RSE Curriculum

Gesellschaft für Informatik deRSE Julian Dehne
Florian Goth Jan Phillip Thiele Jan Linxweiler
Anna-Lena Lamprecht Maja Toebs

1 WORK IN PROGRESS THIS IS NOT THE OFFICIAL STATEMENT OF THE COMMUNITY BUT THE CURRENT VERSION

2 Why a RSE Curriculum?

The term Research Software Engineer, or RSE, emerged a little over 10 years ago as a way to represent individuals working in the research community but focusing on software development. The term has been widely adopted and there are a number of high-level definitions of what an RSE is. However, the roles of RSEs vary depending on the institutional context they work in. At one end of the spectrum, RSE roles may look similar to a traditional research role. At the other extreme, they resemble that of a software engineer in industry. Most RSE roles inhabit the space between these two extremes.

For the purpose of creating an RSE-Master Programm we identify the RSE as a person who creates or improves research software and/or the structures that the software interacts with in the computational environment of a research domain. In this spectrum we see skilled team member who may also choose to conduct own research as part of their role. But on the other end we also see paths for an RSE to specifically focus on a technical role as an alternative to a traditional research role because they enjoy and wish to focus on the development of research software.

For this task, to support research with/in the creation of digital tools, we structure this sample curriculum along three pillars (Goth et al. 2024):

- research skills: these are competencies that enable an RSE to effectively participate in the research domain.
- technical skills: these are competencies, that enable an RSE to create effective tools for research
- communication skills: these are skills that enable an RSE to effectively work and communicate with its peers and stakeholders across multiple domains.

3 Balancing Computer Science Fundamentals with Application Demands

Research Software Engineering is a fast-growing field and the curriculum should engender the development of both experts in RSE (Fachexperten), and multidisciplinary researchers that are capable of transferring high-level software-engineering concepts to their respective domains. This can be mapped to the typology of the German Computer Science Association (GI). (Gesellschaft für Informatik e.V. (GI) 2016) defines computer science programs with a three-fold typology:

- Type 1: Computer Science Programs: Computer Science is solely responsible.
- Type 2: Computer Science Programs with a Specific Application Area: Computer Science is responsible in coordination with the participating application discipline.
- Type 3: Interdisciplinary Programs with a Computer Science Component Equal in Weight to Other Participating Disciplines: Computer Science shares responsibility jointly with the participating disciplines.

Fully qualified computer science experts with a focus on research software fit type 1 of CS programs. Interdisciplinary researchers where the computer science background is rivalled by the domain expertise fit the type 3.

(At this point) the curriculum tries to support both aspects equally with two branches (or tracks) of the curriculum focussing on the different weights. However, both tracks share common modules and concepts such as the idea to teach RSE-specific Open Science Tooling, an advanced Software Engineering module with specific patterns and modelling techniques for RSE and dual-lab and dual-thesis ideas, where RSE-students interface with a domain or industry field to apply their cross-cutting research skills.

You can find the tracks as follows:

- Computer Science Generalist Track here
- Natural Science / Domain Science Track here

The RSE community has long used the three pillar structure for RSE specific skills. These are distributed in the modules as following:

3.1 Research skills

Implemented in the following components:

- Domain Science Project
- Domain Science Wildcard Courses
- RSE Thesis

3.2 Technical skills

Implemented in:

- RSE Nuts and Bolts
- Data Science Foundations
- Programming Foundations
- Software Engineering Foundations
- RSE Software Engineering
- Scientific Computing Basics and High Performance Computing

3.3 Communication skills

Implemented in:

- RSE Management
- Domain Science Project or RSE Lab Project
- RSE Philosophy
- RSE Lecture Series

4 Possible Job Roles for an RSE

4.1 Open Science RSE

Open science and FAIRness of data and software are increasingly important topics in research, as exemplified by the demand of an increasing amount of research funding agencies requiring openness. Hence, an Open Science RSE is required to have a deeper knowledge in **Research Culture (RC)** and how to distribute software publicly (**Software Reusability (SRU)**, **Software Publication (SP)**). Open Science RSEs can help researchers navigate the technical questions that come up when practising Open Science, such as:

- "How do I make my code presentable?"
- "How do I make my code citable?"
- "What do I need to do to make my software FAIR?"
- "How do I sustainably work with an (international) team on a large code base?"

Like the Data-focused RSE, they have a deep understanding of **Research Data Management (RDM)** topics.

4.2 Project/Community Manager RSEs

When research software projects become larger, they need someone who manages processes and people. In practice, this concerns change management for code and documentation, and community work to safeguard usability and adaptability, but also handling project governance and scalable decision-making processes. This gap can be filled by people who invest in the **Project Management (PM)**, **User Support (USERS)**, and **Team Management (TEAM)** skills.

Building a community around a research project is an important building block in building sustainable software (Segal 2009), so these RSEs play an important role, even if they do not necessarily touch much of the code themselves.

4.3 Teaching RSEs

RSEs interested in developing their **Teaching (TEACH)** skill can focus on teaching the next generation of researchers and/or RSEs and will play a vital role in improving the quality of research software. They need to have a good understanding of all RSE competencies relevant to their domain and additionally should have experience or training in the educational field.

4.4 User Interface/User Experience Designers for Research Software

Scientific software is a complex product that often needs to be refined in order to be usable even by other scientists. To facilitate this, there are people required that specialise in the **Documentation & Best Practices (DOCBB)** and probably the **Distribution (DIST)** competency with a focus on making end-user-facing software really reusable and hence FAIR. This task is supported by strong **Modelling (MOD)** skills to reason about the behaviour of potential users of the software.

5 Track 1: Computer Science Generalist Track

5.1 Suggested Study Plan

Type	Description	SWS	Sem 1	Sem 2	Sem 3	Sem 4
Lecture	RSE Lecture Series	2	3	0	0	0
Seminar	RSE Nuts and Bolts I	2	3	0	0	0
Lecture	RSE Management Lecture	2	2	0	0	0
Exercise	RSE Management Exercise	2	2	0	0	0
Lecture	Statistical Data Analysis	4	6	0	0	0
Exercise	Statistical Data Analysis Exercise	2	3	0	0	0
Lecture	Mathematical Foundations of Data Science	2	2	0	0	0
Exercise	Mathematical Foundations of Data Science Exercise	2	4	0	0	0
Seminar	RSE Philosophy	2	3	0	0	0
Lab	RSE Lab Project	8	0	12	0	0
Lecture	Scientific Computing Basics	2	0	3	0	0
Exercise	Scientific Computing Basics Exercise	2	0	3	0	0
Lecture	Distributed Systems	2	0	2	0	0
Exercise	Distributed Systems Exercise	2	0	4	0	0
Lecture	Efficient Algorithms	2	0	2	0	0
Exercise	Efficient Algorithms Exercise	2	0	4	0	0
Seminar	RSE Nuts and Bolts II	2	0	0	3	0
Lecture	Text2Data	2	0	0	4	0
Exercise	Text2Data Exercise	2	0	0	2	0
Lecture	High Performance Computing	2	0	0	3	0
Exercise	High Performance Computing Exercise	2	0	0	3	0
Lecture	Security and Cryptography	2	0	0	2	0
Exercise	Security and Cryptography Exercise	2	0	0	4	0

Type	Description	SWS	Sem 1	Sem 2	Sem 3	Sem 4
Seminar	Current topics in artificial intelligence	2	0	0	3	0
Seminar	RSE Software Engineering	2	0	0	3	0
Exercise	RSE Software Engineering Exercise	2	0	0	3	0
Thesis	RSE Master Thesis	0	0	0	0	30
	Total ECTS per semester		28	30	30	30

Total Curriculum ECTS: 118

5.2 RSE Nuts and Bolts

5.2.1 Introduction

This module, inspired by the MIT Missing Semester, addresses the "nuts and bolts" often missing from traditional academic training in computing. It aims to provide students with practical skills and conceptual understanding for building robust, maintainable, and reproducible research software—key competencies in Research Software Engineering (RSE).

5.2.2 General Competencies

The module begins with general-purpose computing tools and techniques that are foundational for any research software engineer:

- Shell tools and scripting
- Command-line environments
- Editors and IDEs (e.g., Vim)
- Version control (Git)
- Data wrangling
- Debugging and profiling
- Metaprogramming
- Security and cryptography

5.2.3 RSE-Specific Topics

Building on these foundations, the module introduces RSE-specific concepts and good practices:

- **Version control and collaboration**
  - Git for code history, collaboration, and issue tracking
- **Virtualization concepts**
  - Containerization and environment management
- **The Data Life Cycle**
  - Managing research data and understanding data provenance
- **Good coding practices**
  - Reproducible and testable code
  - Meaningful documentation and error messages
  - Modular software design
  - Performance-conscious coding
  - Easily installable and distributable software
  - Coding standards, formatting, and linting
- **Software management planning**
  - Writing Data and Software Management Plans
  - Sustainable development and community involvement
- **Low-level programming**
  - Introduction to a compiled language (e.g., C) to expose hardware-level concerns and efficient memory management
- **Long-term software maintenance**
  - Version tracking, bug management, and sustainability strategies
  - Building and maintaining research software communities

5.2.4 Beyond the Basics

Finally, the module touches on practices that support the scholarly nature of research software:

- Software publication and citation (see SP in (Goth et al. 2024))
- Use of domain-specific repositories and registries (see DOMREP in (Goth et al. 2024))

By the end of this module, students will be well-equipped to design, develop, document, and maintain research software that meets high standards of quality, sustainability, and reproducibility.

The module is made up of two seminars that the students take at different stages in their master's program: In the first seminar during their first semester, students mainly learn new

concepts and get to know essential tools, whereas the second seminar in the third semester focuses on teaching others about research software and the development process of it (see TEACH in (Goth et al. 2024)).

Seminar: RSE Nuts and Bolts I This is an introductory class to essential techniques an RSE needs in everyday life. **SWS:** 2 **ECTS:** 3

Seminar: RSE Nuts and Bolts II This is an advanced class of RSE techniques that includes a teaching component as part of the preparation for working as an RSE in interdisciplinary teams. **SWS:** 2 **ECTS:** 3

5.2.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_1	Use literate programming tools (e.g., Quarto, Marimo, Pluto.jl, Jupyter) to combine code, results, and narrative	Research Software Engineering		Submit a literate notebook or document integrating code, visualizations, and explanatory text	Workshop Participants	Link
rse_tooling_2	Use Python for visualization, scripting, templating, and integration tasks	Research Software Engineering		Submit a Python project demonstrating use of libraries for visualisation, web tasks, and templating	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_3	Write and use Bash scripts for automation	Research Software Engineering		Submit shell scripts automating file manipulation or computational workflows	Workshop Participants	Link
rse_tooling_4	Apply testing, debugging, and logging techniques to ensure software reliability	Research Software Engineering	rse_tooling_2	Submit logs, test cases, and debugging documentation for a non-trivial Python or Bash project	Workshop Participants	Link
rse_tooling_5	Use workflow management tools (e.g., CWL, Nextflow) to design scalable, reproducible pipelines	Research Software Engineering	rse_tooling_3, rse_tooling_11	Submit a reproducible workflow including metadata and input/output definitions	Workshop Participants	Link
rse_tooling_6	Estimate resource requirements for computational tasks using profiling and benchmarking	Research Software Engineering	rse_tooling_2, rse_tooling_5	Provide resource usage profiles and discuss optimization implications	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_7	Use package managers and virtual environments (e.g., conda, nix) to manage software dependencies	Research Software Engineering		Submit environment definitions and reproducible setup instructions for a project	Workshop Participants	Link
rse_tooling_8	Document and package software for usability and reusability, using generators and modular design	Research Software Engineering	rse_tooling_2	Submit user and developer documentation generated with Sphinx or similar, plus a reusable code module	Workshop Participants	Link
rse_tooling_9	Communicate technical RSE topics effectively with non-technical audiences	Research Software Engineering		Prepare and deliver a presentation or write an article explaining RSE concepts to a general audience	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_10	Apply authentication and authorization mechanisms (e.g., LDAP, ACLs, Active Directory)	Research Software Engineering		Configure and demonstrate access control for a multi-user service or application	Workshop Participants	Link
rse_tooling_11	Make informed decisions about tooling and infrastructure (e.g., Jupyter vs scripts, local vs HPC/cloud)	Research Software Engineering	rse_tooling_1, rse_tooling_2, rse_tooling_3	Submit a comparative analysis justifying tooling and infrastructure choices for a research project	Workshop Participants	Link
rse_tooling_12	Teach and practice collaborative development, including version control and code review	Research Software Engineering	rse_tooling_2	Submit a project with version history and documented code reviews	Workshop Participants	Link
rse_tooling_13	Mentor others in research software engineering practices	Research Software Engineering	rse_tooling_12	Document a mentoring session, workshop, or support activity	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_14	Deploy and maintain web servers for research applications	Research Software Engineering	rse_tooling_2	Deploy a working web application with setup and maintenance documentation	Workshop Participants	Link
rse_tooling_15	Understand and manage file systems, including local and network-attached storage	Research Software Engineering		Document storage strategies and access mechanisms in a real-world setup	Workshop Participants	Link

5.2.6 Sources & Implementations:

5.2.6.1 Courses

- MIT Missing Semester
- CodeRefinery
- INTERSECT Training Materials
- Digital Research Academy Materials (Git, HPC, Reproducibility, Research Software)
- Building Better Research Software (SSI)
- Docker for neuroscience (jupyter book)

5.3 Classical Software Engineering

To summarise the vast range of the skills a software engineer is typically equipped with, we refer to the Guide to the Software Engineering Body of Knowledge (Bourque, Fairley, and IEEE Computer Society 2014). Because research software engineering is an interface discipline, RSEs are often stronger in topics more commonly encountered in research software contexts (e.g., mathematical and engineering foundations) than in other areas (e.g., software engineering

economics). However, they bring a solid level of competence in all software engineering topics. Therefore, RSEs can set and analyse software requirements in the context of open-ended, question-driven research. They can design software so that it can sustainably grow, often in an environment of rapid turnover of contributors. They are competent in implementing solutions themselves in a wide range of technologies fit for different scientific applications. They can formulate and implement various types of tests, they can independently maintain software and automate operations of the integration and release process. They can provide working, scalable, and future-proof solutions in a professional context and with common project and software management techniques, adapted to the needs of the research environment. Finally, as people who have often gained significant research experience in a particular discipline, they combine the necessary foundations from their domain with software engineering skills to develop complex software.(Goth et al. 2024)

This module tries to lay the foundations for the advanced RSE software engineering training.

Bourque, Pierre, Richard E. Fairley, and IEEE Computer Society. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd ed. Washington, DC, USA: IEEE Computer Society Press.

Gesellschaft für Informatik e.V. (GI). 2016. "Empfehlungen Für Bachelor- Und Masterprogramme Im Studienfach Informatik an Hochschulen." GI-Empfehlungen.

Goth, F, R Alves, M Braun, LJ Castro, G Chourdakis, S Christ, J Cohen, et al. 2024. "Foundational Competencies and Responsibilities of a Research Software Engineer [Version 1; Peer Review: Awaiting Peer Review]." *F1000Research* 13 (1429). https://doi.org/10.12688/f1000research.157778.1.

Segal, Judith. 2009. "Some Challenges Facing Software Engineers Developing Software for Scientists." In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE. https://doi.org/10.1109/secse.2009.5069156.

5.3.1 Software Engineering I

Basic concepts of software engineering, software and product life cycle, process models for the design of large software systems, semantic aspects of domain description, hierarchy, parallelism, real-time and embedded systems as fundamental paradigms, organizational principles of complex software systems, design by contract, patterns in modeling and design methods of quality assurance, evolution and re-engineering, selected languages and tools for process- and object-oriented modeling, methods and languages for object-oriented design, architectures and architectural patterns of software systems, architecture of enterprise applications, design and implementation models in the object-oriented paradigm, e.g., Java 2 SE, design patterns, software testing methods.

Lecture: Software Engineering I

SWS: 2 **ECTS:** 4

Exercise: Software Engineering I Exercise

SWS: 2 **ECTS:** 2

5.3.2 Software Engineering 2

The module covers a selection of advanced topics in the field of software engineering, such as software quality assurance, service engineering, virtualization, programming languages and design, and formal methods in system design.

Lecture: Software Engineering II

SWS: 2 **ECTS:** 2

Exercise: Software Engineering II Exercise

SWS: 2 **ECTS:** 4

5.3.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_programming_1	Understand the fundamental concepts of software engineering	Computer Science		Demonstrate understanding through theoretical assessments and practical examples	University of Potsdam	Link
gen_programming_2	Apply various approaches of software engineering	Computer Science	gen_programming_1	Complete assignments or projects using different software engineering methods	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_programming_3	Identify and utilize essential technologies and tools for specification, component-based development, and quality assurance of modern software systems	Computer Science	gen_programming_1	Work with selected tools and technologies in practical exercises and case studies	University of Potsdam	Link
gen_programming_4	Demonstrate an in-depth understanding and ability to apply various approaches of software engineering	Computer Science	gen_programming_1, gen_programming_2	Successfully complete advanced projects employing different software engineering methods	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_programming_5	Understand the characteristics of a wide range of technologies and tools for specification, component-based development, and quality assurance of modern software systems, and apply them in various contexts	Computer Science	gen_programming_3	Apply appropriate technologies and tools in complex case studies and demonstrate their use in different application scenarios	University of Potsdam	Link

5.3.4 Sources & Implementations:

5.3.4.1 Curricula

- Computing Curricula 2020

5.3.4.2 Courses

- Software Engineering I
- Software Engineering
- Softwaretechnik

5.3.4.3 Programs

- UP Informatik/Computational Science Bachelor
- JMUW Informatik Bachelor
- RWTH Informatik Bachelor

5.4 Computer Vision and Image Processing

5.4.1 Introduction

The course imparts knowledge about current methods and algorithms in the field of computer vision. Important fundamentals and the latest approaches to image representation, image processing and image analysis are taught. Current models and methods of machine learning and their technical background are presented, and their respective applications in image processing are demonstrated.

5.4.2 Contents

- fundamental methods and techniques that enable a machine to analyse images and videos and understand their content
- fundamentals of image generation, linear filters, image segmentation, object recognition, object categorisation, 3D reconstructio
- application of current methods of machine learning for the topics described above

5.4.3 Learning Objectives

- Students describe current research trends and developments in the field of computer vision
- Students name relevant techniques necessary for image and video analysis tasks
- Students will be able to derive and explain methods and techniques that enable a machine to analyse images and videos and understand their content
- Students select basic computer vision techniques that are necessary for these analysis tasks
- Students independently apply the methods covered to real-world problems
- Students implement the algorithms presented themselves and translate them into a programming language of their choice

5.4.4 Examination Methods

- either a written exam (90-100 minutes)
- or an oral examination (20-30 minutes)
- or a project report (20-30 pages)

Lecture: Computer Vision

SWS: 2 **ECTS:** 2

Exercise: Computer Vision Exercise

SWS: 2 **ECTS:** 4

5.4.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
compiler_tech_1	apply modelling techniques	Computer Science		develop solutions through technical methods	University of Potsdam	Link
compiler_tech_2	analyse problems in given software systems	Computer Science		discuss problems in a team	University of Potsdam	Link

5.4.6 Sources & Implementations:

5.4.6.1 Curricula

- None

5.4.6.2 Courses

- Image Processing and Phenotyping in Bioinformatics
- Computer Vision
- Computer Vision
- Computer Vision 2

5.4.6.3 Recommended Course Literature

- None

5.4.6.4 Programs

- UP Computational Science Master
- RWTH Informatik Master
- JMUW Informatik Master

5.5 Fundamentals of Computer Science

This module covers the basics of computer science and introduces the students from domain sciences to programming.

5.5.1 Basics of Computer Science

5.5.1.1 Basic Concepts

- Introduction to computer science, basic concepts of operating systems using UNIX/Linux as an example
- From problem to algorithm: concept of an algorithm, design of algorithms, pseudocode, refinement, brute-force algorithms, models and modeling, graphs and their representation, simple algorithms on graphs, analysis of algorithms (correctness, termination, runtime)
- Implementation of algorithms (e.g., using Python)
- Programming paradigms: procedural, object-oriented, and functional programming; recursion versus iteration
- From program to process: assembly languages, assembler, compiler, interpreter, syntax and semantics of programming languages
- Limits of algorithms: computability, decidability, undecidability

Lecture: Basic Programming

SWS: 2 **ECTS:** 2

Exercise: Basic Programming Exercise

SWS: 2 **ECTS:** 4

5.5.2 Applied Programming

5.5.2.1 Procedural Programming Concepts

Programming with an imperative-procedural language (such as C):

- Data types, type casting, control structures, functions and procedures, parameter passing paradigms, call stack
- Pointers, arrays, strings, structured types
- Errors and their handling
- Dynamic memory management
- Program libraries

5.5.2.2 Programming in an Object-Oriented Language (e.g., Java)

- Classes, objects, constructors
- Inheritance, polymorphism, abstract classes/interfaces
- Exceptions and exception handling
- Namespaces (packages)
- Generic classes and types
- Program libraries

Lecture: Applied Programming

SWS: 2 **ECTS:** 2

Exercise: Applied Programming Exercise

SWS: 2 **ECTS:** 4

5.5.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_1	Use an imperative-procedural programming language (e.g., C) and an object-oriented language (e.g., Java) with confidence	Computer Science		Submit working programs in both languages demonstrating syntax and language-specific features	University of Potsdam	Link
ex_programming_mod1_2	Implement basic data structures and algorithms	Computer Science	ex_programming_mod1_1	Submit a project with implemented algorithms and data structures (e.g., lists, trees, sorting)	University of Potsdam	Link
ex_programming_mod1_3	Distinguish between error types and handle them appropriately in code	Computer Science	ex_programming_mod1_1	Demonstrate error handling techniques in submitted code (e.g., input validation, error codes, exceptions)	University of Potsdam	Link
ex_programming_mod1_4	Identify and use appropriate library functions in programming tasks	Computer Science	ex_programming_mod1_1	Integrate external libraries in coding tasks and document their usage	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_5	Use basic functions and mechanisms of operating systems using UNIX/Linux as an example	Computer Science		Demonstrate file handling, permissions, and process control using UNIX/Linux commands	University of Potsdam	Link
ex_programming_mod1_6	Create and refine simple algorithms using semi-formal notation	Computer Science		Submit pseudocode or flowcharts for given algorithmic problems	University of Potsdam	Link
ex_programming_mod1_7	Evaluate and compare algorithms using runtime analysis	Computer Science	ex_programming_mod1_6	Provide time complexity comparisons for multiple algorithmic solutions	University of Potsdam	Link
ex_programming_mod1_8	Implement simple algorithms using imperative and functional programming styles (e.g., in Python)	Computer Science	ex_programming_mod1_6	Submit code demonstrating both imperative and functional styles for the same problem	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_9	Distinguish between programming paradigms and identify their characteristics	Computer Science	ex_programming_mod1_1	Classify given code snippets by paradigm and justify the classification	University of Potsdam	Link
C10	Express simple programs in an assembly language	Computer Science		Translate simple high-level logic into assembler code	University of Potsdam	Link
C11	Discuss the limits of algorithms, including computability and decidability	Computer Science		Write a short essay or present on concepts such as the Halting Problem or undecidability	University of Potsdam	Link

5.5.4 Sources & Implementations:

5.5.4.1 Curricula

- Computing Curricula 2020

5.5.4.2 Courses

- Grundlagen der Programmierung
- Praxis der Programmierung
- Python for Psychologists
- Grundlagen der Informatik

- Grundlagen der Programmierung
- Programmierung

5.5.4.3 Programs

- UP Informatik/Computational Science Bachelor
- JMUW Informatik Bachelor
- RWTH Informatik Bachelor

5.6 Declarative Programming

5.6.1 Introduction

This module deals with delarative modelling, problem-solving and programming. It can focus on the functional or logial programming paradigm.

5.6.2 Contents

- Logical-relational programming paradigm, top-down evaluation with SLD(NF) resolution. Introduction to the logical programming language Prolog: recursion, predicate-oriented programming, backtracking and cut, side effects, aggregations. Connection to (deductive) databases. Comparison with Datalog and brief introduction to advanced concepts such as constraint logic programming.
- Predicate logic fundamentals: Unification, Resolution, Horn clauses and SLD resolution
- Logic programmes: Operational and denotational semantics, Evaluation strategies
- The Prolog programming language: Negation as failure, Non-logical components of Prolog, Programming techniques
- Applications and extensions of logic programming OR
- Introduction to the Haskell programming language: Syntax of the various language constructs, Higher-order functions, Programming with lazy evaluation
- Denotational semantics of functional programmes: Complete orders and fixed points, Denotational semantics of Haskell
- Lambda calculus: Syntax and operational semantics of lambda calculus, Reduction of Haskell to lambda calculus
- Type checking and type inference

5.6.3 Learning Objectives

Students - have a broad, detailed and critical understanding of the latest knowledge in selected specialist areas in the field of declarative modelling - can implement compact and declarative programmes in Prolog and distinguish this approach from classical imperative programming - formally define the semantics of logical programming languages - targeted use of logical programming languages in various areas of application OR Students - explain the concepts underlying functional programming languages - learn and apply programming techniques in functional languages - formally define the semantics of functional programming languages - implement functional languages - design type checking procedures for functional languages - use functional languages in a targeted manner in various areas of application

5.6.4 Examination Methods

60-120 min written exam or 30 minute oral examination.

Lecture: Declarative Programming

SWS: 2 **ECTS:** 2

Exercise: Declarative Modelling Exercise

SWS: 2 **ECTS:** 4

5.6.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
declarative_model	define and interpret the special features, limitations, terminology and doctrines in the field of declarative modelling	Computer Science		develop declarative models for new problems	University of Potsdam	Link

5.6.6 Sources & Implementations:

5.6.6.1 Curricula

- None

5.6.6.2 Courses

- Deklarative Modellierung
- Deklarative Problemlösung und Optimierung
- Logische Programmierung
- Foundations of Functional Programming
- Foundations of Logic Programming

5.6.6.3 Recommended Course Literature

- None

5.6.6.4 Programs

- UP Computational Science Master
- JMUW Informatik Master
- RWTH Informatik Master

5.7 Design of Efficient Algorithms

5.7.1 Introduction

This module deals with efficient algorithms, optimisation, and approximation algorithms.

5.7.2 Contents

Architecture and implementation of formal systems for the design of efficient algorithms, correctness-preserving optimisation, programme verification and synthesis. Design and analysis of approximation algorithms.

5.7.3 Learning Objectives

Students will be familiar with and understand basic techniques of formal programme development and its implementation. Students will be able to anticipate efficiency problems when designing algorithms and respond accordingly. Students will write better programmes that are highly optimised in their runtime-critical areas. Students will be able to analyse simple approximation methods in terms of their quality. They understand basic design techniques such as greedy, local search, scaling and methods based on linear programming and can also apply these to new problems.

5.7.4 Examination Methods

- Either a 90-minute written exam.
- Or a 30 minute oral examination.

Lecture: Efficient Algorithms

SWS: 2 **ECTS:** 2

Exercise: Efficient Algorithms Exercise

SWS: 2 **ECTS:** 4

5.7.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
eff_algorithms_1	describe formal systems for the development of efficient algorithms and correctness-preserving optimisation	Computer Science		exemplary application of those formal methods	University of Potsdam	Link

5.7.6 Sources & Implementations:

5.7.6.1 Curricula

- None

5.7.6.2 Courses

- Entwurf effizienter Algorithmen
- Analyse von Algorithmen
- Approximationsalgorithmen

5.7.6.3 Recommended Course Literature

- None

5.7.6.4 Programs

- UP Computational Science Master
- RWTH Informatik Master
- JMUW Informatik Master

5.8 Current Topics in Artificial Intelligence

5.8.1 Introduction

This module deals with recent research in artificial intelligence.

5.8.2 Contents

The seminar addresses current research questions in the field of artificial intelligence.

5.8.3 Learning Objectives

Students - are able to define and interpret the special features, limitations, terminology and doctrines in the field of artificial intelligence. The knowledge and understanding gained by students forms the basis for the development and/or application of independent ideas in the field of artificial intelligence in a research-oriented context - have a broad, detailed and critical understanding of the latest state of knowledge in selected specialist areas in the field of artificial intelligence - are able to apply their knowledge, understanding and problem-solving skills in new and unfamiliar situations that are related to the field of artificial intelligence in a broader or multidisciplinary context.

5.8.4 Examination Methods

- 30 minute presentation with discussion

Seminar: Current topics in artificial intelligence

SWS: 2 **ECTS:** 3

5.8.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ai_1	describe and differentiate current research topics in AI	Computer Science		answer questions about those research topics	University of Potsdam	Link

5.8.6 Sources & Implementations:

5.8.6.1 Curricula

- None

5.8.6.2 Courses

- Aktuelle Themen der Künstlichen Intelligenz
- Aktuelle Themen der Informatik

5.8.6.3 Recommended Course Literature

- None

5.8.6.4 Programs

- UP Computational Science Master
- JMUW Informatik Master

5.9 Distributed Systems

5.9.1 Introduction

This module deals with distributed IT systems and their performance.

5.9.2 Contents

The module covers a selection of the following topics:

- Reliability of distributed systems: Concepts of distributed file systems, synchronization techniques for reliable distributed applications, concepts of load balancing in high-availability clusters,
- Example: Sensor networks: Routing in sensor networks, operating systems for sensor networks, security in sensor networks,
- Secure internet protocols (IP security (IPsec), Pretty Good Privacy (PGP), Secure Socket Layer (SSL), Transport Layer Security (TLS), Secure Shell (SSH), DNS security (DNSsec)), secure IPv6 networks.
- example: internet of things
- Traffic theory models, basic concepts of probability theory, transformation methods, stochastic processes, methodology for performance analysis of technical systems, queueing and traffic theory discrete-time and continuous-time Markov chains, analysis of Markov and non-Markov systems, application examples for performance analysis of current computer systems and networks: service quality and other characteristics
- Algorithms in message-passing systems
- Leader election and consensus
- Routing in networks: centralized and distributed approaches
- Randomized methods for contention resolution and congestion avoidance
- Contagion and Distributed Network Dynamics

5.9.3 Learning Objectives

Students - can evaluate existing distributed systems in terms of reliability and security and identify vulnerabilities. - can correctly identify reliability and security requirements when designing new distributed systems and consider them early in the development process. - have the methodological knowledge and practical skills to model technical systems using probability theory and mathematical statistics. - describe foundational problems arising in distributed systems and explain algorithmic solutions for these problems. - apply general algorithmic design principles like randomized contention resolution and congestion avoidance and to use techniques like load balancing and randomization to solve problems arising in network contexts. - model distributed systems in a formal way and to develop algorithmic solutions enabling the efficient usage of computer networks and other distributed systems

5.9.4 Examination Methods

- either 120 min written exam
- or 20-30 min oral examination

Lecture: Distributed Systems

SWS: 2 **ECTS:** 2

Exercise: Distributed Systems Exercise

SWS: 2 **ECTS:** 4

5.9.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
dist_systems_1	evaluate existing distributed systems in terms of reliability and security and identify vulnerabilities	Computer Science		Submit written analysis of existing distributed systems	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
dist_systems_2	identify reliability and security requirements when designing new distributed system and consider them early in the development process	Computer Science		discuss necessary requirements for a design of a distributed system	University of Potsdam	Link

5.9.6 Sources & Implementations:

5.9.6.1 Curricula

- None

5.9.6.2 Courses

- Verteilte Systeme
- Theory of Distributed Systems
- Leistungsbewertung verteilter Systeme

5.9.6.3 Recommended Course Literature

- None

5.9.6.4 Programs

- UP Computational Science Master
- RWTH Informatik Master
- JMUW Informatik Master

5.10 RSE-Software Engineering

This module extends the Classical Software Engineering Module with research specific learnings. This includes but is not limited to

- software re-use (see SRU in (Goth et al. 2024))
- creating documented code building blocks (see DOCBB in (Goth et al. 2024))
- building distributable software (see DIST in (Goth et al. 2024))
- research specific programming languages
- research specific code requirements (scalability, functional programming, …)
- Adapting the software life cycle to research (see SWLC in (Goth et al. 2024))
- Software behaviour awareness and analysis (see MOD in (Goth et al. 2024))
- Research specific Engineering Patterns

Also, the seminar provides room for reflection and discussions of SE lab experiences.

Seminar: RSE Software Engineering

SWS: 2 **ECTS:** 3

Exercise: RSE Software Engineering Exercise

SWS: 2 **ECTS:** 3

5.10.1 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_1	Apply good coding practices including formatting, linting, and modular design	Research Software Engineering	rse_tooling_2	Submit a code project demonstrating modularity, consistent formatting, and use of linters	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_2	Write code and documentation that supports reproducibility in research	Research Software Engineering	rse_tooling_1, rse_tooling_4	Submit a project with data, software, and instructions allowing full reproduction of results	RSE Curriculum Draft	Link
rse_practices_3	Organise files and name code artifacts using clear, consistent conventions	Research Software Engineering		Submit a software repository with a structured layout and consistent naming scheme	RSE Curriculum Draft	Link
rse_practices_4	Version control code and collaborate using platforms like GitHub or GitLab	Research Software Engineering	rse_tooling_12	Participate in a collaborative coding project using Git-based workflows and merge requests	RSE Curriculum Draft	Link
rse_practices_5	Write effective documentation and user-facing error messages	Research Software Engineering	rse_tooling_8	Provide documentation and example error handling demonstrating clarity and user support	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_12	Write performant code suitable for use in compute-intensive contexts	Research Software Engineering	rse_tooling_2	Submit benchmark results comparing an optimized version of code with a naive implementation	RSE Curriculum Draft	Link
rse_practices_13	Publish code and software in trusted repositories and package managers	Research Software Engineering	rse_practices_6	Publish software to a repository (e.g., GitHub, PyPI, CRAN) and register it with a long-term archive (e.g., Zenodo)	RSE Curriculum Draft	Link
rse_practices_6	Apply licensing and publishing strategies to make software reusable and citable	Research Software Engineering		Submit a software project with an appropriate open license and published DOI (e.g., via Zenodo)	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_7	Apply principles of Open Source and FAIR (Findable, Accessible, Interoperable, Reusable) software	Research Software Engineering	rse_practices_6	Review or create a software project and evaluate its compliance with FAIR/Open Source principles	RSE Curriculum Draft	Link
rse_practices_14	Manage data within a software project in accordance with best practices	Research Software Engineering, Data Management		Submit a data-driven project showing clear data organisation, metadata, and reproducibility	RSE Curriculum Draft	Link

5.10.2 Sources & Implementations:

5.10.2.1 Courses

- Research Software Engineering

5.10.2.2 Programs

- UP Computational Science Master

5.11 RSE Computing

RSEs with expertise in HPC and other performance-critical computing domains specialize in optimizing code for efficient execution across various platforms, including clusters, cloud,

edge, and embedded systems. They understand parallel programming models, hardware-specific optimizations, profiling tools, and platform constraints such as memory, energy, and latency. Their skills enable them to adapt software to diverse infrastructures, manage complex dependencies, and support researchers in accessing and using advanced computing resources effectively and sustainably.

5.11.1 Basic Scientific Computing

5.11.1.1 Module Overview

This module provides an entry-level yet rigorous foundation in scientific computing for graduate students and researchers who need to **design, implement, and evaluate computational experiments**. Learners gain an awareness of the numerical underpinnings of modern simulation and data-driven research, with an emphasis on writing *reproducible, efficient, and trustworthy* code.

5.11.1.2 Intended Learning Outcomes

By the end of the module participants will be able to

1. Benchmark small programs and interpret performance metrics in a research context.
2. Explain how approximation theory and floating-point arithmetic affect numerical accuracy and stability.
3. Identify when to use established simulation libraries (e.g. BLAS/LAPACK, PETSc, Trilinos) instead of custom code.
4. Write simple GPU kernels and describe the core principles of accelerator programming.
5. Submit and monitor batch & array jobs on a mid-size compute cluster.
6. Describe common HPC challenges—such as I/O bottlenecks, threading, and NUMA—and propose mitigation strategies.
7. Maintain research software through continuous benchmarking.

5.11.1.3 Syllabus (Indicative Content)

Week	Theme	Topics
1	**Benchmarking & Profiling**	Timing strategies · micro vs. macro benchmarks · tooling overview
2	**Precision & Approximation**	IEEE-754 recap · conditioning & stability · error propagation
3	**Scientific Libraries**	BLAS/LAPACK anatomy · hierarchical I/O libraries · overview of PETSc/Trilinos/Hypre

Week	Theme	Topics
4	**GPU Primer**	Kernel model · memory hierarchy · CUDA/OpenCL/PyTorch lightning intro
5	**Working on a Cluster**	Slurm basics · job arrays · job dependencies · simple Bash launchers
6	**HPC Pitfalls**	I/O throughput · thread oversubscription · NUMA awareness
7	**Software Maintenance**	Regression + performance tests · continuous benchmarking pipelines

5.11.1.4 Teaching & Learning Methods

Short lectures (30%) are coupled with hands-on labs (70%). Students complete **weekly notebooks** and a **mini-project** that reproduces and optimises a published computational result.

5.11.1.5 Assessment

Component	Weight	Details
Continuous labs	40%	Weekly graded notebooks
Final mini-project	60%	Report, code, and benchmark suite

5.11.1.6 Prerequisites

- Basic programming in Python, C/C++, or Julia
- Undergraduate calculus & linear algebra

5.11.1.7 Key Resources

ChatGPT fantasy

Lecture: Scientific Computing Basics

SWS: 2 **ECTS:** 3

Exercise: Scientific Computing Basics Exercise

SWS: 2 **ECTS:** 3

5.11.2 High-Performance Scientific Computing

5.11.2.1 Module Overview

Building on "Basic Scientific Computing", this module dives deeper into **scalable algorithms, architectures, and software engineering techniques** required to run large-scale simulations and data analysis on high-performance computing (HPC) systems.

5.11.2.2 Intended Learning Outcomes

Participants who successfully complete the module will be able to

1. Classify scientific problems by their dominant parallel pattern (memory-parallel, compute-parallel, task-parallel).
2. Map each class to appropriate numerical libraries and hardware architectures.
3. Analyse floating-point and algorithmic approximation errors at scale.
4. Explain modern HPC hardware features (GPUs, SIMD/AVX, NUMA, high-speed interconnects) and select relevant optimisation strategies.
5. Design portable, performance-portable code employing MPI, OpenMP, and accelerator frameworks.
6. Use continuous benchmarking to guide sustainable performance evolution of research software.
7. Plan for long-term maintenance, archival, and FAIR publication of large-scale codes and data.

5.11.2.3 Syllabus (Indicative Content)

Week	Theme	Topics
1	**Parallel Problem Taxonomy**	Sparse vs. dense linear algebra · embarrassingly parallel workloads
2	**Distributed Memory (MPI)**	Domain decomposition · halo exchange · scalability metrics
3	**Shared Memory & SIMD**	OpenMP · threading pitfalls · AVX intrinsics
4	**Accelerator Programming**	Multi-GPU kernels · unified memory · portability layers (Kokkos, SYCL)
5	**Advanced I/O & Checkpointing**	Parallel file systems · burst buffers · HDF5/ADIOS-based workflows
6	**Performance Engineering**	Roofline model · continuous & comparative benchmarking · autotuning

Week	Theme	Topics
7	**Sustainable HPC Software**	Release engineering · long-term archiving · community governance

5.11.2.4 Teaching & Learning Methods

Blended delivery: interactive lectures (40%), coding workshops on the national cluster (50%), expert seminars (10%).

5.11.2.5 Assessment

Component	Weight	Details
Cluster labs	30%	Submission of working MPI/OpenMP/GPU exercises
Performance study	30%	Roofline + scaling analysis of an existing code
Capstone project	40%	Implement & optimise a solver or ML pipeline at scale, plus written report

5.11.2.6 Prerequisites

- Completion of "Basic Scientific Computing" **or** equivalent experience
- Familiarity with Linux command line and version control

5.11.2.7 Key Resources

- replace CHatGPT fantasies

Lecture: High Performance Computing

SWS: 2 **ECTS:** 3

Exercise: High Performance Computing Exercise

SWS: 2 **ECTS:** 3

5.11.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
comp_module_1	Benchmark and profile computational code to evaluate performance and bottlenecks	Scientific Computing	rse_tooling_2	Submit benchmark reports comparing implementations and justifying trade-offs	RSE Curriculum Draft	Link
comp_module_2	Explain and apply principles of approximation theory and numerical precision in scientific computing	Scientific Computing		Answer conceptual questions and implement small examples highlighting precision trade-offs	RSE Curriculum Draft	Link
comp_module_3	Explain floating-point arithmetic and its implications for scientific accuracy and performance	Scientific Computing	comp_module_2	Provide examples showing effects of precision loss and propose mitigations	RSE Curriculum Draft	Link
comp_module_4	Describe common simulation libraries and numerical frameworks (e.g., BLAS, LAPACK, PETSc, Trilinos)	Scientific Computing		List relevant libraries for a task and justify choice or avoidance of custom implementations	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
comp_module_5	Compare interpreted and compiled languages in terms of performance and suitability for computing tasks	Scientific Computing		Write code samples in both types of language and explain their performance characteristics	RSE Curriculum Draft	Link
hpc_module_1	Run batch and array jobs on a cluster, including job dependencies	High-Performance Computing	rse_tooling_3	Submit job scripts using SLURM or similar systems demonstrating correct use of job arrays and dependencies	RSE Curriculum Draft	Link
hpc_module_2	Identify and manage common challenges in HPC systems (e.g., I/O bottlenecks, threading, NUMA memory)	High-Performance Computing	hpc_module_1	Provide performance logs and interpret bottlenecks in a real or simulated HPC task	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
hpc_module_3	Use shell scripting (e.g., Bash) to automate HPC job submission	High-Performance Computing	rse_tooling_3	Submit scripts that automate the execution of HPC jobs and handle job logic	RSE Curriculum Draft	Link
hpc_module_4	Understand and use the principles of accelerator programming (e.g., GPU kernels and frameworks)	High-Performance Computing		Submit a small CUDA or OpenCL program with documentation of the principles used	RSE Curriculum Draft	Link
hpc_module_5	Maintain scientific computing software including use of continuous benchmarking	High-Performance Computing	comp_module_1	Provide benchmark and performance history for evolving versions of software	RSE Curriculum Draft	Link

5.11.4 Sources & Implementations:

5.11.4.1 Curricula

- EUMaster4HPC

5.11.4.2 Courses

- Viral Instructions Hardware
- HPC Computing

- High-Performance Computing

5.11.4.3 Recommended Course Literature

- What every computer scientist should know about floating-point arithmetic

5.11.4.4 Programs

- HPC-carpentry
- RWTH Informatik Bachelor
- RWTH Informatik Master

5.12 Data Science

This module covers the mathematical foundations of data science, introduces the students to statistical data analysis and text processing.

5.12.1 Mathematical Foundations of Data Science

5.12.1.1 Contents

The module teaches the mathematical fundamentals of data science. Topics include a selection from the areas of graph analysis, stochastic models and signal analysis with wavelets.

5.12.1.2 Learning outcomes

Students will acquire comprehensive, detailed and specialised knowledge of the latest findings in selected fundamentals of data science. Students will have an in-depth understanding of selected data science methods. They will be able to analyse novel data assimilation and inference problems, develop and implement solutions, and determine the quality of the solutions. They will be able to develop new ideas and procedures, weigh up alternatives when information is incomplete, and evaluate them taking into account different assessment criteria.

5.12.1.3 Examination Method:

written exam (120 minutes) or oral exam (30 minutes)

Lecture: Mathematical Foundations of Data Science

SWS: 2 **ECTS:** 2

Exercise: Mathematical Foundations of Data Science Exercise

SWS: 2 **ECTS:** 4

5.12.2 Statistical Data Analysis

5.12.2.1 Contents

This module focuses on the statistical study and quantitative analysis of the dependence between observed random variables (e.g., yield/production settings; lifespan/treatment type and injury type). Essential foundations for the statistical treatment of such relationships are provided by the linear regression model, which is studied in detail in the first part of the lecture. Within this framework, topics such as estimation, testing, and uncertainty quantification (analysis of variance) are addressed. In the second part, an introduction to advanced methods and approaches for examining relationships is offered, including nonlinear and nonparametric regression models. Additionally, questions of classification and dimensionality reduction are covered.

5.12.2.2 Learning outcomes

Students will acquire a comprehensive, detailed and specialised understanding of the linear regression model based on the latest findings. They will learn basic concepts and methods of non-parametric statistics. They will also be able to solve complex statistical data analysis problems, weigh up alternative modelling approaches and evaluate them according to different criteria. They will be able to use functions from statistical software packages for this purpose.

5.12.2.3 Examination method

exam (120-180 minutes) or oral exam (30 minutes)

Lecture: Statistical Data Analysis

SWS: 4 **ECTS:** 6

Exercise: Statistical Data Analysis Exercise

SWS: 2 **ECTS:** 3

5.12.3 Text to Data

5.12.3.1 Contents

- Introduction to textual data and Natural Language Processing (NLP): key concepts and use cases
- Text acquisition and preprocessing: tokenisation, stemming/lemmatisation, stopwords, named entity recognition, text cleaning
- Text representations: bag-of-words, TF-IDF, word embeddings (Word2Vec, GloVe), contextualised embeddings (BERT, RoBERTa, GPT variants)
- Text mining and information extraction: key terms, topics, sentiment analysis, relation extraction
- Information Retrieval and Web Search Engines
- Data-to-insight pipelines: from raw text to structured data (CSV/SQL), feature engineering for text data
- Modelling and evaluation: classification, regression, sequence models, transformers; evaluation metrics (accuracy, F1, MCC, AUC)
- Prototyping and reproducibility: experiment tracking, versioning, reproducible pipelines (Docker/Kubernetes, MLflow, DVC)
- Risk management and ethics: bias, fairness, data protection (GDPR), transparency, interpretability (LIME, SHAP)
- Deployment and practical applications: API-based models, batch vs real-time processing, monitoring
- Project work: from a research question to data collection, modelling, evaluation and documentation

5.12.3.2 Learning outcomes

- Understand how text data is collected, preprocessed, and transformed into usable formats
- Be able to select and justify different text representations
- Apply NLP methods to extract relevant information from text
- Develop, train and evaluate text-based models with attention to reproducibility and ethics
- Assess model limitations, interpretability, and risk of errors
- Build a reproducible, scalable text-data pipeline
- Communicate results clearly in reports, presentations, and prototype designs

5.12.3.3 Examination method

project report (x pages) and presentation (15 minutes)

Exercise: Text2Data Exercise

SWS: 2 **ECTS:** 2

Lecture: Text2Data

SWS: 2 **ECTS:** 4

5.12.4 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_data-science_1	Possess comprehensive, detailed, and specialized knowledge of selected fundamentals in the field of Data Science	Data Science		Demonstrate knowledge through theoretical exams and practical assignments	University of Potsdam	Link
gen_data-science_2	Demonstrate an in-depth understanding of selected Data Science methods	Data Science	gen_data-science_1	Apply Data Science methods in practical projects and case studies	University of Potsdam	Link
gen_data-science_3	Analyze novel data assimilation and inference problems, develop and implement solutions, and assess solution quality	Data Science	gen_data-science_2	Solve complex inference problems and present implemented solutions with evaluation	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_data-science_1	Develop new ideas and methods, weigh alternatives under incomplete information, and evaluate them considering different evaluation criteria	Data Science	gen_data-science_2	Present projects showcasing creative problem-solving and alternative evaluations under uncertainty	University of Potsdam	Link
gen_statistics_1	Possess comprehensive, detailed, and specialized understanding of the linear regression model based on the latest research	Data Science, Statistics		Apply linear regression models to practical problems and interpret results	University of Potsdam	Link
gen_statistics_2	Understand fundamental concepts and methods of nonparametric statistics	Data Science, Statistics	gen_statistics_1	Solve problems involving nonparametric methods and explain applied techniques	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_statistics_3	Solve complex statistical data analysis problems, evaluate alternative modeling approaches according to various criteria, and use statistical software packages for analysis	Data Science, Statistics	gen_statistics_2	Develop solutions for complex data problems using appropriate statistical methods and software	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_statistics_4	Demonstrate academic competences including self-organization, planning skills (identifying work steps), scientific thinking and working techniques (developing solutions for complex questions), discussion of methods, verification of hypotheses, application of mathematical and statistical methods, and use of software packages	Data Science, Statistics	gen_statistics_2	Document project workflows demonstrating planning, analysis, evaluation, and use of statistical software tools	University of Potsdam	Link

5.12.5 Sources & Implementations:

5.12.5.1 Curricula

- Empfehlungen Masterstudiengänge Data Science

5.12.5.2 Courses

- Statistical Data Analysis (MATVMD837)
- Mathematical Foundations of Data Science (MAT-DSAM8B)

- Programmieren für Data Scientists Python
- Elements of Machine Learning and Data Science
- Mathematical Foundations of Data Science
- Data Science

5.12.5.3 Programs

- UP Data Science Master
- JMUW Informatik Bachelor
- JMUW Informatik Master
- RWTH Informatik Bachelor

5.13 Complexity Theory

5.13.1 Introduction

This module covers advanced complexity theory.

5.13.2 Contents

The module includes the following topics:

- Deterministic, non-deterministic, probabilistic and parallel computational models
- Complexity classes; reductions
- Complexity of approximation problems
- Selection of advanced topics such as interactive proofs, derandomisation, circuit complexity, communication complexity, parametric complexity
- Properties of NP-complete sets, autoreducibility, polynomial time hierarchy
- complexity of probabilistic algorithms.

5.13.3 Learning Objectives

Students are able to analyse the correctness, security and complexity of processes. Students will be able to classify algorithmic problems according to their complexity. Students will be able to analyse the relationship between different complexity classes.

5.13.4 Examination Methods

- Either a 90-minute written exam.
- Or a 20-30 minute oral examination.

Lecture: Complexity Theory

SWS: 2 **ECTS:** 2

Exercise: Complexity Theory Exercise

SWS: 2 **ECTS:** 4

5.13.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
formal_methods_1	apply formal methods in system design and software engineering.	Computer Science	Basic knowledge of software engineering.	Submit application of formal methods for a given software system.	University of Potsdam	Link
formal_methods_2	Analyse theoretical and practical problems in modelling and implementation using formal methods together with others.	Computer Science	Basic knowledge of formal methods.	Participate in self-regulated team exercises and presentations.	University of Potsdam	Link

5.13.6 Sources & Implementations:

5.13.6.1 Curricula

- None

5.13.6.2 Courses

- Komplexitätstheorie
- Sicherheit, Information und Komplexität
- Komplexitätstheorie II

5.13.6.3 Recommended Course Literature

- None

5.13.6.4 Programs

- UP Computational Science Master
- RWTH Informatik Master
- JMUW Informatik Master

5.14 Embedded Systems

5.14.1 Introduction

Embedded systems control many things in our daily lives. Energy-efficient refrigerators, lift controls and advanced driver assistance systems are just a few examples. Embedded systems also control processes in industrial environments and are used to detect and prevent system failures. This lecture provides a general introduction to the topic of embedded systems and their performance. It presents basic concepts and highlights important differences to “normal” computer systems.

5.14.2 Contents

The module covers a selection of the following topics: - security, reliability, formal methods and dynamic systems. - microcontrollers - programmable logic controllers (PLCs) - PLC programming languages - real-time requirements - real-time operating systems - characteristics of embedded software design - intra-vehicle communication (e.g. CAN bus)

5.14.3 Learning Objectives

Students - can understand the solution to complex problems in embedded systems and apply those to related issues. - describe modern software techniques for embedded systems - apply a model-based, quality-oriented approach to the design of embedded software - recognise special quality requirements for the design of embedded software and take them into account appropriately

5.14.4 Examination Methods

- either 120 min written exam
- or 20-30 min oral examination

Lecture: Embedded Systems

SWS: 2 **ECTS:** 2

Exercise: Embedded Systems Exercise

SWS: 2 **ECTS:** 4

5.14.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
emb_systems_1	evalua	Computer Science		Sub		Link

5.14.6 Sources & Implementations:

5.14.6.1 Curricula

- None

5.14.6.2 Courses

- Eingebettete Systeme
- Ausgewählte Kapitel der Embedded Systems

5.14.6.3 Recommended Course Literature

- None

5.14.6.4 Programs

- RWTH Informatik Master
- JMUW Informatik Master

5.15 Master's Thesis Module: Research Software Engineering Thesis

The master's thesis is the culminating component of the RSE programme. In this module, students apply the full spectrum of Research Software Engineering skills in a real-world research setting, demonstrating their ability to independently design, implement, and document a computational research contribution.

The thesis must address a research question in collaboration with a scientific or applied domain, but its core should include a substantial computational component. This may involve software development, data-intensive research, reproducibility infrastructure, or performance engineering — depending on the chosen topic and specialization.

Each thesis must be supervised jointly by:

- A domain expert (e.g., in physics, life sciences, or humanities)
- An RSE mentor (who ensures the quality and relevance of the computational contribution)

Students are expected to follow best practices in software engineering, version control, testing, and documentation. The final submission must include:

- A written thesis describing both the scientific and software contributions
- A structured, reproducible code repository
- A presentation and defense in a thesis colloquium

The colloquium serves as both a public communication exercise and a final evaluation, where students present their project and reflect on challenges and insights gained during the thesis.

Thesis: RSE Master Thesis

SWS: 0 **ECTS:** 30

5.16 IT-Security and Cryptography

5.16.1 Introduction

This module deals with IT-security and cryptography.

5.16.2 Contents

The course offers a broad overview of concepts and technologies relevant to IT security: - Theoretical aspects: security in information theory and computability, introduction to cryptography (historical and modern ciphers, hash functions, pseudo-random number generators, message authentication codes, public-key cryptography) - Network security: security of protocols and TCP/IP, public key infrastructure, user authentication, Kerberos, IPsec, TLS protocol, SSH, DNS Security, Email Security, and Phishing Attacks. - Software security: security vulnerabilities, common programming errors and techniques for exploiting them, reverse engineering and obfuscation, malware and anti-malware - Platform security: access control models, security policies, operating system security, virtualisation, security mechanisms with hardware support - Methods for secure and reliable transmission and processing of information, error-correcting coding methods - Fundamentals of cryptographic systems, methods for information analysis, complexity aspects, applications - Necessary foundations of mathematics and complexity theory are introduced alongside the topics - Private key cryptosystems, Vernam one-time pad, AES, perfect security, public key cryptosystems, RSA, Diffie-Hellman, Elgamal, Goldwasser-Micali, digital signature, challenge-response procedure, secret sharing, millionaire problem, secure circuit evaluation, homomorphic encryption - Symmetric Encryption, Integrity protection, Asymmetric Encryption, Digital Signatures, Certificates and Public Key Infrastructures, and Authentication and Key Agreement

5.16.3 Learning Objectives

Students are able to - understand the mathematical foundations of secure and reliable information processing and their complexity-theoretical basis - are capable of analysing thesecurity of methods - model threats and evaluate the security of systems critically from the attacker's perspective - understand the purpose and functioning of some security technologies and be aware of their limitations - select and apply appropriate cryptographic primitives in different application scenarios - select suitable security protocols for a given scenario and configure the relevant options for the cryptographic building blocks used within these protocols - identify security requirements and adequate security mechanisms in various application domains - identify potential weaknesses in security protocols not covered in detail during the course and propose appropriate fixes - assess the severity of novel attacks against security protocols and cryptographic primitives

5.16.4 Examination methods

- Either a written exam (90 minutes).
- Or an oral examination (30 minutes).

Lecture: Security and Cryptography

SWS: 2 **ECTS:** 2

Exercise: Security and Cryptography Exercise

SWS: 2 **ECTS:** 4

5.16.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
sec_complex_1	explain the necessity and methods of error-protected transmission and storage of data	Computer Science		describe and apply the taught methods to given examples	University of Potsdam	Link
sec_complex_2	analyse the correctness, security and complexity of algorithms	Computer Science		Submit a written analysis for a given algorithm	University of Potsdam	Link

5.16.6 Sources & Implementations:

5.16.6.1 Curricula

- None

5.16.6.2 Courses

- Sicherheit, Information und Komplexität
- Resiliente Systeme
- IT-Sicherheit I
- Algorithmische Kryptographie
- Einführung in die IT-Sicherheit
- Kryptographie und Datensicherheit

5.16.6.3 Recommended Course Literature

- None

5.16.6.4 Programs

- UP Computational Science Master
- JMUW Informatik Bachelor
- JMUW Informatik Master
- RWTH Informatik Master

5.17 RSE Philosophy

5.17.1 Introduction

The RSE master program is more than a computer science specialisation for researchers. People working as RSE are often involved in digitalization projects, institutional development or other non-technical tasks.

5.17.2 Contents

For a university level study program it is fitting that students learn an abstract high-level understanding of their field so that they can adapt technical models, communication frameworks and policy recommendations to the complex cases. For this they need a solid understanding in some of the more theoretical fields such as …

- philosophy of science
- sociology of technology
- ethics and artificial intelligence
- human computer interaction
- digital humanities

5.17.3 General Competences

This module conveys competences in areas such as but not limited to …

- conducting and leading research (NEW)
- understanding the research cycle (RC)
- interaction with users and stakeholders (USERS)

Seminar: RSE Philosophy This is an introductory class to … **SWS:** 2 **ECTS:** 3

5.17.4 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_philosophy_1	TODO	Research Software Engineering		…		Link

5.17.5 Sources & Implementations:

5.17.5.1 Courses

- Philosophische Grundlagen der Wissenschaften
- Einführung in die Wissenschaftstheorie
- Ethics, technology, and data
- Basismodul Grundlagen der Theoretischen Philosophie

5.17.5.2 Recommended Course Literature

- Values in Science

5.17.5.3 Programs

- RWTH Informatik Bachelor

5.18 RSE Lab Module

Applied Research Software Engineering in scientific institutions

This lab module provides students with a hands-on opportunity to apply research software engineering principles to real-world scientific problems from the MINT disciplines (Mathematics, Informatics, Natural Sciences, and Technology). Students work on projects originating from active research contexts — such as simulations in physics, data analysis in chemistry, modeling in biology, or

Lab: RSE Lab Project

SWS: 8 **ECTS:** 12

5.18.1 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
project_1		Computer Science				Link

5.18.2 Sources & Implementations:

5.18.2.1 Curricula

- None

5.18.2.2 Courses

- Software-Projektpraktikum
- Praktikum
- Softwarepraktikum
- Praktikum Software Engineering
- Praktikum
- Interdisziplinäre Projektarbeit

5.18.2.3 Recommended Course Literature

- None

5.18.2.4 Programs

- JMUW Informatik Bachelor
- JMUW Informatik Master
- RWTH Informatik Bachelor
- RWTH Informatik Master
- UP Informatik/Computational Science Bachelor
- UP Computational Science Master

5.19 RSE Management and Communication

5.19.1 Introduction

This module comprises the communication and management skills that are relevant for working in the interdisciplinary setting of RSE-professionals.

This includes but is not limited to:

- working in a team (see TEAM in (Goth et al. 2024))
- teaching RSE-basics (see TEACH in (Goth et al. 2024))
- project management (see PM in (Goth et al. 2024))
- interaction with users and other stakeholders (see USERS in (Goth et al. 2024))

5.19.2 Contents

- **research management**
  - research cycle
  - open science, FAIR, FAIR4RS
  - publication workflow: obstacles and embargoes
  - legal aspects of research data, e.g. GDPR
  - pseudonymization/anonymization methods for data privacy
  - public databases
- **quality control**
  - requirements engineering
  - trying goals with quality: test-driven development
  - behavior-driven development, Gherkin-Style acceptance testing
  - project folder organization
  - code review principles
- **communication and collaboration**
  - communication frameworks, e.g. AIDA, RACE, 7 C's
  - personality traits and their impact on cooperation
  - collaboration frameworks for remote work
  - realisation and benefits of pair programming and mob/ensemble programming
  - technical English writing skills: writing in issues and merge requests, code review…
  - conflict management, e.g. dealing with researchers that do not listen
  - how to address relevant stakeholders (i.e. users and SEs) with different background knowledge, experience and expectations
  - equity, diversity and inclusion principles
- **team management**
  - challenges of transient teams (that only exist for 5-15 hours)
  - effects of varying team sizes
  - management depending on project size/type
  - specialised roles in a software team
  - intercultural and interdisciplinary differences
  - team management methodologies
  - importance of shared values in a RSE team
  - dual goals: project vs. personal goal
- **time and project management**
  - goal-setting
  - project management methods, their strengths and weaknesses
  - agile (not necessarily Scrum)

- Lean & Kanban (Small-Batch Philosophies)
- division of tasks into sub-tasks and task-dependencies
- iterative workflows
- continuous delivery
- communication with a manager/supervisor

Lecture: RSE Management Lecture This is an introductory lecture covering research, project and team management techniques an RSE needs in everyday life. **SWS:** 2 **ECTS:** 2

Exercise: RSE Management Exercise This is an exercise to apply and practice the taught methods with case-studys, role-plays etc. **SWS:** 2 **ECTS:** 2

5.19.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_8	Build and manage sustainable research software communities	Research Software Engineering, Community Engagement	rse_tooling_13	Document strategies used for user engagement, feedback, and community growth in a real project	RSE Curriculum Draft	Link
rse_practices_9	Work in an agile software development process, including requirement gathering and iteration	Research Software Engineering		Submit a project that uses agile planning (e.g., user stories, sprints, stand-ups) and reflects on iteration outcomes	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_10	Define project scope, gather requirements, and manage stakeholder expectations	Research Software Engineering		Provide a requirements document and stakeholder communication log for a software project	RSE Curriculum Draft	Link
rse_practices_11	Plan for software maintenance and long-term sustainability, including archiving strategies	Research Software Engineering	rse_practices_6	Submit a sustainability or exit plan describing how the software will be maintained or archived	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_management_01	Explain particular implementation choices in a convincing manner	Research Software Engineering		Deliver a defense of chosen implementation decisions in a discussion with a domain expert who has limited technical knowledge (ideally oral examination or project presentation with potential 'customer/user' questions)	RSE Community	Link
rse_management_02	Exemplify and articulate shared team values and their impact on work	Research Software Engineering		Identify core team values and demonstrate how they influence key implementation decisions (e.g., design, communication, and collaboration)	RSE Community	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_management_03	Plan and manage projects using standard methods effectively and efficiently	Research Software Engineering		Develop a comprehensive project plan for a given project, including scope, milestones, risks, resources, and success criteria	RSE Community	Link
rse_management_04	Discuss methods to set up a Diversity, Equity and Inclusion (DEI) framework in an RSE team	Research Software Engineering		Analyse and evaluate a DEI framework for a given project	RSE Community	Link

5.19.4 Sources & Implementations:

5.19.4.1 Courses

- RSE Leadership Course
- Professionelles Projektmanagement in der Praxis
- Communication Psychology
- Grundlagen des Management
- Software-Projektmanagement
- Projektmanagement
- Organizational Behavior & Human Resource Management

5.19.4.2 Recommended Course Literature

- Remote Mob Programming
- Code with the Wisdom of the Crowd
- Collaboration Explained
- Team Topologies
- Technical Agile Coaching with the Samman method
- Lean Product and Process Development
- Extreme Programming Explained

5.19.4.3 Programs

- RWTH Informatik Bachelor
- RWTH Informatik Master
- JMUW Informatik Master

5.20 RSE Lecture Series

5.20.1 Introduction

This lecture series covers current RSE research and provides room for guest lectures and varying focus topics.

5.20.2 Contents

vary between semesters and weeks, but are all related to RSE

5.20.3 Learning Objectives

- students get a broad image of RSE in the wild

5.20.4 Examination Methods

none?

Lecture: RSE Lecture Series

SWS: 2 **ECTS:** 3

5.20.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_0	describe RSE activities	Computer Science		name x different fields and analyse their differences		Link

5.20.6 Sources & Implementations:

5.20.6.1 Curricula

- None

5.20.6.2 Courses

- None

5.20.6.3 Recommended Course Literature

- None

5.20.6.4 Programs

- None

6 Track 2: Domain Science / Natural Science Track

6.1 Suggested Study Plan

Type	Description	SWS	Sem 1	Sem 2	Sem 3	Sem 4
Lecture	RSE Lecture Series	2	3	0	0	0
Seminar	RSE Nuts and Bolts I	2	3	0	0	0
Lecture	RSE Management Lecture	2	2	0	0	0
Exercise	RSE Management Exercise	2	2	0	0	0
Lecture	Statistical Data Analysis	4	6	0	0	0
Exercise	Statistical Data Analysis Exercise	2	3	0	0	0
Lecture	Basic Programming	2	2	0	0	0
Exercise	Basic Programming Exercise	2	4	0	0	0
Lecture	Wildcard Science I	2	2	0	0	0
Seminar	RSE Philosophy	2	3	0	0	0
Lab	Wildcard Science Lab I	2	0	4	0	0
Lecture	Wildcard Science II	2	0	2	0	0
Lecture	Computational Wildcard Science	2	0	3	0	0
Lab	Computational Wildcard Science Lab	2	0	3	0	0
Lab	Science Lab Project	4	0	6	0	0
Lecture	Applied Programming	2	0	2	0	0
Exercise	Applied Programming Exercise	2	0	4	0	0
Lecture	Scientific Computing Basics	2	0	3	0	0
Exercise	Scientific Computing Basics Exercise	2	0	3	0	0
Lab	Wildcard Science Lab II	2	0	0	4	0
Seminar	RSE Nuts and Bolts II	2	0	0	3	0
Lecture	Text2Data	2	0	0	4	0
Exercise	Text2Data Exercise	2	0	0	2	0
Lecture	Software Engineering I	2	0	0	4	0
Exercise	Software Engineering I Exercise	2	0	0	2	0
Lecture	High Performance Computing	2	0	0	3	0
Exercise	High Performance Computing Exercise	2	0	0	3	0
Seminar	RSE Software Engineering	2	0	0	3	0

Type	Description	SWS	Sem 1	Sem 2	Sem 3	Sem 4
Exercise	RSE Software Engineering Exercise	2	0	0	3	0
Thesis	RSE Master Thesis	0	0	0	0	30
	Total ECTS per semester		30	30	31	30

Total Curriculum ECTS: 121

6.2 Wildcard Computational Science

This module offers RSE students the opportunity to deepen their understanding of computational methods specific to a science discipline. Students choose a science module — such as physics, chemistry, biology, or earth sciences — and engage with its computational practices, core questions, and data/software challenges.

The goal is to apply the general competences acquired in the general programming and software engineering courses to the practices and special needs of the chosen discipline. Computational Physics might face different algorithmic or conceptual challenges than computational chemistry. This module is intended for the case that the institution offers such a specialized computational course.

Lecture: Computational Wildcard Science

SWS: 2 **ECTS:** 3

Lab: Computational Wildcard Science Lab

SWS: 2 **ECTS:** 3

6.2.1 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
None		Wildcard Domain Science				Link

6.2.2 Sources & Implementations:

6.2.2.1 Courses

- Computational Chemistry
- Computational Physics
- Computeranwendungen in der Chemie

6.2.2.2 Programs

- UP Master Bioinformatik
- UP Computational Science Master
- RWTH Informatik Bachelor
- RWTH Informatik Master

6.3 RSE Nuts and Bolts

6.3.1 Introduction

This module, inspired by the MIT Missing Semester, addresses the "nuts and bolts" often missing from traditional academic training in computing. It aims to provide students with practical skills and conceptual understanding for building robust, maintainable, and reproducible research software—key competencies in Research Software Engineering (RSE).

6.3.2 General Competencies

The module begins with general-purpose computing tools and techniques that are foundational for any research software engineer:

- Shell tools and scripting
- Command-line environments
- Editors and IDEs (e.g., Vim)
- Version control (Git)
- Data wrangling
- Debugging and profiling
- Metaprogramming
- Security and cryptography

6.3.3 RSE-Specific Topics

Building on these foundations, the module introduces RSE-specific concepts and good practices:

- **Version control and collaboration**
  - Git for code history, collaboration, and issue tracking
- **Virtualization concepts**
  - Containerization and environment management
- **The Data Life Cycle**
  - Managing research data and understanding data provenance
- **Good coding practices**
  - Reproducible and testable code
  - Meaningful documentation and error messages
  - Modular software design
  - Performance-conscious coding
  - Easily installable and distributable software
  - Coding standards, formatting, and linting
- **Software management planning**
  - Writing Data and Software Management Plans
  - Sustainable development and community involvement
- **Low-level programming**
  - Introduction to a compiled language (e.g., C) to expose hardware-level concerns and efficient memory management
- **Long-term software maintenance**
  - Version tracking, bug management, and sustainability strategies
  - Building and maintaining research software communities

6.3.4 Beyond the Basics

Finally, the module touches on practices that support the scholarly nature of research software:

- Software publication and citation (see SP in (Goth et al. 2024))
- Use of domain-specific repositories and registries (see DOMREP in (Goth et al. 2024))

By the end of this module, students will be well-equipped to design, develop, document, and maintain research software that meets high standards of quality, sustainability, and reproducibility.

The module is made up of two seminars that the students take at different stages in their master's program: In the first seminar during their first semester, students mainly learn new concepts and get to know essential tools, whereas the second seminar in the third semester focuses on teaching others about research software and the development process of it (see TEACH in (Goth et al. 2024)).

Seminar: RSE Nuts and Bolts I This is an introductory class to essential techniques an RSE needs in everyday life. **SWS:** 2 **ECTS:** 3

Seminar: RSE Nuts and Bolts II This is an advanced class of RSE techniques that includes a teaching component as part of the preparation for working as an RSE in interdisciplinary teams. **SWS:** 2 **ECTS:** 3

6.3.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_1	Use literate programming tools (e.g., Quarto, Marimo, Pluto.jl, Jupyter) to combine code, results, and narrative	Research Software Engineering		Submit a literate notebook or document integrating code, visualizations, and explanatory text	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_2	Use Python for visualization, scripting, templating, and integration tasks	Research Software Engineering		Submit a Python project demonstrating use of libraries for visualisation, web tasks, and templating	Workshop Participants	Link
rse_tooling_3	Write and use Bash scripts for automation	Research Software Engineering		Submit shell scripts automating file manipulation or computational workflows	Workshop Participants	Link
rse_tooling_4	Apply testing, debugging, and logging techniques to ensure software reliability	Research Software Engineering	rse_tooling_2	Submit logs, test cases, and debugging documentation for a non-trivial Python or Bash project	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_5	Use workflow management tools (e.g., CWL, Nextflow) to design scalable, reproducible pipelines	Research Software Engineering	rse_tooling_3, rse_tooling_11	Submit a reproducible workflow including metadata and input/output definitions	Workshop Participants	Link
rse_tooling_6	Estimate resource requirements for computational tasks using profiling and benchmarking	Research Software Engineering	rse_tooling_2, rse_tooling_5	Provide resource usage profiles and discuss optimization implications	Workshop Participants	Link
rse_tooling_7	Use package managers and virtual environments (e.g., conda, nix) to manage software dependencies	Research Software Engineering		Submit environment definitions and reproducible setup instructions for a project	Workshop Participants	Link
rse_tooling_8	Document and package software for usability and reusability, using generators and modular design	Research Software Engineering	rse_tooling_2	Submit user and developer documentation generated with Sphinx or similar, plus a reusable code module	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_9	Communicate technical RSE topics effectively with non-technical audiences	Research Software Engineering		Prepare and deliver a presentation or write an article explaining RSE concepts to a general audience	Workshop Participants	Link
rse_tooling_10	Apply authentication and authorization mechanisms (e.g., LDAP, ACLs, Active Directory)	Research Software Engineering		Configure and demonstrate access control for a multi-user service or application	Workshop Participants	Link
rse_tooling_11	Make informed decisions about tooling and infrastructure (e.g., Jupyter vs scripts, local vs HPC/cloud)	Research Software Engineering	rse_tooling_1, rse_tooling_2, rse_tooling_3	Submit a comparative analysis justifying tooling and infrastructure choices for a research project	Workshop Participants	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_tooling_12	Teach and practice collaborative development, including version control and code review	Research Software Engineering	rse_tooling_2	Submit a project with version history and documented code reviews	Workshop Participants	Link
rse_tooling_13	Mentor others in research software engineering practices	Research Software Engineering	rse_tooling_12	Document a mentoring session, workshop, or support activity	Workshop Participants	Link
rse_tooling_14	Deploy and maintain web servers for research applications	Research Software Engineering	rse_tooling_2	Deploy a working web application with setup and maintenance documentation	Workshop Participants	Link
rse_tooling_15	Understand and manage file systems, including local and network-attached storage	Research Software Engineering		Document storage strategies and access mechanisms in a real-world setup	Workshop Participants	Link

6.3.6 Sources & Implementations:

6.3.6.1 Courses

- MIT Missing Semester

- CodeRefinery
- INTERSECT Training Materials
- Digital Research Academy Materials (Git, HPC, Reproducibility, Research Software)
- Building Better Research Software (SSI)
- Docker for neuroscience (jupyter book)

6.4 Science Lab Module

Applied Research Software Engineering in STEM sciences

This lab module provides students with a hands-on opportunity to apply research software engineering principles to real-world scientific problems from the MINT disciplines (Mathematics, Informatics, Natural Sciences, and Technology). Students work on projects originating from active research contexts — such as simulations in physics, data analysis in chemistry, modeling in biology, or

Lab: Science Lab Project

SWS: 4 **ECTS:** 6

6.4.1 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
None		Wildcard Domain Science				Link

6.4.2 Sources & Implementations:

6.4.2.1 Programs

- UP Master Bioinformatik

6.5 Classical Software Engineering

To summarise the vast range of the skills a software engineer is typically equipped with, we refer to the Guide to the Software Engineering Body of Knowledge (Bourque, Fairley, and IEEE Computer Society 2014). Because research software engineering is an interface discipline, RSEs are often stronger in topics more commonly encountered in research software contexts (e.g., mathematical and engineering foundations) than in other areas (e.g., software engineering economics). However, they bring a solid level of competence in all software engineering topics. Therefore, RSEs can set and analyse software requirements in the context of open-ended, question-driven research. They can design software so that it can sustainably grow, often in an environment of rapid turnover of contributors. They are competent in implementing solutions themselves in a wide range of technologies fit for different scientific applications. They can formulate and implement various types of tests, they can independently maintain software and automate operations of the integration and release process. They can provide working, scalable, and future-proof solutions in a professional context and with common project and software management techniques, adapted to the needs of the research environment. Finally, as people who have often gained significant research experience in a particular discipline, they combine the necessary foundations from their domain with software engineering skills to develop complex software.(Goth et al. 2024)

This module tries to lay the foundations for the advanced RSE software engineering training.

Bourque, Pierre, Richard E. Fairley, and IEEE Computer Society. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd ed. Washington, DC, USA: IEEE Computer Society Press.

Gesellschaft für Informatik e.V. (GI). 2016. "Empfehlungen Für Bachelor- Und Masterprogramme Im Studienfach Informatik an Hochschulen." GI-Empfehlungen.

Goth, F, R Alves, M Braun, LJ Castro, G Chourdakis, S Christ, J Cohen, et al. 2024. "Foundational Competencies and Responsibilities of a Research Software Engineer [Version 1; Peer Review: Awaiting Peer Review]." *F1000Research* 13 (1429). https://doi.org/10.12688/f1000research.157778.1.

Segal, Judith. 2009. "Some Challenges Facing Software Engineers Developing Software for Scientists." In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE. https://doi.org/10.1109/secse.2009.5069156.

6.5.1 Software Engineering I

Basic concepts of software engineering, software and product life cycle, process models for the design of large software systems, semantic aspects of domain description, hierarchy, parallelism, real-time and embedded systems as fundamental paradigms, organizational principles of complex software systems, design by contract, patterns in modeling and design methods of quality assurance, evolution and re-engineering, selected languages and tools for process-

and object-oriented modeling, methods and languages for object-oriented design, architectures and architectural patterns of software systems, architecture of enterprise applications, design and implementation models in the object-oriented paradigm, e.g., Java 2 SE, design patterns, software testing methods.

Lecture: Software Engineering I

SWS: 2 **ECTS:** 4

Exercise: Software Engineering I Exercise

SWS: 2 **ECTS:** 2

6.5.2 Software Engineering 2

The module covers a selection of advanced topics in the field of software engineering, such as software quality assurance, service engineering, virtualization, programming languages and design, and formal methods in system design.

Lecture: Software Engineering II

SWS: 2 **ECTS:** 2

Exercise: Software Engineering II Exercise

SWS: 2 **ECTS:** 4

6.5.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_programming_1	Understand the fundamental concepts of software engineering	Computer Science		Demonstrate understanding through theoretical assessments and practical examples	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_programming_2	Apply various approaches of software engineering	Computer Science	gen_programming_1	Complete assignments or projects using different software engineering methods	University of Potsdam	Link
gen_programming_3	Identify and utilize essential technologies and tools for specification, component-based development, and quality assurance of modern software systems	Computer Science	gen_programming_1	Work with selected tools and technologies in practical exercises and case studies	University of Potsdam	Link
gen_programming_4	Demonstrate an in-depth understanding and ability to apply various approaches of software engineering	Computer Science	gen_programming_1, gen_programming_2	Successfully complete advanced projects employing different software engineering methods	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_programming_5	Understand the characteristics of a wide range of technologies and tools for specification, component-based development, and quality assurance of modern software systems, and apply them in various contexts	Computer Science	gen_programming_3	Apply appropriate technologies and tools in complex case studies and demonstrate their use in different application scenarios	University of Potsdam	Link

6.5.4 Sources & Implementations:

6.5.4.1 Curricula

- Computing Curricula 2020

6.5.4.2 Courses

- Software Engineering I
- Software Engineering
- Softwaretechnik

6.5.4.3 Programs

- UP Informatik/Computational Science Bachelor
- JMUW Informatik Bachelor
- RWTH Informatik Bachelor

6.6 Fundamentals of Computer Science

This module covers the basics of computer science and introduces the students from domain sciences to programming.

6.6.1 Basics of Computer Science

6.6.1.1 Basic Concepts

- Introduction to computer science, basic concepts of operating systems using UNIX/Linux as an example
- From problem to algorithm: concept of an algorithm, design of algorithms, pseudocode, refinement, brute-force algorithms, models and modeling, graphs and their representation, simple algorithms on graphs, analysis of algorithms (correctness, termination, runtime)
- Implementation of algorithms (e.g., using Python)
- Programming paradigms: procedural, object-oriented, and functional programming; recursion versus iteration
- From program to process: assembly languages, assembler, compiler, interpreter, syntax and semantics of programming languages
- Limits of algorithms: computability, decidability, undecidability

Lecture: Basic Programming

SWS: 2 **ECTS:** 2

Exercise: Basic Programming Exercise

SWS: 2 **ECTS:** 4

6.6.2 Applied Programming

6.6.2.1 Procedural Programming Concepts

Programming with an imperative-procedural language (such as C):

- Data types, type casting, control structures, functions and procedures, parameter passing paradigms, call stack
- Pointers, arrays, strings, structured types
- Errors and their handling
- Dynamic memory management
- Program libraries

6.6.2.2 Programming in an Object-Oriented Language (e.g., Java)

- Classes, objects, constructors
- Inheritance, polymorphism, abstract classes/interfaces
- Exceptions and exception handling
- Namespaces (packages)
- Generic classes and types
- Program libraries

Lecture: Applied Programming

SWS: 2 **ECTS:** 2

Exercise: Applied Programming Exercise

SWS: 2 **ECTS:** 4

6.6.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_1	Use an imperative-procedural programming language (e.g., C) and an object-oriented language (e.g., Java) with confidence	Computer Science		Submit working programs in both languages demonstrating syntax and language-specific features	University of Potsdam	Link
ex_programming_mod1_2	Implement basic data structures and algorithms	Computer Science	ex_programming_mod1_1	Submit a project with implemented algorithms and data structures (e.g., lists, trees, sorting)	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_3	Distinguish between error types and handle them appropriately in code	Computer Science	ex_programming_mod1_1	Demonstrate error handling techniques in submitted code (e.g., input validation, error codes, exceptions)	University of Potsdam	Link
ex_programming_mod1_4	Identify and use appropriate library functions in programming tasks	Computer Science	ex_programming_mod1_1	Integrate external libraries in coding tasks and document their usage	University of Potsdam	Link
ex_programming_mod1_5	Use basic functions and mechanisms of operating systems using UNIX/Linux as an example	Computer Science		Demonstrate file handling, permissions, and process control using UNIX/Linux commands	University of Potsdam	Link
ex_programming_mod1_6	Create and refine simple algorithms using semi-formal notation	Computer Science		Submit pseudocode or flowcharts for given algorithmic problems	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_7	Evaluate and compare algorithms using runtime analysis	Computer Science	ex_programming_mod1_6	Provide time complexity comparisons for multiple algorithmic solutions	University of Potsdam	Link
ex_programming_mod1_8	Implement simple algorithms using imperative and functional programming styles (e.g., in Python)	Computer Science	ex_programming_mod1_6	Submit code demonstrating both imperative and functional styles for the same problem	University of Potsdam	Link
ex_programming_mod1_9	Distinguish between programming paradigms and identify their characteristics	Computer Science	ex_programming_mod1_1	Classify given code snippets by paradigm and justify the classification	University of Potsdam	Link
C10	Express simple programs in an assembly language	Computer Science		Translate simple high-level logic into assembler code	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
C11	Discuss the limits of algorithms, including computability and decidability	Computer Science		Write a short essay or present on concepts such as the Halting Problem or undecidability	University of Potsdam	Link

6.6.4 Sources & Implementations:

6.6.4.1 Curricula

- Computing Curricula 2020

6.6.4.2 Courses

- Grundlagen der Programmierung
- Praxis der Programmierung
- Python for Psychologists
- Grundlagen der Informatik
- Grundlagen der Programmierung
- Programmierung

6.6.4.3 Programs

- UP Informatik/Computational Science Bachelor
- JMUW Informatik Bachelor
- RWTH Informatik Bachelor

6.7 Wildcard Science Module

This module offers RSE students the opportunity to deepen their understanding of a scientific discipline outside of their home domain. Students choose a science module — such as physics, chemistry, biology, or earth sciences — and engage with its research practices, core questions, and data/software challenges.

The goal is to help students become better collaborators by gaining first-hand exposure to the terminology, logic, and needs of another scientific domain. This broadens the student's ability to apply RSE skills in interdisciplinary teams and unfamiliar environments.

The module may consist of lectures, lab sessions, and domain-specific mini-projects. RSEs are encouraged to reflect on how software engineering, data handling, reproducibility, and tooling intersect with the chosen discipline.

This module is deliberately flexible to accommodate institutional offerings and student interests as well as providing the option to stay attached to the identity of the chosen discipline.

Lecture: Wildcard Science I

SWS: 2 **ECTS:** 2

Lecture: Wildcard Science II

SWS: 2 **ECTS:** 2

Lab: Wildcard Science Lab I

SWS: 2 **ECTS:** 4

Lab: Wildcard Science Lab II

SWS: 2 **ECTS:** 4

6.7.1 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
None		Wildcard Domain Science				Link

6.7.2 Sources & Implementations:

6.7.2.1 Programs

- JMUW Master Physik
- JMUW Master Chemie

6.8 RSE-Software Engineering

This module extends the Classical Software Engineering Module with research specific learnings. This includes but is not limited to

- software re-use (see SRU in (Goth et al. 2024))
- creating documented code building blocks (see DOCBB in (Goth et al. 2024))
- building distributable software (see DIST in (Goth et al. 2024))
- research specific programming languages
- research specific code requirements (scalability, functional programming, …)
- Adapting the software life cycle to research (see SWLC in (Goth et al. 2024))
- Software behaviour awareness and analysis (see MOD in (Goth et al. 2024))
- Research specific Engineering Patterns

Also, the seminar provides room for reflection and discussions of SE lab experiences.

Seminar: RSE Software Engineering

SWS: 2 **ECTS:** 3

Exercise: RSE Software Engineering Exercise

SWS: 2 **ECTS:** 3

6.8.1 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_1	Apply good coding practices including formatting, linting, and modular design	Research Software Engineering	rse_tooling_2	Submit a code project demonstrating modularity, consistent formatting, and use of linters	RSE Curriculum Draft	Link
rse_practices_2	Write code and documentation that supports reproducibility in research	Research Software Engineering	rse_tooling_1, rse_tooling_4	Submit a project with data, software, and instructions allowing full reproduction of results	RSE Curriculum Draft	Link
rse_practices_3	Organise files and name code artifacts using clear, consistent conventions	Research Software Engineering		Submit a software repository with a structured layout and consistent naming scheme	RSE Curriculum Draft	Link
rse_practices_4	Version control code and collaborate using platforms like GitHub or GitLab	Research Software Engineering	rse_tooling_12	Participate in a collaborative coding project using Git-based workflows and merge requests	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_5	Write effective documentation and user-facing error messages	Research Software Engineering	rse_tooling_8	Provide documentation and example error handling demonstrating clarity and user support	RSE Curriculum Draft	Link
rse_practices_12	Write performant code suitable for use in compute-intensive contexts	Research Software Engineering	rse_tooling_2	Submit benchmark results comparing an optimized version of code with a naive implementation	RSE Curriculum Draft	Link
rse_practices_13	Publish code and software in trusted repositories and package managers	Research Software Engineering	rse_practices_6	Publish software to a repository (e.g., GitHub, PyPI, CRAN) and register it with a long-term archive (e.g., Zenodo)	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_6	Apply licensing and publishing strategies to make software reusable and citable	Research Software Engineering		Submit a software project with an appropriate open license and published DOI (e.g., via Zenodo)	RSE Curriculum Draft	Link
rse_practices_7	Apply principles of Open Source and FAIR (Findable, Accessible, Interoperable, Reusable) software	Research Software Engineering	rse_practices_6	Review or create a software project and evaluate its compliance with FAIR/Open Source principles	RSE Curriculum Draft	Link
rse_practices_14	Manage data within a software project in accordance with best practices	Research Software Engineering, Data Management		Submit a data-driven project showing clear data organisation, metadata, and reproducibility	RSE Curriculum Draft	Link

6.8.2 Sources & Implementations:

6.8.2.1 Courses

- Research Software Engineering

6.8.2.2 Programs

- UP Computational Science Master

6.9 RSE Computing

RSEs with expertise in HPC and other performance-critical computing domains specialize in optimizing code for efficient execution across various platforms, including clusters, cloud, edge, and embedded systems. They understand parallel programming models, hardware-specific optimizations, profiling tools, and platform constraints such as memory, energy, and latency. Their skills enable them to adapt software to diverse infrastructures, manage complex dependencies, and support researchers in accessing and using advanced computing resources effectively and sustainably.

6.9.1 Basic Scientific Computing

6.9.1.1 Module Overview

This module provides an entry-level yet rigorous foundation in scientific computing for graduate students and researchers who need to **design, implement, and evaluate computational experiments**. Learners gain an awareness of the numerical underpinnings of modern simulation and data-driven research, with an emphasis on writing *reproducible, efficient, and trustworthy* code.

6.9.1.2 Intended Learning Outcomes

By the end of the module participants will be able to

1. Benchmark small programs and interpret performance metrics in a research context.
2. Explain how approximation theory and floating-point arithmetic affect numerical accuracy and stability.
3. Identify when to use established simulation libraries (e.g. BLAS/LAPACK, PETSc, Trilinos) instead of custom code.
4. Write simple GPU kernels and describe the core principles of accelerator programming.
5. Submit and monitor batch & array jobs on a mid-size compute cluster.
6. Describe common HPC challenges—such as I/O bottlenecks, threading, and NUMA—and propose mitigation strategies.
7. Maintain research software through continuous benchmarking.

6.9.1.3 Syllabus (Indicative Content)

Week	Theme	Topics
1	**Benchmarking & Profiling**	Timing strategies · micro vs. macro benchmarks · tooling overview
2	**Precision & Approximation**	IEEE-754 recap · conditioning & stability · error propagation
3	**Scientific Libraries**	BLAS/LAPACK anatomy · hierarchical I/O libraries · overview of PETSc/Trilinos/Hypre
4	**GPU Primer**	Kernel model · memory hierarchy · CUDA/OpenCL/PyTorch lightning intro
5	**Working on a Cluster**	Slurm basics · job arrays · job dependencies · simple Bash launchers
6	**HPC Pitfalls**	I/O throughput · thread oversubscription · NUMA awareness
7	**Software Maintenance**	Regression + performance tests · continuous benchmarking pipelines

6.9.1.4 Teaching & Learning Methods

Short lectures (30%) are coupled with hands-on labs (70%). Students complete **weekly notebooks** and a **mini-project** that reproduces and optimises a published computational result.

6.9.1.5 Assessment

Component	Weight	Details
Continuous labs	40%	Weekly graded notebooks
Final mini-project	60%	Report, code, and benchmark suite

6.9.1.6 Prerequisites

- Basic programming in Python, C/C++, or Julia
- Undergraduate calculus & linear algebra

6.9.1.7 Key Resources

ChatGPT fantasy

Lecture: Scientific Computing Basics

SWS: 2 **ECTS:** 3

Exercise: Scientific Computing Basics Exercise

SWS: 2 **ECTS:** 3

6.9.2 High-Performance Scientific Computing

6.9.2.1 Module Overview

Building on "Basic Scientific Computing", this module dives deeper into **scalable algorithms, architectures, and software engineering techniques** required to run large-scale simulations and data analysis on high-performance computing (HPC) systems.

6.9.2.2 Intended Learning Outcomes

Participants who successfully complete the module will be able to

1. Classify scientific problems by their dominant parallel pattern (memory-parallel, compute-parallel, task-parallel).
2. Map each class to appropriate numerical libraries and hardware architectures.
3. Analyse floating-point and algorithmic approximation errors at scale.
4. Explain modern HPC hardware features (GPUs, SIMD/AVX, NUMA, high-speed interconnects) and select relevant optimisation strategies.
5. Design portable, performance-portable code employing MPI, OpenMP, and accelerator frameworks.
6. Use continuous benchmarking to guide sustainable performance evolution of research software.
7. Plan for long-term maintenance, archival, and FAIR publication of large-scale codes and data.

6.9.2.3 Syllabus (Indicative Content)

Week	Theme	Topics
1	**Parallel Problem Taxonomy**	Sparse vs. dense linear algebra · embarrassingly parallel workloads
2	**Distributed Memory (MPI)**	Domain decomposition · halo exchange · scalability metrics
3	**Shared Memory & SIMD**	OpenMP · threading pitfalls · AVX intrinsics
4	**Accelerator Programming**	Multi-GPU kernels · unified memory · portability layers (Kokkos, SYCL)
5	**Advanced I/O & Checkpointing**	Parallel file systems · burst buffers · HDF5/ADIOS-based workflows
6	**Performance Engineering**	Roofline model · continuous & comparative benchmarking · autotuning
7	**Sustainable HPC Software**	Release engineering · long-term archiving · community governance

6.9.2.4 Teaching & Learning Methods

Blended delivery: interactive lectures (40%), coding workshops on the national cluster (50%), expert seminars (10%).

6.9.2.5 Assessment

Component	Weight	Details
Cluster labs	30%	Submission of working MPI/OpenMP/GPU exercises
Performance study	30%	Roofline + scaling analysis of an existing code
Capstone project	40%	Implement & optimise a solver or ML pipeline at scale, plus written report

6.9.2.6 Prerequisites

- Completion of “Basic Scientific Computing” **or** equivalent experience
- Familiarity with Linux command line and version control

6.9.2.7 Key Resources

- replace CHatGPT fantasies

Lecture: High Performance Computing

SWS: 2 **ECTS:** 3

Exercise: High Performance Computing Exercise

SWS: 2 **ECTS:** 3

6.9.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
comp_module_1	Benchmark and profile computational code to evaluate performance and bottlenecks	Scientific Computing	rse_tooling_2	Submit benchmark reports comparing implementations and justifying trade-offs	RSE Curriculum Draft	Link
comp_module_2	Explain and apply principles of approximation theory and numerical precision in scientific computing	Scientific Computing		Answer conceptual questions and implement small examples highlighting precision trade-offs	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
comp_module_3	Explain floating-point arithmetic and its implications for scientific accuracy and performance	Scientific Computing	comp_module_2	Provide examples showing effects of precision loss and propose mitigations	RSE Curriculum Draft	Link
comp_module_4	Describe common simulation libraries and numerical frameworks (e.g., BLAS, LAPACK, PETSc, Trilinos)	Scientific Computing		List relevant libraries for a task and justify choice or avoidance of custom implementations	RSE Curriculum Draft	Link
comp_module_5	Compare interpreted and compiled languages in terms of performance and suitability for computing tasks	Scientific Computing		Write code samples in both types of language and explain their performance characteristics	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
hpc_module_1	Run batch and array jobs on a cluster, including job dependencies	High-Performance Computing	rse_tooling_3	Submit job scripts using SLURM or similar systems demonstrating correct use of job arrays and dependencies	RSE Curriculum Draft	Link
hpc_module_2	Identify and manage common challenges in HPC systems (e.g., I/O bottlenecks, threading, NUMA memory)	High-Performance Computing	hpc_module_1	Provide performance logs and interpret bottlenecks in a real or simulated HPC task	RSE Curriculum Draft	Link
hpc_module_3	Use shell scripting (e.g., Bash) to automate HPC job submission	High-Performance Computing	rse_tooling_3	Submit scripts that automate the execution of HPC jobs and handle job logic	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
hpc_module_4	Understand and use the principles of accelerator programming (e.g., GPU kernels and frameworks)	High-Performance Computing		Submit a small CUDA or OpenCL program with documentation of the principles used	RSE Curriculum Draft	Link
hpc_module_5	Maintain scientific computing software including use of continuous benchmarking	High-Performance Computing	comp_module_1	Provide benchmark and performance history for evolving versions of software	RSE Curriculum Draft	Link

6.9.4 Sources & Implementations:

6.9.4.1 Curricula

- EUMaster4HPC

6.9.4.2 Courses

- Viral Instructions Hardware
- HPC Computing
- High-Performance Computing

6.9.4.3 Recommended Course Literature

- What every computer scientist should know about floating-point arithmetic

6.9.4.4 Programs

- HPC-carpentry
- RWTH Informatik Bachelor
- RWTH Informatik Master

6.10 Data Science

This module covers the mathematical foundations of data science, introduces the students to statistical data analysis and text processing.

6.10.1 Mathematical Foundations of Data Science

6.10.1.1 Contents

The module teaches the mathematical fundamentals of data science. Topics include a selection from the areas of graph analysis, stochastic models and signal analysis with wavelets.

6.10.1.2 Learning outcomes

Students will acquire comprehensive, detailed and specialised knowledge of the latest findings in selected fundamentals of data science. Students will have an in-depth understanding of selected data science methods. They will be able to analyse novel data assimilation and inference problems, develop and implement solutions, and determine the quality of the solutions. They will be able to develop new ideas and procedures, weigh up alternatives when information is incomplete, and evaluate them taking into account different assessment criteria.

6.10.1.3 Examination Method:

written exam (120 minutes) or oral exam (30 minutes)

Lecture: Mathematical Foundations of Data Science

SWS: 2 **ECTS:** 2

Exercise: Mathematical Foundations of Data Science Exercise

SWS: 2 **ECTS:** 4

6.10.2 Statistical Data Analysis

6.10.2.1 Contents

This module focuses on the statistical study and quantitative analysis of the dependence between observed random variables (e.g., yield/production settings; lifespan/treatment type and injury type). Essential foundations for the statistical treatment of such relationships are provided by the linear regression model, which is studied in detail in the first part of the lecture. Within this framework, topics such as estimation, testing, and uncertainty quantification (analysis of variance) are addressed. In the second part, an introduction to advanced methods and approaches for examining relationships is offered, including nonlinear and nonparametric regression models. Additionally, questions of classification and dimensionality reduction are covered.

6.10.2.2 Learning outcomes

Students will acquire a comprehensive, detailed and specialised understanding of the linear regression model based on the latest findings. They will learn basic concepts and methods of non-parametric statistics. They will also be able to solve complex statistical data analysis problems, weigh up alternative modelling approaches and evaluate them according to different criteria. They will be able to use functions from statistical software packages for this purpose.

6.10.2.3 Examination method

exam (120-180 minutes) or oral exam (30 minutes)

Lecture: Statistical Data Analysis

SWS: 4 **ECTS:** 6

Exercise: Statistical Data Analysis Exercise

SWS: 2 **ECTS:** 3

6.10.3 Text to Data

6.10.3.1 Contents

- Introduction to textual data and Natural Language Processing (NLP): key concepts and use cases
- Text acquisition and preprocessing: tokenisation, stemming/lemmatisation, stopwords, named entity recognition, text cleaning

- Text representations: bag-of-words, TF-IDF, word embeddings (Word2Vec, GloVe), contextualised embeddings (BERT, RoBERTa, GPT variants)
- Text mining and information extraction: key terms, topics, sentiment analysis, relation extraction
- Information Retrieval and Web Search Engines
- Data-to-insight pipelines: from raw text to structured data (CSV/SQL), feature engineering for text data
- Modelling and evaluation: classification, regression, sequence models, transformers; evaluation metrics (accuracy, F1, MCC, AUC)
- Prototyping and reproducibility: experiment tracking, versioning, reproducible pipelines (Docker/Kubernetes, MLflow, DVC)
- Risk management and ethics: bias, fairness, data protection (GDPR), transparency, interpretability (LIME, SHAP)
- Deployment and practical applications: API-based models, batch vs real-time processing, monitoring
- Project work: from a research question to data collection, modelling, evaluation and documentation

6.10.3.2 Learning outcomes

- Understand how text data is collected, preprocessed, and transformed into usable formats
- Be able to select and justify different text representations
- Apply NLP methods to extract relevant information from text
- Develop, train and evaluate text-based models with attention to reproducibility and ethics
- Assess model limitations, interpretability, and risk of errors
- Build a reproducible, scalable text-data pipeline
- Communicate results clearly in reports, presentations, and prototype designs

6.10.3.3 Examination method

project report (x pages) and presentation (15 minutes)

Exercise: Text2Data Exercise

SWS: 2 **ECTS:** 2

Lecture: Text2Data

SWS: 2 **ECTS:** 4

6.10.4 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_data-science_1	Possess comprehensive, detailed, and specialized knowledge of selected fundamentals in the field of Data Science	Data Science		Demonstrate knowledge through theoretical exams and practical assignments	University of Potsdam	Link
gen_data-science_2	Demonstrate an in-depth understanding of selected Data Science methods	Data Science	gen_data-science_1	Apply Data Science methods in practical projects and case studies	University of Potsdam	Link
gen_data-science_3	Analyze novel data assimilation and inference problems, develop and implement solutions, and assess solution quality	Data Science	gen_data-science_2	Solve complex inference problems and present implemented solutions with evaluation	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_data-science_1	Develop new ideas and methods, weigh alternatives under incomplete information, and evaluate them considering different evaluation criteria	Data Science	gen_data-science_2	Present projects showcasing creative problem-solving and alternative evaluations under uncertainty	University of Potsdam	Link
gen_statistics_1	Possess comprehensive, detailed, and specialized understanding of the linear regression model based on the latest research	Data Science, Statistics		Apply linear regression models to practical problems and interpret results	University of Potsdam	Link
gen_statistics_2	Understand fundamental concepts and methods of nonparametric statistics	Data Science, Statistics	gen_statistics_1	Solve problems involving nonparametric methods and explain applied techniques	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_statistics_3	Solve complex statistical data analysis problems, evaluate alternative modeling approaches according to various criteria, and use statistical software packages for analysis	Data Science, Statistics	gen_statistics_2	Develop solutions for complex data problems using appropriate statistical methods and software	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_statistics_4	Demonstrate academic competences including self-organization, planning skills (identifying work steps), scientific thinking and working techniques (developing solutions for complex questions), discussion of methods, verification of hypotheses, application of mathematical and statistical methods, and use of software packages	Data Science, Statistics	gen_statistics_2	Document project workflows demonstrating planning, analysis, evaluation, and use of statistical software tools	University of Potsdam	Link

6.10.5 Sources & Implementations:

6.10.5.1 Curricula

- Empfehlungen Masterstudiengänge Data Science

6.10.5.2 Courses

- Statistical Data Analysis (MATVMD837)
- Mathematical Foundations of Data Science (MAT-DSAM8B)

- Programmieren für Data Scientists Python
- Elements of Machine Learning and Data Science
- Mathematical Foundations of Data Science
- Data Science

6.10.5.3 Programs

- UP Data Science Master
- JMUW Informatik Bachelor
- JMUW Informatik Master
- RWTH Informatik Bachelor

6.11 Master's Thesis Module: Research Software Engineering Thesis

The master's thesis is the culminating component of the RSE programme. In this module, students apply the full spectrum of Research Software Engineering skills in a real-world research setting, demonstrating their ability to independently design, implement, and document a computational research contribution.

The thesis must address a research question in collaboration with a scientific or applied domain, but its core should include a substantial computational component. This may involve software development, data-intensive research, reproducibility infrastructure, or performance engineering — depending on the chosen topic and specialization.

Each thesis must be supervised jointly by:

- A domain expert (e.g., in physics, life sciences, or humanities)
- An RSE mentor (who ensures the quality and relevance of the computational contribution)

Students are expected to follow best practices in software engineering, version control, testing, and documentation. The final submission must include:

- A written thesis describing both the scientific and software contributions
- A structured, reproducible code repository
- A presentation and defense in a thesis colloquium

The colloquium serves as both a public communication exercise and a final evaluation, where students present their project and reflect on challenges and insights gained during the thesis.

Thesis: RSE Master Thesis

SWS: 0 **ECTS:** 30

6.12 RSE Philosophy

6.12.1 Introduction

The RSE master program is more than a computer science specialisation for researchers. People working as RSE are often involved in digitalization projects, institutional development or other non-technical tasks.

6.12.2 Contents

For a university level study program it is fitting that students learn an abstract high-level understanding of their field so that they can adapt technical models, communication frameworks and policy recommendations to the complex cases. For this they need a solid understanding in some of the more theoretical fields such as …

- philosophy of science
- sociology of technology
- ethics and artificial intelligence
- human computer interaction
- digital humanities

6.12.3 General Competences

This module conveys competences in areas such as but not limited to …

- conducting and leading research (NEW)
- understanding the research cycle (RC)
- interaction with users and stakeholders (USERS)

Seminar: RSE Philosophy This is an introductory class to … **SWS:** 2 **ECTS:** 3

6.12.4 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_phi-loso-phy_1	TODO	Research Software Engineering		...		Link

6.12.5 Sources & Implementations:

6.12.5.1 Courses

- Philosophische Grundlagen der Wissenschaften
- Einführung in die Wissenschaftstheorie
- Ethics, technology, and data
- Basismodul Grundlagen der Theoretischen Philosophie

6.12.5.2 Recommended Course Literature

- Values in Science

6.12.5.3 Programs

- RWTH Informatik Bachelor

6.13 RSE Management and Communication

6.13.1 Introduction

This module comprises the communication and management skills that are relevant for working in the interdisciplinary setting of RSE-professionals.

This includes but is not limited to:

- working in a team (see TEAM in (Goth et al. 2024))
- teaching RSE-basics (see TEACH in (Goth et al. 2024))
- project management (see PM in (Goth et al. 2024))
- interaction with users and other stakeholders (see USERS in (Goth et al. 2024))

6.13.2 Contents

- **research management**
  - research cycle
  - open science, FAIR, FAIR4RS
  - publication workflow: obstacles and embargoes
  - legal aspects of research data, e.g. GDPR
  - pseudonymization/anonymization methods for data privacy
  - public databases
- **quality control**
  - requirements engineering
  - trying goals with quality: test-driven development
  - behavior-driven development, Gherkin-Style acceptance testing
  - project folder organization
  - code review principles
- **communication and collaboration**
  - communication frameworks, e.g. AIDA, RACE, 7 C's
  - personality traits and their impact on cooperation
  - collaboration frameworks for remote work
  - realisation and benefits of pair programming and mob/ensemble programming
  - technical English writing skills: writing in issues and merge requests, code review…
  - conflict management, e.g. dealing with researchers that do not listen
  - how to address relevant stakeholders (i.e. users and SEs) with different background knowledge, experience and expectations
  - equity, diversity and inclusion principles
- **team management**
  - challenges of transient teams (that only exist for 5-15 hours)
  - effects of varying team sizes
  - management depending on project size/type
  - specialised roles in a software team
  - intercultural and interdisciplinary differences
  - team management methodologies
  - importance of shared values in a RSE team
  - dual goals: project vs. personal goal
- **time and project management**
  - goal-setting
  - project management methods, their strengths and weaknesses
  - agile (not necessarily Scrum)

- Lean & Kanban (Small-Batch Philosophies)
- division of tasks into sub-tasks and task-dependencies
- iterative workflows
- continuous delivery
- communication with a manager/supervisor

Lecture: RSE Management Lecture This is an introductory lecture covering research, project and team management techniques an RSE needs in everyday life. **SWS:** 2 **ECTS:** 2

Exercise: RSE Management Exercise This is an exercise to apply and practice the taught methods with case-studys, role-plays etc. **SWS:** 2 **ECTS:** 2

6.13.3 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_8	Build and manage sustainable research software communities	Research Software Engineering, Community Engagement	rse_tooling_13	Document strategies used for user engagement, feedback, and community growth in a real project	RSE Curriculum Draft	Link
rse_practices_9	Work in an agile software development process, including requirement gathering and iteration	Research Software Engineering		Submit a project that uses agile planning (e.g., user stories, sprints, stand-ups) and reflects on iteration outcomes	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_practices_10	Define project scope, gather requirements, and manage stakeholder expectations	Research Software Engineering		Provide a requirements document and stakeholder communication log for a software project	RSE Curriculum Draft	Link
rse_practices_11	Plan for software maintenance and long-term sustainability, including archiving strategies	Research Software Engineering	rse_practices_6	Submit a sustainability or exit plan describing how the software will be maintained or archived	RSE Curriculum Draft	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_management_01	Explain particular implementation choices in a convincing manner	Research Software Engineering		Deliver a defense of chosen implementation decisions in a discussion with a domain expert who has limited technical knowledge (ideally oral examination or project presentation with potential 'customer/user' questions)	RSE Community	Link
rse_management_02	Exemplify and articulate shared team values and their impact on work	Research Software Engineering		Identify core team values and demonstrate how they influence key implementation decisions (e.g., design, communication, and collaboration)	RSE Community	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_management_03	Plan and manage projects using standard methods effectively and efficiently	Research Software Engineering		Develop a comprehensive project plan for a given project, including scope, milestones, risks, resources, and success criteria	RSE Community	Link
rse_management_04	Discuss methods to set up a Diversity, Equity and Inclusion (DEI) framework in an RSE team	Research Software Engineering		Analyse and evaluate a DEI framework for a given project	RSE Community	Link

6.13.4 Sources & Implementations:

6.13.4.1 Courses

- RSE Leadership Course
- Professionelles Projektmanagement in der Praxis
- Communication Psychology
- Grundlagen des Management
- Software-Projektmanagement
- Projektmanagement
- Organizational Behavior & Human Resource Management

6.13.4.2 Recommended Course Literature

- Remote Mob Programming
- Code with the Wisdom of the Crowd
- Collaboration Explained
- Team Topologies
- Technical Agile Coaching with the Samman method
- Lean Product and Process Development
- Extreme Programming Explained

6.13.4.3 Programs

- RWTH Informatik Bachelor
- RWTH Informatik Master
- JMUW Informatik Master

6.14 RSE Lecture Series

6.14.1 Introduction

This lecture series covers current RSE research and provides room for guest lectures and varying focus topics.

6.14.2 Contents

vary between semesters and weeks, but are all related to RSE

6.14.3 Learning Objectives

- students get a broad image of RSE in the wild

6.14.4 Examination Methods

none?

Lecture: RSE Lecture Series

SWS: 2 **ECTS:** 3

6.14.5 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
rse_0	describe RSE activities	Computer Science		name x different fields and analyse their differences		Link

6.14.6 Sources & Implementations:

6.14.6.1 Curricula

- None

6.14.6.2 Courses

- None

6.14.6.3 Recommended Course Literature

- None

6.14.6.4 Programs

- None

7 Glossary

C A general-purpose programming language often used for system-level development.

Cpp C++ — an extension of C that supports object-oriented programming.

DIST Software distribution — the practice of packaging and delivering software and its dependencies.

DOCBB Documentation and best practices — ensuring code is understandable and maintainable.

DOMREP Domain repositories — platforms that store and share domain-specific research data.

HPC High-Performance Computing — using supercomputers and parallel processing for complex tasks.

MOD Modularity — the design principle of separating software into interchangeable, functional components.

NEW Novel research — work that contributes original insights to a scientific domain.

PM Project Management — planning, executing, and overseeing projects effectively.

Python A high-level programming language widely used in data science and scripting.

R A programming language and environment for statistical computing and graphics.

RSE Research Software Engineer — someone who applies software engineering skills to scientific research.

SP Software publication — the process of preparing and disseminating software artifacts.

SRU Software reuse — the practice of using existing software components in new projects.

STEM Science, Technology, Engineering, and Mathematics.

SWREPOS Software repositories — systems for storing and managing software code and versions.

TEAM Teamwork — the ability to collaborate effectively in a group setting.

TEACH Teaching — the skill of communicating knowledge and helping others learn.

USERS End users — the scientists or researchers who rely on software tools.