Aki Vehtari, and Donald Rubin, CRC Press, 2013

"Probabilistic Programming & Bayesian Methods for Hackers" by Cameron Davidson-Pilon, Addison-Wesley, 2015

"Machine Learning: A Probabilistic Perspective" by Kevin P. Murphy, MIT Press, 2012

Usability and Relationship to other Modules

- Familiarity with basic probability and statistics, as well as machine learning concepts, is fundamental for almost all advanced modules in artificial intelligence and data science. This module additionally introduces advanced concepts of Bayesian methods and their application in machine learning, which are needed in advanced AI and data science-oriented modules in the 2nd year of the MSc program and also for research purposes.
- This module belongs to the Data Science Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.18 Advanced Functional Programming

<table>
<tr><td colspan="3">Module Name
Advanced Functional Programming</td><td>Module Code
MAST-104</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-104-A</td><td colspan="3">Advanced Functional Programming</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-104-B</td><td colspan="3">Advanced Functional Programming - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Anton Podkopaev</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology(AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements

Pre-requisites
☒ none

Co-requisites
☒ none

Knowledge, Abilities, or Skills
☒ none</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
The basics of functional programming, including higher-order functions, recursion, and immutability.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module aims to provide students with a thorough understanding advanced concepts and design patterns in functional programming, interacting with the external world using functional programming, being able to profile and debug functional programs, understanding and implementing persistent data structures in functional programming, understanding the principles of functional programming and be able to use them to solve complex problems.

Content:
● Advanced functional programming concepts and design patterns
● Interacting with the external world in functional programming
● Profiling and debugging functional programs
● Persistent data structures and their implementation in functional programming
● Type-level programming and meta-programming in functional programming
● Hands-on experience with the Haskell programming language</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. understand advanced concepts and design patterns in functional programming, and be able to apply them to design and implement functional programs.
2. interact with the external world using functional programming techniques, such as IO Monad, and other related concepts.
3. profile and debug functional programs and identify performance bottlenecks.
4. understand and implement persistent data structures in functional programming, such as persistent arrays and persistent linked lists.</td></tr>
</table>

5. learn how to use type-level programming and meta-programming in functional programming, such as type-level programming with type families, GADT, and other related topics.
6. develop hands-on experience with the Haskell programming language and be able to apply functional programming concepts in other mainstream programming languages.

Indicative Literature

Hudak, Paul. "The Haskell school of expression: learning functional programming through multMedia." (1999).

Thompson, Simon. Haskell: The Craft of Functional Programming. Addison-Wesley, 1999.

Löh, Andres. "Functional pearl: The monad-reader pattern." (2009).

Marlow, Simon. Parallel and Concurrent Programming in Haskell. O'Reilly Media, Inc., 2013.

O'Sullivan, Bryan, John Goerzen, and Don Stewart. Real World Haskell. O'Reilly Media, Inc., 2008.

Röjemo, András, and Erik Hesselink. "Functional pearl: Implicit configurations." (2010).

Wadler, Philip, and Stephanie Weirich. "The essence of functional programming." (2002).

Usability and Relationship to other Modules

- Familiarity with the basics of functional programming, and the Haskell programming language is fundamental for almost all advanced modules in computer science that rely on functional programming. This course introduces advanced concepts of functional programming such as persistent data structures, type-level programming, and meta-programming, which are needed in advanced programming-oriented modules such as functional software design, functional programming languages, and formal verification. Understanding the principles of functional programming and the Haskell programming language will enable students to apply these concepts in various fields such as computer science, finance, and data analysis. Additionally, the course provides a solid ground to use functional programming principles in mainstream programming languages such as Scala, F#, or OCaml.
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.19 Weak Memory Models

<table>
<tr><td colspan="2">Module Name
Weak Memory Models</td><td>Module Code
MAST-208</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="5">Module Components</td></tr>
<tr><td>Number</td><td colspan="2">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-208-A</td><td colspan="2">Weak Memory Models</td><td>Lectures</td><td>2.5</td></tr>
<tr><td>MAST-208-B</td><td colspan="2">Weak Memory Models - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="2">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="2">Entry Requirements
Pre-requisites
☒ none
Co-requisites
☒ none
Knowledge, Abilities, or Skills
☒ none</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (35 hours)
● Independent study (52.5 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td colspan="2"></td><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation
The basics of concurrent programming and its relation to parallel programming. Having a solid understanding of programming languages such as C and C++, as well as a good knowledge of the memory model of these languages. Familiarizing with the basics of algorithms and data structures, as well as the principles of computer architecture.</td></tr>
<tr><td colspan="5">Content and Educational Aims
The module aims to provide students with a thorough understanding of weak memory models in modern programming languages, to give students the skills and knowledge to analyze the trade-offs between performance and guarantees provided to software developers, to equip students with the ability to implement and verify memory models in programming languages, to teach students about data-race-freedom (DRF) and its implications on program behaviors, to give students the ability to apply the concepts and techniques learned in the course to real-world problems and projects, to expose students to the latest research in the field of weak memory concurrency, to provide students with an understanding of modern formalisms for expressing memory models of programming languages and CPU architectures, to give students an overview of open problems in the research area of weak memory concurrency, and potential avenues for further research.

Content:
● Overview of weak memory models in modern programming languages
● Formalisms for expressing memory models of programming languages and CPU architectures
● Study of modern memory models such as TSO, PSO, and ARM
● Analysis of the trade-offs between performance and guarantees provided to software developers
● Study of data-race-freedom (DRF) and its implications on program behaviors
● Overview of open problems in the research area of weak memory concurrency
● Study of techniques for implementing and verifying memory models in programming languages
● Real-world case studies and projects</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. understand the theoretical foundations of weak memory models and its relation to other memory models such as sequential consistency.
2. develop proficiency in using formalisms for expressing memory models of programming languages and CPU architectures.
3. learn how to analyze the trade-offs between performance and guarantees provided to software developers in weak memory models.
4. understand the implications of data-race-freedom (DRF) on program behaviors, and how to implement it in weak memory models.
5. learn how to implement and verify memory models in programming languages.
6. understand the open problems in the research area of weak memory concurrency and potential avenues for further research.
7. apply the concepts and techniques learned in the course to real-world problems and projects.
8. understand the differences between popular weak memory models such as TSO, PSO, and ARM, and when to use each.
9. develop the ability to evaluate the performance and guarantees of different memory models, and choose the most appropriate one for a given problem or system.

Indicative Literature

John Regehr: "The Memory Model in C and C++" (2019)

Jim Davis, Paul E. McKenney: "Is Parallel Programming Hard, And, If So, What Can You Do About It?" (2018)

Maurice Herlihy, Nir Shavit: "The Art of Multiprocessor Programming" (2012)

Hans-J. Boehm, Alan J. Demers, Scott Shenker, and L. Peter Deutsch: "The Weak Memory Model: A Useful Lie" (1996)

Usability and Relationship to other Modules

- A solid understanding of weak memory models and concurrent programming is essential for many advanced topics in computer science, including parallel programming, distributed systems, and computer architecture. This course provides a deep dive into the theory and practice of weak memory models, and equips students with the skills and knowledge necessary to analyze and optimize the performance of concurrent systems. Additionally, the course covers the advanced features of popular memory models such as TSO, PSO, and ARM, which are essential for advanced topics in the field of computer science.
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.20 Virtual Machines

<table>
<tr><td colspan="2">Module Name
Virtual Machines</td><td>Module Code
MAST-106</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="5">Module Components</td></tr>
<tr><td>Number</td><td colspan="2">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-106-A</td><td colspan="2">Virtual Machines</td><td>Lecture</td><td>2.5</td></tr>
<tr><td>MAST-106-B</td><td colspan="2">Virtual Machines - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Kirill Krinkin</td><td colspan="2">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)</td><td>Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (70 hours)
● Exam preparation (20 hours)</td></tr>
<tr><td>Duration
1 semester</td><td>Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation
The basics of computer systems and operating systems.</td></tr>
<tr><td colspan="5">Content and Educational Aims
The objectives of mastering the module "Virtual Machines" are the formation of students' theoretical knowledge and practical skills on the basics of working and building modern virtual machines. Considerable attention is paid to issues related to the theoretical foundations and practical methods for creating modern efficient virtual machines that meet the requirements for security and speed.

Content:
● Introduction. Virtualization and virtual machines
● Typical VM components. Multithreading
● Implementation of the executing component
● Competitiveness, safety, reliability. Performance
● Virtual machine designs. Implementation examples</td></tr>
<tr><td colspan="5">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. know the principles of building virtual machines; main ways to improve performance of virtual machines; main features of the implementation of existing VMs.
2. be able to create virtual machines; create a JIT compiler; create virtual machines with support for multi-threaded execution mode.</td></tr>
</table>

3. have the skills (gain experience) of building secure and reliable virtual machines; application of implementation algorithms for JIT compilers, physical memory managers.

Indicative Literature

"Modern Operating Systems" by Andrew S. Tanenbaum and Herbert Bos, Prentice Hall, 4th edition, 2015

"Virtualization: A Beginner's Guide" by Neil J. Ross and Anthony Velte, McGraw-Hill Professional, 1st edition, 2009

"Virtualization: From the Desktop to the Enterprise" by Chris Wolf, John Wiley & Sons, 1st edition, 2007

"Docker: Up & Running: Shipping Reliable Containers in Production" by Karl Matthias and Sean P. Kane, O'Reilly Media, 1st edition, 2016

"Kubernetes: Up and Running: Dive into the Future of Infrastructure" by Kelsey Hightower, Brendan Burns and Joe Beda, O'Reilly Media, 1st edition, 2017.

Usability and Relationship to other Modules

- Familiarity with basic concepts of computer systems and operating systems is fundamental for almost all advanced modules in computer science and software engineering. This module additionally introduces advanced concepts of virtual machines, such as virtualization, containerization, and their use in cloud computing and distributed systems, that are needed in advanced system-oriented modules in the 2nd year of the MSc program, as well as for research purposes
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.21 Metacomputations

<table>
<tr><td colspan="2">Module Name
Metacomputations</td><td>Module Code
MAST-107</td><td>Level (type)
Year 1</td><td>CP
5.0</td></tr>
<tr><td colspan="5">Module Components</td></tr>
<tr><td>Number</td><td colspan="2">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-107-A</td><td colspan="2">Metacomputations</td><td>Lectures</td><td>2.5</td></tr>
<tr><td>MAST-107-B</td><td colspan="2">Metacomputations - Tutorial</td><td>Tutorial</td><td>2.5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Anton Podkopaev</td><td colspan="2">Program Affiliation
• MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Spring)</td><td>Forms of Learning and Teaching
• Lecture attendance (17.5 hours)
• Tutorial attendance (17.5 hours)
• Independent study (70 hours)
• Exam preparation (20 hours)</td></tr>
<tr><td colspan="3"></td><td>Duration
1 semester</td><td>Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation
To master the module, students need knowledge gained as a result of studying the module s "Functional programming", "Compilers", "Semantics of programming languages".</td></tr>
<tr><td colspan="5">Content and Educational Aims
Metacomputing is a branch of programming devoted to the development of methods for analyzing and transforming programs by implementing constructive metasystems (metaprograms) over programs. Metacomputing primarily includes the theory of supercompilation and related methods and tools. As part of the study of the module , students will gain an understanding of the basic principles of metacomputing and supercompilation, learn how to apply them to implement partial calculators and supercompilers. The purpose of mastering the module is to develop students' theoretical knowledge and practical skills on the basics of the analysis of programming languages, the development of metacalculators for various programming languages, the development by students of the methods of static and dynamic analyzes, semantic analysis, and abstract interpretation.
Content
• Introduction to Metacomputations
• Program Specialization
• Program Specialization Criteria and Jones Optimality
• Collapsing a tower of interpreters
• Positive, Perfect, Multi-level, and Multi-result Supercompilation</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to

1. know the basic concepts and facts of theories of programming languages and metacomputing, such as Futamura-Ershov-Turchin projections, program partitioning, self-applying, compilation, semantics and semantic analysis, types, data flow analysis, link-time analysis, termination, security, specialization, supercompilation, abstract interpretation and others.
2. know how to implement various types of metacalculators for functional and imperative programming languages.
3. have skills in the analysis of programs and programming languages.

Indicative Literature

"Metacomputing: Techniques and Applications" by Jarek Nabrzyski, Ian Foster, and Jack Dongarra, Cambridge University Press, 2005

"Metacomputing: Applications and Opportunities" by M. Parashar, Springer, 2018

"Metacomputing and Grid Technologies" by G. Fox, J. Frey, and T. Hey, John Wiley & Sons, 2003

"High-Performance Computing: Paradigm and Infrastructure" by David A. Bader, Springer, 2005

"Cloud Computing: Concepts, Technology & Architecture" by Thomas Erl, Prentice Hall, 2009

Usability and Relationship to other Modules

- Familiarity with basic concepts of computer science and distributed computing is fundamental for almost all advanced modules in computer science and software engineering. This module additionally introduces advanced concepts of metacomputing, such as grid computing, cloud computing and high-performance computing, that are needed in advanced distributed systems-oriented modules in the 2nd year of the MSc program, as well as for research purposes.
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Component Examination

Component 1: Lecture

Assessment: Written examination — Duration: 60 min, Weight: 50%

Scope: All theoretical intended learning outcomes of the module

Component 2: Tutorial

Assessment: Practical assessment (Programming assignments) — Weight: 50%

Scope: All practical intended learning outcomes of the module

Completion: To pass this module, the examination of each module component has to be passed with at least 45%.

3.22 Dependent Types

<table>
<tr><td colspan="3">Module Name
Dependent Types</td><td>Module Code
MAST-209</td><td>Level (type)
Year 2</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-209-A</td><td colspan="3">Dependent Types</td><td>Lectures/ Tutorial</td><td>5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Anton Podkopaev</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ Functional Programming</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
● Lecture attendance (17.5 hours)
● Tutorial attendance (17.5 hours)
● Independent study (90 hours)</td></tr>
<tr><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
To master the module students need the knowledge gained from studying the module s "Formal languages", and "Functional programming".</td></tr>
<tr><td colspan="6">Content and Educational Aims
In this course, we'll learn the basics of program verification, specification, and formal theorem proving, We will also talk about the theoretic foundation of dependently typed systems. By the end of the course, students will be able to formulate and prove correctness properties of functional programs, algorithms, and simple maths theorems.
Content:
● Simple types
● Subtypes and Recursive Types
● Polymorphic types
● Type systems of higher orders</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. understand the relationship between logic and functional programming
2. know various type-theoretic constructions occurring in dependently typed languages
3. formulate and prove simple theorems
4. prove correctness of various algorithms</td></tr>
</table>

Indicative Literature

"Types and Programming Languages" by Benjamin C. Pierce, MIT Press, 2002

"Advanced Topics in Types and Programming Languages" by Benjamin C. Pierce, MIT Press, 2005

"Introduction to the Theory of Programming Languages" by Michael J.C. Gordon, Cambridge University Press, 1996

Usability and Relationship to other Modules

- Familiarity with basic concepts of programming languages and formal methods is fundamental for almost all advanced modules in computer science and software engineering. This module additionally introduces advanced concepts of type systems, type inference, and type-based program analysis that are needed in advanced programming languages-oriented modules in the 2nd year of the MSc program, as well as for research purposes.
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Examination

Assessment: Practical assessment (Programming assignments) Weight: 100%

Scope: All intended learning outcomes of the module

Completion: To pass this module, the examination has to be passed with at least 45%.

3.23 Homotopy Type Theory

<table>
<tr><td colspan="3">Module Name
Homotopy Type Theory</td><td>Module Code
MAST-210</td><td>Level (type)
Year 2</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components</td></tr>
<tr><td>Number</td><td colspan="3">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-210-A</td><td colspan="3">Homotopy Type Theory</td><td>Lecture/ Tutorial</td><td>5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="3">Program Affiliation
• MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3" rowspan="2">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually
(Fall)</td><td colspan="2">Forms of Learning and Teaching
• Lecture/Tutorial attendance (35 hours)
• Independent study (90 hours)</td></tr>
<tr><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
The basics of type theory and type systems in programming languages.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module aims to provide students with theoretical knowledge and practical skills in understanding the basic principles and concepts of homotopy type theory, developing the ability to express homotopy-theoretic concepts in the language of homotopy type theory, learning the relationship of homotopy type theory with logic, set theory, and group theory, understanding how homotopy type theory can be applied in programming languages, developing the ability to use homotopy type theory to prove theorems in geometry and topology.

Content:
• Introduction to type theory and its extensions
• Fundamentals of homotopy theory and its relationship to geometry and topology
• Logic, set theory, and group theory in the context of homotopy type theory
• Concepts from homotopy theory expressed in the language of homotopy type theory</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. understand the fundamental concepts and principles of homotopy type theory and its relationship to type theory, logic, set theory, group theory, geometry and topology.
2. express homotopy-theoretic concepts in the language of homotopy type theory and use this language to prove theorems in geometry and topology.
3. understand the connection between homotopy type theory and programming languages and be able to apply homotopy type theory in the context of programming languages.
4. write and understand formal proofs in homotopy type theory.
5. understand the basic concepts of category theory and how they relate to homotopy type theory.
6. understand the basic concepts of functional programming and how they relate to homotopy type theory
7. understand how the concepts of homotopy type theory can be used to reason about and reason with the properties of programs and systems.</td></tr>
</table>

8. communicate effectively and express your understanding of homotopy type theory in written and oral form.

Indicative Literature

Martin-Löf, Per. "Intuitionistic type theory." Bibliopolis (1984): 343-441.

Awodey, Steve. Category theory. Vol. 48. Oxford: Clarendon Press, 2006.

Hofmann, Martin, and Thomas Streicher. "The groupoid interpretation of type theory." Mathematical Structures in Computer Science 8.6 (1998): 613-630.

Homotopy Type Theory: Univalent Foundations of Mathematics. The Univalent Foundations Program, Institute for Advanced Study, 2013.

HoTT Book, Homotopy Type Theory: Univalent Foundations of Mathematics. The Univalent Foundations Program, Institute for Advanced Study, 2013.

Usability and Relationship to other Modules

- Familiarity with basic concepts of type theory, geometry, and topology is fundamental for almost all advanced modules in mathematics and computer science that rely on homotopy theory. This course introduces advanced concepts of homotopy type theory and its connection to programming languages that are needed in advanced programming-oriented modules such as type systems, programming semantics, and formal verification. Additionally, understanding the principles of homotopy type theory will enable students to apply these concepts in various fields such as mathematics, physics, computer science, and engineering.
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Examination

Assessment: Written examination — Duration 120 mins

Weight: 100%

Scope: All intended learning outcomes of the module

Completion: To pass this module, the examination has to be passed with at least 45%.

3.24 Category Theory for Programmers

<table>
<tr><td colspan="3">Module Name
Category Theory for Programmers (Programming Languages Track)</td><td>Module Code
MAST-211</td><td>Level (type)
Year 2</td><td>CP
5.0</td></tr>
<tr><td colspan="6">Module Components
Number | Name | Type | CP
MAST-211-A | Category Theory for Programmers | Lecture/ Tutorial | 5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Aleksandr Omelchenko</td><td colspan="3">Program Affiliation
● MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for AST</td></tr>
<tr><td colspan="3">Entry Requirements
Pre-requisites: ☒ none
Co-requisites: ☒ none
Knowledge, Abilities, or Skills: ☒ none</td><td>Frequency
Annually (Fall)
Duration
1 semester</td><td colspan="2">Forms of Learning and Teaching
● Lecture/Tutorial attendance (35 hours)
● Independent study (90 hours)
Workload
125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation
Good understanding of the fundamental concepts of mathematics, such as set theory, logic and functions.</td></tr>
<tr><td colspan="6">Content and Educational Aims
The module is aimed at developing students' theoretical knowledge and practical skills related to the use of functional programming languages. The course introduces the basic concepts of category theory, such as category, functor and monad. Students will learn to understand commutative diagrams. The course will help you better understand modern programming languages such as Agda, Coq and Idris. To master the module , students need to have knowledge of set theory, algebra, and topology.

Content
• Introduction to category theory and its basic structures
• Fundamental concepts and theorems of category theory
• Relationship with functional programming and type theory
• Introduction to toposes and their internal language</td></tr>
<tr><td colspan="6">Intended Learning Outcomes
Upon completion of this module, students will be able to
1. know the basics of category theory
2. understand categorical models of lambda calculus and simple type theory
3. understand the relationship between, logic, type theory, and category theory
4. work within the internal language of a topos</td></tr>
<tr><td colspan="6">Indicative Literature
"Category Theory for Programmers" by Bartosz Milewski, self-published, 2018
"Categories for the Working Mathematician" by Saunders Mac Lane, Springer, 1971</td></tr>
</table>

"Conceptual Mathematics: A First Introduction to Categories" by F. William Lawvere and Stephen Hoel Schanuel, Cambridge University Press, 1997

"The Joy of Cats" by Barry Mazur and Emily Riehl, American Mathematical Society, 2020

"Categories and Types in Logic, Language, and Physics" by Bob Coecke, Aleks Kissinger, and Mehrnoosh Sadrzadeh, Cambridge University Press, 2018

Usability and Relationship to other Modules

- Familiarity with basic concepts of mathematics and programming is fundamental for almost all advanced modules in computer science and software engineering. This module additionally introduces advanced concepts of category theory and its application to functional programming and type systems that are needed in advanced programming languages-oriented modules in the 2nd year of the MSc program, as well as for research purposes.
- This module belongs to the Programming Languages Track in the MSc AST

Examination Type: Module Examination

Assessment: Practical assessment (Programming assignments) Weight: 100%

Scope: All intended learning outcomes of the module

Completion: To pass this module, the examination has to be passed with at least 45%.

3.25 Research Project

<table>
<tr><td colspan="2">Module Name
Research Project</td><td>Module Code
MAST-201</td><td>Level (type)
Year 2</td><td>CP
5</td></tr>
<tr><td colspan="5">Module Components</td></tr>
<tr><td>Number</td><td colspan="2">Name</td><td>Type</td><td>CP</td></tr>
<tr><td>MAST-201-A</td><td colspan="2">Research Project</td><td>Project</td><td>5</td></tr>
<tr><td>Module Coordinator
Prof. Dr. Alexander Omelchenko</td><td colspan="2">Program Affiliation
• MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status
Mandatory elective for MSc AST</td></tr>
<tr><td colspan="2">Entry Requirements
Co-requisites
☒ none
Knowledge, Abilities, or Skills
Pre-requisites
☒ none</td><td>Frequency
Annually (Fall)</td><td colspan="2">Forms of Learning and Teaching
• Research group meetings (21 hours)
• Independent project work (104 hours)</td></tr>
<tr><td colspan="2"></td><td>Duration
1 semester</td><td colspan="2">Workload
125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation</td></tr>
<tr><td colspan="5">Content and Educational Aims
The competencies and knowledge earned in the first two semesters are deepened by developing a small research project. Students will be exposed to state-of-the-art research with the goal of reproducing results of recent research papers or extending ideas presented in recent research papers. Students will learn how to organize and execute a research project and how to present the results in the format of a typical research paper. Students are expected to participate in the meetings of the research group in which they are doing their research projects. The module is conducted together with JetBrains which provides research topics for the students.</td></tr>
<tr><td colspan="5">Intended Learning Outcomes
Upon completion of this module, students will be able to:
1. understand state-of-the-art research papers in a chosen field of specialization
2. plan a research project to reproduce research results or to extend ideas of recent research results
3. explain research questions and choose suitable methodologies to address them
4. document a research project in the style of a typical scientific paper</td></tr>
<tr><td colspan="5">Indicative Literature
▪ Recent publications provided by the research project supervisors.</td></tr>
<tr><td colspan="5">Usability and Relationship to other Modules</td></tr>
<tr><td colspan="5">Examination Type: Module Examination
Assessment: Project report (5000 words) Weight: 100%
Scope: All intended learning outcomes of the module.
Completion: To pass this module, the examination has to be passed with at least 45%.</td></tr>
</table>

3.26 Capstone Project 1

<table>
<tr><td colspan="3">Module Name
Capstone Project 1</td><td>Module Code
MCSSE-CAP-01</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP
MCSSE-CAP-01 | Capstone Project 1 | Project | 5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Manuel Oriol</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills:
• Programming skills in an imperative language at CS bachelor level
• Algorithms and data structure at CS bachelor level</td><td>Frequency

Annually (Fall)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Tutorials (35 hours)
• Group-based and independent project work (55 hours)

Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Train and advance programming, read about agile development, watch videos on ideation processes and read books on team and teamwork.</td></tr>
<tr><td colspan="6">Content and Educational Aims

This series of Capstone modules gives the possibility of experiencing knowledge and expertise learned in the master by a posteriori analysis, transformational adaptation and coherent planning hands-on practice. The series spans over three modules during which students develop a complete product from scratch. The project starts with an ideation process, creation of clickable demos and initial requirements. It continues with the practical creation of a software architecture and development of the solution. It then finishes with application of artificial intelligence and cybersecurity. During the project, students are going through various steps during which they are encouraged to talk directly to potential real-world customers and users, thus gathering an understanding of what real users and customers for their project might want.
The project is organized in tribes (20-30 people) in charge of exactly one project. The tribes are then further split in agile teams working with the advice of the instructors and the assistants (impersonating the business owners and product owners). The teams can be geographically distributed and work with an up-to-date environment supported with open source IDEs and engineering tools. Few lectures indicate the best practices to follow and the interim goals. Periodic meetings with instructor and teaching assistants steer the process towards the overall goal.

This instance is the first semester of the Capstone project that focuses on ideation and requirements elicitation.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes

1. Create and propose mocks
2. Perform requirements elicitation
3. Prototype
4. Approach customers and users
5. Specify user stories
6. Organize themselves through collaborative tools
7. Understand team dynamics and resolve most interpersonal issues</td></tr>
</table>

Indicative Literature

Agile the good the hype and the ugly. Book by Bertrand Meyer

The Five Dysfunctions of a Team. Book by Patrick Lencioni

Group dynamics and Teams interventions. Book by Timothy M. Franz

Online resources on team dynamics:

- https://www.challengeapplications.com/stages-of-team-development
- https://agilescrumguide.com/blog/files/tag-5-stages-of-team-development.html

Usability and Relationship to other Modules

It is highly recommended to take the three Capstone Project modules in their numerical order to gain the full experience of the project.

Examination Type: Module Examination

Assessment: Project Assessment Weight: 100%

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.27 Capstone Project 2

<table>
<tr><td colspan="3">Module Name
Capstone Project 2</td><td>Module Code
MCSSE-CAP-02</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP
MCSSE-CAP-02 | Capstone Project 2 | Project | 5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Manuel Oriol</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills:
• Programming skills in an imperative language at CS bachelor level
• Algorithms and data structure at CS bachelor level</td><td>Frequency

Annually (Spring)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Tutorials (35 hours)
• Group-based and independent project work (55 hours)

Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Train and advance programming, read about agile development, watch videos on ideation processes and read books on team and teamwork.</td></tr>
<tr><td colspan="6">Content and Educational Aims

This series of modules gives the possibility of experiencing knowledge and expertise learned in the master by aposteriori analysis, transformational adaptation and coherent planning hands-on practice. The course series spans over three modules during which students develop a complete product from scratch. The project starts with an ideation process, creation of clickable demos and initial requirements. It continues with the practical creation of a software architecture and development of the solution. It then finishes with application of artificial intelligence and cybersecurity. During the project students are going through various steps during which they are encouraged to talk directly to potential real-world customers and users, thus gathering an understanding of what real users and customers for their project might want.

The project is organized in tribes (20-30 people) in charge of exactly one project. The tribes are then further split in agile teams working with the advice of the instructors and the assistants (impersonating the business owners and product owners). The teams can be geographically distributed and work with an up-to-date environment supported with open source IDEs and engineering tools. Few lectures indicate the best practices to follow and the interim goals. Periodic meetings with instructor and teaching assistants steer the process towards the overall goal.

This instance is the second semester of the capstone project that focuses on architecture and base implementation.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes

1. Describe and defend a software architecture
2. Code in groups
3. Code as a large team
4. Integrate independent works
5. Use a source code versioning system
6. Specify user stories
7. Hold practical discussions with stakeholders
8. Organize themselves through collaborative tools</td></tr>
</table>

9. Understand team dynamics and resolve most interpersonal issues

Indicative Literature

Agile the good the hype and the ugly. Book by Bertrand Meyer

The Five Dysfunctions of a Team. Book by Patrick Lencioni

Group dynamics and Teams interventions. Book by Timothy M. Franz

Online resources on team dynamics:

- https://www.challengeapplications.com/stages-of-team-development
- https://agilescrumguide.com/blog/files/tag-5-stages-of-team-development.html

Usability and Relationship to other Modules

It is highly recommended to take the three Capstone Project modules in their numerical order to gain the full experience of the project.

Examination Type: Module Examination

Assessment: Project Assessment Weight: 100%
Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.28 Capstone Project 3

<table>
<tr><td colspan="3">Module Name
Capstone Project 3</td><td>Module Code
MCSSE-CAP-03</td><td>Level (type)
Year 1 and 2</td><td>CP
15</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP
MCSSE-CAP-03 | Capstone Project | Project | 15</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Manuel Oriol</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills:
• Programming skills in an imperative language at CS bachelor level
• Algorithms and data structure at CS bachelor level</td><td>Frequency

Annually (Fall)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

• Lectures (35 hours)
• Tutorials (35 hours)
• Group-based and independent project work (55 hours)

Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Train and advance programming, read about agile development, watch videos on ideation processes and read books on team and teamwork.</td></tr>
<tr><td colspan="6">Content and Educational Aims

This series of modules gives the possibility of experiencing knowledge and expertise learned in the master by aposteriori analysis, transformational adaptation and coherent planning hands-on practice. The course series spans over three modules during which students develop a complete product from scratch. The project starts with an ideation process, creation of clickable demos and initial requirements. It continues with the practical creation of a software architecture and development of the solution. It then finishes with application of artificial intelligence and cybersecurity. During the project students are going through various steps during which they are encouraged to talk directly to potential real-world customers and users, thus gathering an understanding of what real users and customers for their project might want.
The project is organized in tribes (20-30 people) in charge of exactly one project. The tribes are then further split in agile teams working with the advice of the instructors and the assistants (impersonating the business owners and product owners). The teams can be geographically distributed and work with an up-to-date environment supported with open source IDEs and engineering tools. Few lectures indicate the best practices to follow and the interim goals. Periodic meetings with instructor and teaching assistants steer the process towards the overall goal.

This instance is the third semester of the Capstone Project that focuses on integrating artificial intelligence, cybersecurity, and develops practices.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes

1. Know practical cybersecurity
2. Hold practical discussions with stakeholders
3. Practice of machine learning
4. Work with continuous improvements tools
5. Organize themselves through collaborative tools
6. Understand team dynamics and resolve most interpersonal issues</td></tr>
</table>

Indicative Literature

Agile the good the hype and the ugly. Book by Bertrand Meyer

The Five Dysfunctions of a Team. Book by Patrick Lencioni

Group dynamics and Teams interventions. Book by Timothy M. Franz

Online resources on team dynamics:

- https://www.challengeapplications.com/stages-of-team-development
- https://agilescrumguide.com/blog/files/tag-5-stages-of-team-development.html

Usability and Relationship to other Modules

It is highly recommended to take the three Capstone Project modules in their numerical order to gain the full experience of the project.

Examination Type: Module Examination

Assessment: Project Assessment Weight: 100%

Scope: All intended learning outcomes of the module.

Completion: To pass this module, the examination has to be passed with at least 45%.

3.29 Master Thesis

<table>
<tr><td colspan="3">Module Name
Master Thesis AST</td><td>Module Code
MAST-300</td><td>Level (type)
Year 2</td><td>CP
30</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP
MAST-300-T | Master Thesis AST | N.A. | 30</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Aleksandr Omelchenko</td><td colspan="3">Program Affiliation

• MSc Advanced Software Technology (AST)</td><td colspan="2">Mandatory Status

Mandatory for AST</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites

☒ None

Co-requisites

☒ None

Knowledge, Abilities, or Skills

• Proficiency in the area of the chosen thesis topic.</td><td>Frequency

Annually (Spring)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

• Private Study (750 hours)

Workload

750 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

Read the Syllabus.</td></tr>
<tr><td colspan="6">Content and Educational Aims

The aim of this module is to train students to motivate, design, carry out and document a research project in one of the areas represented by the research groups of the faculty of AST. Some familiarity with the requisite Advanced Software Technology techniques will typically have been acquired in one of the preceding Advanced Projects. The thesis topic is determined in mutual agreement with the module instructor. They may arise from the ongoing research in the instructor's own research group, but it is also possible for a student to adopt a topic of his/her own choice provided the instructor agrees to supervise it. The thesis work comprises the full cycle of a scientific research endeavor: (i) identifying a relevant open research question, (ii) carrying out a literature survey to put the planned work in its context and relate it to the state of the art (SoA), (iii) formulate a concrete research objective, (iv) design a research plan including a statement of criteria to evaluate the success of the project, (v) carry out the plan (with the possibility to change the original plan when motivated), (vi) document the results, (vii) analyze the results with respect to the SoA, the original objective, and the success criteria, and (viii) document all of this in a thesis report. All of this work should be done with as much self-guidance as can be reasonably expected. The instructor will likely give substantial guidance for (i) and (iii), whereas the other aspects will be addressed with larger degrees of self-guidance. A research proposal document summarizing (i) – (iv) is expected as an interim result and milestone (target size: 10 pages). In the first weeks of the course, an intense taught tutorial on scientific working and writing is held. The subsequent weeks follow a seminar style where students present and discuss literature as well as their own results to date. The project consists of the proposal, a thesis report (target size: 30–60 pages, and an oral presentation at the end of the course.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes

Discipline-Specific Skills (subject area depending on research discipline of the hosting group):

1. understanding, at a professional level, of a circumscribed segment of the hosting group's research area;
2. ability to apply specific and selected AST techniques, as required for the project, at a professional level;
3. general professional skills;
4. designing and carrying out the full cycle of a scientific research project in a professional manner;
5. formulating a research proposal such that that it could serve as a funding proposal;
6. writing a research thesis such that it could be submitted to a scientific publication venue, or as a project report to a funding agency or industrial client;
7. presentation of project results for specialists and non-specialists.</td></tr>
</table>

Indicative Literature

N.A.

Usability and Relationship to other Modules

Examination Type: Module Examination

Assessment Component 1: Thesis	Length: 30 – 60 pages Weight: 75%

Scope: All intended learning outcomes of this module.

Assessment Component 2: Oral Examination (Defense)	Duration: 20 minutes Weight: 25%

Scope: Mainly presentation of project results but the presentation touches all intended learning outcomes

Completion: This module is passed with an assessment-component weighted average grade of 45% or higher.

4 Management Modules

4.1 Agile Product Development & Design

<table>
<tr><td colspan="3">Module Name
Agile Product Development & Design</td><td>Module Code
MCSSE-MGT-01</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="6">Module Components
Number | Name | Type | CP
MCSSE-MGT-01 | Agile Product Development & Design | Lecture | 5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Tilo Halaszovich</td><td colspan="3">Program Affiliation

▪ MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills: ☒ None</td><td>Frequency

Annually (Fall)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

▪ Lecture (80 hours)
▪ Private study (45 hours)

Workload

125 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

N.A.</td></tr>
<tr><td colspan="6">Content and Educational Aims

This course is focused on key aspects of agile product and service development and design process.
State-of-the-art user centered design methods will be at the core of the course.

The overall goal of this module is to help managers without a business degree to learn, understand and practice agile customer- and data-driven innovation processes in the information age. This module helps students to understand today's real-life challenges in a complex world, with wicked problems and with multiple stakeholder interests, where unpredictable is common, and where managers need to focus on achieving goals rather than repetitive tasks.
Students learn to develop and present innovative user-centered and theory-oriented solutions for real-world challenges in an IT-driven world.
This course is strongly based on the agile paradigm of user-centeredness, user-centered design and the ideas of the Service Dominant Logic. Service-dominant (S-D) logic is a meta-theoretical framework for explaining value co-creation, through exchange, among configurations of actors.
Major challenges and concerns will be reflected:
• the role of the customer and data in a transformed business world
• new theories, concepts, and approaches (such as service dominant logic, customer integration, gamification, new service models)
• new methods and management techniques in (service) innovation (Design Thinking)
• new methods in handling business processes: (agile) business process management - BPM
• ethics and security issues.

The module will enable students to collaborate across disciplines with experts from various areas.</td></tr>
</table>

Intended Learning Outcomes

Upon completion of this module, students will be able to:

1. develop practical knowledge and management skills, and mind sets to master the challenges from an agile business environment
2. understand (routine) business processes in various context and how to adapt business processes to an agile business environment (agile Business Process Management)
3. summarize and classify the new data- and customer-driven technologies in a business context
4. understand the ideas of the "service dominant logic" as a business opportunity, such as user-centricity, value in use, value in interaction, business service ecosystems.
5. apply innovative creativity methods and processes for product and software development (Design Thinking)
6. adapt to a new working culture based on a user-centricity, empathy, and playful testing of new products and services.

Indicative Literature

Service Dominant Logic

Vargo, S.L., & Lusch, R. (2004). Evolving to a New Dominant Logic for Marketing. Journal of Marketing, Vol. 68(1), 1 – 17
Vargo SL, Akaka MA, Vaughan CM. (2017). Conceptualizing Value: A Service-ecosystem View. Journal of Creating Value. 3(2):117-124. https://doi.org/10.1177%2F2394964317732861
Lusch, R.F., Nambisan, S. (2015). Service Innovation: A Service-Dominant Logic Perspective. MIS Quarterly. Vol. 39 No.1 , pp. 155-175. https://doi.org/10.25300/MISQ/2015/39.1.07

Business Process Management and agile Management

Daniel Paschek, D., Frank Rennung, F., Trusculescu, A., Draghici,A. (2016). Corporate Development with Agile Business Process Modeling as a Key Success Factor, Procedia Computer Science, Vol 100, Pages 1168-1175, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2016.09.273.

Design Thinking

Brenner, W., Uebernickel, F., Abrell, T. (2016). Design Thinking as Mindset, Process, and Toolbox, in: Brenner, W., Uebernickel, F. (Eds.), Design Thinking for Innovation. Springer International Publishing, pp. 3–21. https://doi.org/10.1007/978-3-319-26100-3_1
Brown, T. (2008). Design Thinking. Harvard Business Review. 86, 84–92. Available at: https://hbr.org/2008/06/design-thinking

Usability and Relationship to other Modules

Examination Type: Module Examination

Assessment Type: Presentation

Duration: 30 min
Weight: 100%

Scope: All intended learning outcomes.

Completion: To pass this module, the examination has to be passed with at least 45%.

4.2 Product Innovation & Marketing

<table>
<tr><td colspan="2">Module Name
Product Innovation & Marketing</td><td>Module Code
MCSSE-MGT-02</td><td>Level (type)
Year 1</td><td>CP
5</td></tr>
<tr><td colspan="5">Module Components
Number | Name | Type | CP
MCSSE-MGT-02 | Product Innovation & Marketing | Lecture | 5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Tilo Halaszovich</td><td colspan="2">Program Affiliation

▪ MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills: ☒ None</td><td>Frequency

Annually (Spring)

Duration

1 semester</td><td>Forms of Learning and Teaching

▪ Lecture (80 hours)
▪ Private study (45 hours)

Workload

125 hours</td></tr>
<tr><td colspan="5">Recommendations for Preparation

N.A.</td></tr>
<tr><td colspan="5">Content and Educational Aims

This course focuses on key strategic aspects of the innovation and commercialization process. The course draws on insights from a variety of fields – in particular, product management, innovation, marketing, and strategic management – in order to (i) develop a holistic, state-of-the art understanding of this process, (ii) to nurture the underlying mindset that spans technology and market elements, and (iii) to provide students with concrete tools that help them in navigating the journey from product idea to market success. The course will take both the perspective of established companies as well as of new ventures.</td></tr>
<tr><td colspan="5">Intended Learning Outcomes

Upon completion of this module, students will be able to:

1. understand the innovation process, particularly in technology domains
2. understand the commercialization process, particularly in technology domains
3. analyze how value can be created and appropriated through innovation
4. understand and apply tools, methods and concepts to manage the commercialization process</td></tr>
<tr><td colspan="5">Indicative Literature

Gruber/Tal (2017). Where to Play: 3 Steps for Identifying your Most Valuable Market Opportunities, Financial Times/Pearson.
Mohr, J. et al. (2013). Marketing of high-technology products and innovations. Pearson Education.
Moore, G. A. (2014). Crossing the chasm. Harper Business.
Schilling, M.A. (2019). Strategic Management of Technological Innovation. McGraw-Hill.</td></tr>
<tr><td colspan="5">Usability and Relationship to other Modules</td></tr>
<tr><td colspan="5">Examination Type: Module Examination

Assessment Type: Presentation
Duration: 30 min
Weight: 100%

Scope: All intended learning outcomes.</td></tr>
</table>

Completion: To pass this module, the examination has to be passed with at least 45%.

4.3 Entrepreneurship & Intrapreneurship

<table>
<tr><td colspan="3">Module Name
Entrepreneurship and Intrapreneurship</td><td>Module Code
MCSSE-LAS-01</td><td>Level (type)
Year 1/2</td><td>CP
2.5</td></tr>
<tr><td colspan="6">Module Components
Number | Name | Type | CP
MCSSE-LAS-01 | Entrepreneurship and Intrapreneurship | Lecture | 2.5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Tilo Halaszovich</td><td colspan="3">Program Affiliation

• MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills: ☒ None</td><td>Frequency

Annually (Fall)

Duration
1 semester</td><td colspan="2">Forms of Learning and Teaching
▪ Lecture (17.5 hours)
▪ Private study (45 hours)

Workload
62.5 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

N.A.</td></tr>
<tr><td colspan="6">Content and Educational Aims

The module introduces students to the themes which are relevant to clearly develop corporate innovation and entrepreneurship as an activity. It introduces entrepreneurial thinking styles that are important to develop radical forms of innovation in companies. This is about a way of thinking, reasoning and acting that is opportunity obsessed and holistic in approach. It is first and foremost a process that has an intention to create, enhance, realize, and renew value, not just for owners, but for all participants and stakeholders in either a new or existing organization. Today, entrepreneurship has evolved beyond the classic start-up notion to include companies and organizations of all types, old and new; small and large; fast and slow growing; private, not-for-profit, and public.
This focus on “entrepreneurship as a process” has become a fundamental part for three main reasons. The first is the growing recognition of the critical importance of entrepreneurial activities in the economy and the society at large. As such, having an insight in the specific challenges and solutions that characterize entrepreneurship has broader implications for any 21st century graduate. The second reason is that many graduates eventually find themselves occupying a position as entrepreneur, or are associated with one as their financier, partner, supplier or customer. This requires an action-oriented approach and approaching the phenomenon from multiple angles. Finally, given the specific challenges entrepreneurs often face in terms of uncertainty and resource scarcity, solutions applied by expert entrepreneurs can be of value to any professional that finds him/herself in similar situations in organizations seeking growth, renewal or even survival.
The module focuses on the tasks and skills that entrepreneurs typically complete/use in their journey towards success. With this in mind, this module aims to provide students with insight into the approach entrepreneurs use to identify opportunities and build new ventures; the analytical skills that are needed to implement this approach; and the background knowledge and managerial skills that are needed for dealing with issues involved in starting, growing, and harnessing the value of new ventures. First and foremost, however, entrepreneurship is about action. Hence our approach is based on the primary objective of having students experience entrepreneurship.

The module assessment will consist of three presentations. Students will know in the first session which topics need to be covered in their presentations.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes

Upon completion of this module, students will be able to:

1. understand the essence of entrepreneurship
2. assess and develop a business case
3. analyse and identify new venture opportunities in a more systematic way</td></tr>
</table>

4. understand the importance of a business model for new venture creation
5. evaluate the viability of a new venture idea
6. understand how to finance a new venture
7. create and present a business case for a new venture

Indicative Literature

Clarysse, B., Kiefer, S. The Smart Entrepreneur. Elliott & Thompson, 2011.

Usability and Relationship to other Modules

Examination Type: Module Examination

Assessment Type: Presentations Duration: 30 min
Weight: 100%

Scope: All intended learning outcomes.

Completion: To pass this module, the examination has to be passed with at least 45%.

4.4 Agile Leadership and Strategic Management

<table>
<tr><td colspan="3">Module Name
Agile Leadership and Strategic Management</td><td>Module Code
MCSSE-LAS-03</td><td>Level (type)
Year 2</td><td>CP
2.5</td></tr>
<tr><td colspan="6">Module Components

Number | Name | Type | CP
MCSSE-LAS-03 | Agile Leadership and Strategic Management | Lecture | 2.5</td></tr>
<tr><td>Module Coordinator

Prof. Dr. Tilo Halaszovich</td><td colspan="3">Program Affiliation

▪ MSc Computer Science and Software Engineering (CSSE)</td><td colspan="2">Mandatory Status

Mandatory for AST and CSSE</td></tr>
<tr><td colspan="3">Entry Requirements

Pre-requisites: ☒ None
Co-requisites: ☒ None
Knowledge, Abilities, or Skills: ☒ None</td><td>Frequency

Annually (Fall)

Duration

1 semester</td><td colspan="2">Forms of Learning and Teaching

▪ Lecture (17.5 hours)
▪ Private study (45 hours)

Workload

62.5 hours</td></tr>
<tr><td colspan="6">Recommendations for Preparation

N.A.</td></tr>
<tr><td colspan="6">Content and Educational Aims

This module focuses on key strategic aspects of the leadership and strategy development processes, specifically strategic problems solving, alignment, engagement and copying with black swans and paradigm shifts. The module draws on insights from a variety of fields such as business strategy, problem solving, strategic communication, strategic planning, and strategic resilience. To build a holistic understanding, the module deals with the following topics:

• The strategic process: from analysis, definition, planning and evaluation
• Hypothesis driven problem solving
• Pyramid principle strategic communication
• Antifragile strategies

The module assessment will consist of three presentations. Students will know in the first session which topics need to be covered in their presentations.</td></tr>
<tr><td colspan="6">Intended Learning Outcomes

Upon completion of this module, students will be able to:

1. understand and analyse business strategies
2. understand and analyse strategic statements and levels of ambition
3. understand opportunities and threats on the external environment
4. evaluate sources of competitive advantage as well as strategic strengths and weaknesses
5. analyse core challenges of agile leadership and strategy development
6. develop and communicate strategic initiatives
7. apply this knowledge to real-world strategic planning processes</td></tr>
<tr><td colspan="6">Indicative Literature

Sola, D. & Couturier, J, 2013, How To Think Strategically, FT Publishing International.</td></tr>
<tr><td colspan="6">Usability and Relationship to other Modules</td></tr>
</table>

Examination Type: Module Examination

Assessment Type: Presentations Duration: 30 min
Weight: 100%

Scope: All intended learning outcomes.

Completion: To pass this module, the examination has to be passed with at least 45%.

5 Advanced Software Technology Graduate Program Regulations

5.1 Scope of These Regulations

The regulations in this handbook are valid for all students who entered the Advanced Software Technology graduate program at Constructor University in Fall 2023. In case of conflict between the regulations in this handbook and the general Policies for Master Studies, the latter apply (see https://constructor.university/student-life/student-services/university-policies/academic-policies).

In exceptional cases, certain necessary deviations from the regulations of this study handbook might occur during the course of study (e.g., change of the semester sequence, assessment type, or the teaching mode of courses).

In general, Constructor University reserves therefore the right to change or modify the regulations of the program handbook according to relevant policies and processes also after its publication at any time and in its sole discretion.

5.2 Degree

Upon successful completion of the program, students are awarded a Master of Science (M.Sc.) degree in Advanced Software Technology.

5.3 Graduation Requirements

In order to graduate, students need to obtain 120 CP. In addition, the following graduation requirements apply:

- In each module, students need to obtain a minimum amount of CP as indicated in chapter 2 of this handbook.
- Students need to complete all mandatory components of the program as indicated in chapter 2 of this handbook.

6 Appendices

6.1 Intended Learning Outcomes Assessment-Matrix

MSc Advanced Software Technology					Quality Engineering	Development ecosystem	Data Analytics	Architectural Strategy	Programming Languages on Software Development	Big Data Software Engineering	Advanced Deep Learning	Recommender Systems	Computer Vision	Machine Learning in Software Engineering	Bayesian Methods in Machine Learning	Static Program Analysis	Mobile Development	Cryptography	System Security	Distributed Ledger Technology and Smart Contracts	Network Security	Human-Computer Interaction	IDE Development	Advanced Functional Programming	Weak memory Models	Virtual Machines	Metacomputations	Dependent Types	Homotopy Type Theory	Category Theory for Programmers	Agile Product Development & Design	Product Innovation & Marketing	Agile Leadership and Strategic Management	Entrepreneurship& Intrapreneurship	Master's Thesis	Research Project	Capstone Project 1	Capstone Project 2	Capstone Project 3
Semester					1	1	1	2	2	2	1	3	1	3	1	1	1/3	1	2	3	3	1/3	1	1	1	3	2	3	3	3	1	2	3	3	4	3	1	2	3
Mandatory/ optional					m	m	m	m	m	m	me	me	me	me	me	me	me	me	me	me	me	me	me	me	me	me	me	me	me	me	m	m	m	m	m	m	m	m	m
Credits					5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	2,5	2,5	30	5	5	5	5
	Competencies*																																						
Program Learning Outcomes	**A**	**E**	**P**	**S**																																			
critically assess and creatively apply technological possibilities and innovations in the fields of data science, software development and programming languages	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x	x
critically assess and apply software engineering methodologies considering real life situations, organizations and industries	x	x			x	x	x	x	x							x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x	x
use, adapt and improve modern techniques in data science, such as deep learning, recommender systems, computer vision, and machine learning in software engineering	x	x					x			x	x	x	x	x																				x	x	x	x	x
apply cross-disciplinary management methodologies to solve academic and professional problems in the context of software development and data science	x	x	x																											x	x	x	x	x	x	x	x	x
critically assess and integrate a consistent tool set of leadership abilities into a professional work environment	x	x	x																											x	x	x	x	x	x	x	x	x
plan, conduct and document small research projects in the context of data science, software development and programming languages	x	x	x		x	x				x																				x	x	x	x	x	x	x	x	x
independently research, document and present a scientific topic with appropriate language skills	x	x	x	x																										x	x	x	x	x	x	x	x	x
use scientific methods as appropriate in the field of data science and software engineering such as defining research questions, justifying methods, collecting, assessing and interpreting relevant information, and drawing scientifically-founded conclusions that consider social, scientific and ethical insights	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
develop and advance solutions to problems and arguments in their subject area and defend these in discussions with specialists and non-specialists		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
engage ethically with academic, professional and wider communities and to actively contribute to a sustainable future, reflecting and respecting different views		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
take responsibility for their own learning, personal and professional development and role in society, evaluating critical feedback and self-analysis		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
apply their knowledge and understanding of data science, software development, and programming languages to a professional context	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x	x
take on responsibility in a diverse team		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					x	x	x	x	x
adhere to and defend ethical, scientific and professional standards	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
use and understand the Kotlin ecosystem	x					x			x						x	x	x	x	x	x	x	x				x	x	x	x					x	x	x	x	x
apply data analytics techniques	x						x			x	x	x	x	x	x																				x	x	x	x	x
understand and utilize agile product development and design methodologies		x	x	x																										x	x	x	x	x	x	x	x	x
understand and apply principles of quality engineering	x				x			x																										x	x	x	x	x
Assessment Type																																							
Oral examination																																							
Written examination						x			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x											
Project assessment																																			x	x	x	x
Project report							x																																
Practical assessment									x	x	x	x		x	x	x	x			x			x	x	x	x	x		x										
Essay																																							
Laboratory report																																							
Poster presentation																																							
Presentation																														x	x	x	x					
Portfolio assessment					x			x																															
Thesis																																		x				

*Competencies: A-scientific/academic proficiency; E-competence for qualified employment; P-development of personality; S-competence for engagement in society

Figure 3: Intended Learning Outcomes Assessment-Matrix