

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 | Design of Efficient Algorithms | 2 | 0 | 2 | 0 | 0 |
| Exercise | Design of 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, Information and Complexity | 2 | 0 | 0 | 2 | 0 |

| Type | Description | SWS | Sem 1 | Sem 2 | Sem 3 | Sem 4 |
|----------|---|-----|-------|-------|-------|-------|
| Exercise | Security, Information and Complexity Exercise | 2 | 0 | 0 | 4 | 0 |
| 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 Security, Information and Complexity

5.2.1 Introduction

This module deals with correctness, security and complexity of algorithms.

5.2.2 Contents

- 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

5.2.3 Learning Objectives

- Understand the mathematical foundations of secure and reliable information processing and their complexity-theoretical basis.
- Are familiar with the fundamentals of error-protected transmission and storage of data.
- Are capable of analysing the correctness, security, and complexity of methods

5.2.4 Examination methods

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

Lecture: Security, Information and Complexity

SWS: 2 **ECTS:** 2

Exercise: Security, Information and Complexity Exercise

SWS: 2 **ECTS:** 4

5.2.5 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 |

5.2.6 Sources & Implementations:

5.2.6.1 Curricula

- [None](#)

5.2.6.2 Courses

- [UP Sicherheit, Information und Komplexität](#)

5.2.6.3 Recommended Course Literature

- [None](#)

5.2.6.4 Programs

- [UP Computational Science Master](#)
- [JMUW Informatik Bachelor](#)
- [JMUW Informatik Master](#)

5.3 Declarative Modelling

5.3.1 Introduction

This module deals with declarative problem-solving.

5.3.2 Contents

Declarative problem-solving methods use general problem-solving methods to automatically solve combinatorial (optimisation) problems. In contrast to traditional programming, no programmes are created to solve the problems; instead, the initial problems are simply modelled (formally). Today, general problem-solving systems are capable of solving problems involving several million variables. The resulting systems are now widely used in industry and the natural sciences. The course focuses on the application of declarative problem-solving methods (especially for solving scientific problems) in the context of modelling and implementation projects.

5.3.3 Learning Objectives

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

5.3.4 Examination Methods

30 minute oral examination.

Lecture: Declarative Modelling

SWS: 2 ECTS: 2

Lab: Declarative Modelling Lab

SWS: 2 ECTS: 4

5.3.5 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|----------------------|---|------------------|---------------|---|-----------------------|----------------------|
| declarative_modeling | 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 |

5.3.6 Sources & Implementations:

5.3.6.1 Curricula

- [None](#)

5.3.6.2 Courses

- [UP Deklarative Modellierung](#)

5.3.6.3 Recommended Course Literature

- [TODO](#)

5.3.6.4 Programs

- [UP Computational Science Master](#)

5.4 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.

5.4.1 Basic Scientific Computing

5.4.1.1 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.

5.4.1.2 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.

5.4.1.3 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 |

5.4.1.4 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.

5.4.1.5 Assessment

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

5.4.1.6 Prerequisites

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

5.4.1.7 Key Resources

ChatGPT fantasy

Lecture: Scientific Computing Basics

SWS: 2 ECTS: 3

Exercise: Scientific Computing Basics Exercise

SWS: 2 ECTS: 3

Lecture: High Performance Computing

SWS: 2 ECTS: 3

Exercise: High Performance Computing Exercise

SWS: 2 ECTS: 3

5.4.2 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-------------|--|----------------------|---------------|--|----------------------|----------------------|
| comp_nule_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_nule_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 |

5.4.3 Sources & Implementations:

5.4.3.1 Curricula

- [EUMaster4HPC](#)

5.4.3.2 Courses

- [Viral Instructions Hardware](#)
- [HPC Computing](#)
- [High-Performance Computing](#)

5.4.3.3 Recommended Course Literature

- [What every computer scientist should know about floating-point arithmetic](#)

5.4.3.4 Programs

- [HPC-carpentry](#)
- [RWTH Informatik Bachelor](#)
- [RWTH Informatik Master](#)

5.5 Current Topics in Artificial Intelligence

5.5.1 Introduction

This module deals with recent research in artificial intelligence.

5.5.2 Contents

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

5.5.3 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.

5.5.4 Examination Methods

- 30 minute presentation with discussion

Seminar: Current topics in artificial intelligence

SWS: 2 **ECTS:** 3

5.5.5 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 |

5.5.6 Sources & Implementations:

5.5.6.1 Curricula

- [None](#)

5.5.6.2 Courses

- [UP Aktuelle Themen der Künstlichen Intelligenz](#)
- [Aktuelle Themen der Informatik](#)

5.5.6.3 Recommended Course Literature

- [None](#)

5.5.6.4 Programs

- [UP Computational Science Master](#)
- [JMUW Informatik Master](#)

5.6 Formal Methods in Software Engineering

5.6.1 Introduction

This module covers advanced topics in Software Engineering with a focus on formal methods for specification, modeling, and verification.

5.6.2 Contents

The module includes the following topics:

- Software Quality Assurance: Formal methods for specifying and verifying system properties.
- Service Engineering: The role of formal methods in service-based architectures.
- System Design: Use of formal methods in system design, focusing on specification and verification.

5.6.3 Learning Objectives

- Understand and apply formal methods in system design and software engineering.
- Analyse theoretical and practical problems in modeling and implementation using formal methods.

5.6.4 Examination Methods

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

Lecture: Formal Methods in Software Engineering Lecture covering formal methods in system design, software specification, and verification. **SWS: 2 ECTS: 2**

Exercise: Formal Methods Exercise Exercise for hands-on application of formal methods in system modeling and analysis. **SWS: 2 ECTS: 4**

5.6.5 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-----------------------|---|------------------|------------------------------------|---|-----------------------|----------------------|
| for- mal_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 |

5.6.6 Sources & Implementations:

5.6.6.1 Curricula

- [None](#)

5.6.6.2 Courses

- [UP Formal Methods in Software Engineering](#)

5.6.6.3 Recommended Course Literature

- [None](#)

5.6.6.4 Programs

- [UP Computational Science Master](#)

5.7 Programming Languages and Compiler Techniques

5.7.1 Introduction

This module provides an in-depth understanding of programming languages and compiler technology. Topics covered include lexical analysis, syntax parsing, code generation, and

optimisation techniques for compilers. The course includes both theoretical lectures and practical exercises, culminating in a project where students will build a simple compiler.

5.7.2 Contents

- Virtualization
- Programming Languages and Design
- Software System Security

5.7.3 Learning Objectives

- describe paradigms and tools for the specification, development, and quality assurance of modern software systems, as well as their application in different contexts
- use various approaches in programming languages and compiler technology

5.7.4 Examination Methods

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

Lecture: Programming Languages and Compiler Technologies

SWS: 2 **ECTS:** 2

Project: Programming Languages and Compiler Technologies Project

SWS: 2 **ECTS:** 4

5.7.5 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|--|---|---------------------|---------------|---|-------------------------------|----------------------|
| com- piler_tech- modelling techniques | apply modelling techniques | Computer Science | | develop solutions through technical methods | Univer- sity of Potsdam | Link |
| com- piler_tech- problems given software systems | analyse problems in given software systems | Computer Science | | discuss problems in a team | Univer- sity of Potsdam | Link |

5.7.6 Sources & Implementations:

5.7.6.1 Curricula

- [None](#)

5.7.6.2 Courses

- [UP Programmiersprachen und Compilertechnologien](#)

5.7.6.3 Recommended Course Literature

- [TODO](#)

5.7.6.4 Programs

- [UP Computational Science Master](#)

5.8 RSE Nuts and Bolts

5.8.1 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).

5.8.2 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

5.8.3 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

5.8.4 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

5.8.5 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 visualization, 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 |

5.8.6 Sources & Implementations:

5.8.6.1 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 \(jupyter book\)](#)

5.9 Design of Efficient Algorithms

5.9.1 Introduction

This module deals with the design of efficient algorithms.

5.9.2 Contents

Architecture and implementation of formal systems for developing efficient algorithms, correctness-preserving optimisation, programme verification and synthesis.

5.9.3 Learning Objectives

Students will be familiar with and understand basic techniques of formal programme development and its implementation.

5.9.4 Examination Methods

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

Lecture: Design of Efficient Algorithms

SWS: 2 **ECTS:** 2

Exercise: Design of Efficient Algorithms Exercise

SWS: 2 **ECTS:** 4

5.9.5 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-------------------|---|------------------|---------------|---|-----------------------|----------------------|
| eff_al-go-rithms_ | 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 |

5.9.6 Sources & Implementations:

5.9.6.1 Curricula

- [None](#)

5.9.6.2 Courses

- [UP Entwurf effizienter Algorithmen](#)

5.9.6.3 Recommended Course Literature

- [TODO](#)

5.9.6.4 Programs

- [UP Computational Science Master](#)

5.10 RSE Philosophy

5.10.1 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.

5.10.2 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

5.10.3 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

5.10.4 Module Competences

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

5.10.5 Sources & Implementations:

5.10.5.1 Courses

- [Philosophische Grundlagen der Wissenschaften](#)
- [Einführung in die Wissenschaftstheorie](#)
- [Ethics, technology, and data](#)

5.10.5.2 Programs

- [RWTH Informatik Bachelor](#)

5.11 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

5.12 RSE Management and Communication

5.12.1 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))

5.12.2 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**

5.12.3 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|--|----------------------------------|-----------------|---|----------------------|----------------------|
| 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 |
| 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_maintenance_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_maintenance_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 |

5.12.4 Sources & Implementations:

5.12.4.1 Courses

- [RSE Leadership Course](#)
- [Professionelles Projektmanagement in der Praxis](#)
- [Communication Psychology](#)
- [Grundlagen des Management](#)
- [Software-Projektmanagement](#)
- [Projektmanagement](#)

5.12.4.2 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](#)

5.12.4.3 Programs

- [RWTH Informatik Bachelor](#)
- [RWTH Informatik Master](#)
- [JMUW Informatik Master](#)

5.13 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

5.13.1 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-----------|-------------|------------------|---------------|----------|--------|----------------------|
| project_1 | | Computer Science | | | | Link |

5.13.2 Sources & Implementations:

5.13.2.1 Curricula

- [None](#)

5.13.2.2 Courses

- [Software-Projektpraktikum](#)
- [Praktikum](#)
- [Softwarepraktikum](#)
- [Praktikum Software Engineering](#)

5.13.2.3 Recommended Course Literature

- [None](#)

5.13.2.4 Programs

- [JMUW Informatik Bachelor](#)
- [JMUW Informatik Master](#)
- [RWTH Informatik Bachelor](#)
- [RWTH Informatik Master](#)

Lecture: Mathematical Foundations of Data Science The module provides mathematical foundations in the field of Data Science. Topics include a selection from the areas of graph analysis, stochastic models, and signal analysis using wavelets. **SWS: 2 ECTS: 2**

Exercise: Mathematical Foundations of Data Science Exercise

SWS: 2 ECTS: 4

5.14 Statistical Data Analysis

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.

Lecture: Statistical Data Analysis

SWS: 4 **ECTS:** 6

5.15 Statistical Data Analysis

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.

Exercise: Statistical Data Analysis Exercise

SWS: 2 **ECTS:** 3

5.16 Text to Data

5.16.1 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
- 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

5.16.2 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

Exercise: Text2Data Exercise

SWS: 2 **ECTS:** 2

5.17 Text to Data

5.17.1 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
- 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

5.17.2 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

Lecture: Text2Data

SWS: 2 **ECTS:** 4

5.17.3 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|--------------------|---|--------------|--------------------|--|-----------------------|----------------------|
| gen_data-science_1 | Assess 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 |
| gen_data-science_4 | 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|--|--------------------------|------------------|--|-----------------------|----------------------|
| gen_statistics_1 | Assess 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 |
| 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 | <p>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</p> | Data Science, Statistics | gen_statistics_2 | Document project workflows demonstrating planning, analysis, evaluation, and use of statistical software tools | University of Potsdam | Link |

5.17.4 Sources & Implementations:

5.17.4.1 Curricula

- [Empfehlungen Masterstudiengänge Data Science](#)

5.17.4.2 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](#)

5.17.4.3 Programs

- [UP Data Science](#)
- [JMUW Informatik Bachelor](#)
- [JMUW Informatik Master](#)
- [RWTH Informatik Bachelor](#)

5.18 RSE Lecture Series

5.18.1 Introduction

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

5.18.2 Contents

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

5.18.3 Learning Objectives

- students get a broad image of RSE in the wild

5.18.4 Examination Methods

none?

Lecture: RSE Lecture Series

SWS: 2 **ECTS:** 3

5.18.5 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 |

5.18.6 Sources & Implementations:

5.18.6.1 Curricula

- [None](#)

5.18.6.2 Courses

- [None](#)

5.18.6.3 Recommended Course Literature

- [None](#)

5.18.6.4 Programs

- [None](#)

5.19 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

5.19.1 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|--|----------------------------------|----------------|---|----------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|---|-------------------------------------|-----------------|---|----------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|---|--|---------------|---|----------------------|----------------------|
| 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 |

5.19.2 Sources & Implementations:

5.19.2.1 Programs

- [UP Computational Science Master](#)

5.20 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>.

5.21 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

5.22 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

5.22.1 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-------------------|---|------------------|--------------------------------------|---|-----------------------|----------------------|
| 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 |
| 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 |

5.22.2 Sources & Implementations:

5.22.2.1 Curricula

- [Computing Curricula 2020](#)

5.22.2.2 Courses

- [Software Engineering I](#)
- [Software Engineering](#)
- [Softwaretechnik](#)

5.22.2.3 Programs

- [UP Computational Science Master](#)
- [JMUW Informatik Bachelor](#)
- [RWTH Informatik Bachelor](#)

5.23 Example Module: Fundamentals of Computer Science

This is an example module to showcase the integration pipeline

5.23.1 Basics of Computer Science

5.23.1.1 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

5.23.2 Applied Programming

5.23.2.1 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

5.23.2.2 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

5.23.3 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-------------------------|--|------------------|------------------------|---|-----------------------|----------------------|
| ex_pro-gramming_model_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_pro-gramming_model_2 | Implement basic data structures and algorithms | Computer Science | ex_programming_model_1 | Submit a project with implemented algorithms and data structures (e.g., lists, trees, sorting) | University of Potsdam | Link |
| ex_pro-gramming_model_3 | Distinguish between error types and handle them appropriately in code | Computer Science | ex_programming_model_1 | Demonstrate error handling techniques in submitted code (e.g., input validation, error codes, exceptions) | University of Potsdam | Link |
| ex_pro-gramming_model_4 | Identify and use appropriate library functions in programming tasks | Computer Science | ex_programming_model_1 | Integrate external libraries in coding tasks and document their usage | University of Potsdam | Link |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------------|--|------------------|-----------------------|---|-----------------------|----------------------|
| ex_pro-gramming_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_pro-gramming_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_pro-gramming_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_pro-gramming_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_model_1 | Distinguish between programming paradigms and identify their characteristics | Computer Science | ex_programming_model_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 |

5.23.4 Sources & Implementations:

5.23.4.1 Curricula

- [Computing Curricula 2020](#)

5.23.4.2 Courses

- [Grundlagen der Programmierung](#)
- [Praxis der Programmierung](#)
- [Python for Psychologists](#)
- [Grundlagen der Informatik](#)

- [Grundlagen der Programmierung](#)
- [Programmierung](#)

5.23.4.3 Programs

- [UP Computational Science Master](#)
- [JMUW Informatik Bachelor](#)
- [RWTH Informatik Bachelor](#)

5.24 Distributed Systems

5.24.1 Introduction

This module deals with distributed IT systems.

5.24.2 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.

5.24.3 Learning Objectives

- 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.

5.24.4 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

5.24.5 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-----------------------------|---|------------------|---------------|---|-----------------------|----------------------|
| dist_sys_evaluate tems_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 |
| dist_sys_identify tems_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 |

5.24.6 Sources & Implementations:

5.24.6.1 Curricula

- [None](#)

5.24.6.2 Courses

- [Verteilte Systeme](#)
- [Theory of Distributed Systems](#)

5.24.6.3 Recommended Course Literature

- [None](#)

5.24.6.4 Programs

- [UP Computational Science Master](#)
- [RWTH Informatik Master](#)

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 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

6.2.1 Module Competences

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

6.2.2 Sources & Implementations:

6.2.2.1 Programs

- [Uni Würzburg Master Physik](#)
- [Uni Würzburg Master Chemie](#)

6.3 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.

6.3.1 Basic Scientific Computing

6.3.1.1 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.

6.3.1.2 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.

6.3.1.3 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 |

6.3.1.4 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.

6.3.1.5 Assessment

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

6.3.1.6 Prerequisites

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

6.3.1.7 Key Resources

ChatGPT fantasy

Lecture: Scientific Computing Basics

SWS: 2 **ECTS:** 3

Exercise: Scientific Computing Basics Exercise

SWS: 2 **ECTS:** 3

Lecture: High Performance Computing

SWS: 2 **ECTS:** 3

Exercise: High Performance Computing Exercise

SWS: 2 **ECTS:** 3

6.3.2 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_nule_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 |

6.3.3 Sources & Implementations:

6.3.3.1 Curricula

- [EUMaster4HPC](#)

6.3.3.2 Courses

- [Viral Instructions Hardware](#)
- [HPC Computing](#)

- [High-Performance Computing](#)

6.3.3.3 Recommended Course Literature

- [What every computer scientist should know about floating-point arithmetic](#)

6.3.3.4 Programs

- [HPC-carpentry](#)
- [RWTH Informatik Bachelor](#)
- [RWTH Informatik Master](#)

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

- [Uni Potsdam Master Bioinformatik](#)

6.5 RSE Nuts and Bolts

6.5.1 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).

6.5.2 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

6.5.3 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

6.5.4 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

6.5.5 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 visualization, 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|---------------|--|-------------------------------------|----------------------------------|---|--------------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|----------------|--|----------------------------------|---------------|---|--------------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|----------------|---|-------------------------------------|---|--|--------------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|----------------|--|-------------------------------------|---------------|---|-----------------------|----------------------|
| 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 |

6.5.6 Sources & Implementations:

6.5.6.1 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 \(jupyter book\)](#)

6.6 RSE Philosophy

6.6.1 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.

6.6.2 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

6.6.3 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

6.6.4 Module Competences

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

6.6.5 Sources & Implementations:

6.6.5.1 Courses

- [Philosophische Grundlagen der Wissenschaften](#)
- [Einführung in die Wissenschaftstheorie](#)
- [Ethics, technology, and data](#)

6.6.5.2 Programs

- [RWTH Informatik Bachelor](#)

6.7 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

6.8 RSE Management and Communication

6.8.1 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))

6.8.2 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**

6.8.3 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_maintenance_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_maintenance_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 |

6.8.4 Sources & Implementations:

6.8.4.1 Courses

- [RSE Leadership Course](#)
- [Professionelles Projektmanagement in der Praxis](#)
- [Communication Psychology](#)
- [Grundlagen des Management](#)
- [Software-Projektmanagement](#)
- [Projektmanagement](#)

6.8.4.2 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](#)

6.8.4.3 Programs

- [RWTH Informatik Bachelor](#)
- [RWTH Informatik Master](#)
- [JMUW Informatik Master](#)

Lecture: Mathematical Foundations of Data Science The module provides mathematical foundations in the field of Data Science. Topics include a selection from the areas of graph analysis, stochastic models, and signal analysis using wavelets. **SWS: 2 ECTS: 2**

Exercise: Mathematical Foundations of Data Science Exercise

SWS: 2 ECTS: 4

6.9 Statistical Data Analysis

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.

Lecture: Statistical Data Analysis

SWS: 4 ECTS: 6

6.10 Statistical Data Analysis

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.

Exercise: Statistical Data Analysis Exercise

SWS: 2 **ECTS:** 3

6.11 Text to Data

6.11.1 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
- 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

6.11.2 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

Exercise: Text2Data Exercise

SWS: 2 **ECTS:** 2

6.12 Text to Data

6.12.1 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
- 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

6.12.2 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

Lecture: Text2Data

SWS: 2 **ECTS:** 4

6.12.3 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|--------------------|---|--------------|--------------------|---|-----------------------|----------------------|
| gen_data-science_1 | Assess 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|--------------------|---|--------------------------|--------------------|--|-----------------------|----------------------|
| 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 |
| gen_data-science_4 | 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 | Assess 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|--|--------------------------|------------------|--|-----------------------|----------------------|
| 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 |
| 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 | <p>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</p> | Data Science, Statistics | gen_statistics_2 | Document project workflows demonstrating planning, analysis, evaluation, and use of statistical software tools | University of Potsdam | Link |

6.12.4 Sources & Implementations:

6.12.4.1 Curricula

- [Empfehlungen Masterstudiengänge Data Science](#)

6.12.4.2 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](#)

6.12.4.3 Programs

- [UP Data Science](#)
- [JMUW Informatik Bachelor](#)
- [JMUW Informatik Master](#)
- [RWTH Informatik Bachelor](#)

6.13 RSE Lecture Series

6.13.1 Introduction

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

6.13.2 Contents

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

6.13.3 Learning Objectives

- students get a broad image of RSE in the wild

6.13.4 Examination Methods

none?

Lecture: RSE Lecture Series

SWS: 2 **ECTS:** 3

6.13.5 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 |

6.13.6 Sources & Implementations:

6.13.6.1 Curricula

- [None](#)

6.13.6.2 Courses

- [None](#)

6.13.6.3 Recommended Course Literature

- [None](#)

6.13.6.4 Programs

- [None](#)

6.14 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

6.14.1 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|--|----------------------------------|----------------|---|----------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|---|-------------------------------------|-----------------|---|----------------------|----------------------|
| 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------|---|--|---------------|---|----------------------|----------------------|
| 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 |

6.14.2 Sources & Implementations:

6.14.2.1 Programs

- [UP Computational Science Master](#)

6.15 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>.

6.16 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

6.17 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

6.17.1 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 |
| 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 |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-------------------|---|------------------|--------------------------------------|---|-----------------------|----------------------|
| 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 |
| 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 |

6.17.2 Sources & Implementations:

6.17.2.1 Curricula

- [Computing Curricula 2020](#)

6.17.2.2 Courses

- [Software Engineering I](#)
- [Software Engineering](#)
- [Softwaretechnik](#)

6.17.2.3 Programs

- [UP Computational Science Master](#)
- [JMUW Informatik Bachelor](#)
- [RWTH Informatik Bachelor](#)

6.18 Example Module: Fundamentals of Computer Science

This is an example module to showcase the integration pipeline

6.18.1 Basics of Computer Science

6.18.1.1 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

6.18.2 Applied Programming

6.18.2.1 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

6.18.2.2 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

6.18.3 Module Competences

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|-------------------------|--|------------------|------------------------|---|-----------------------|----------------------|
| ex_pro-gramming_model_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_pro-gramming_model_2 | Implement basic data structures and algorithms | Computer Science | ex_programming_model_1 | Submit a project with implemented algorithms and data structures (e.g., lists, trees, sorting) | University of Potsdam | Link |
| ex_pro-gramming_model_3 | Distinguish between error types and handle them appropriately in code | Computer Science | ex_programming_model_1 | Demonstrate error handling techniques in submitted code (e.g., input validation, error codes, exceptions) | University of Potsdam | Link |
| ex_pro-gramming_model_4 | Identify and use appropriate library functions in programming tasks | Computer Science | ex_programming_model_1 | Integrate external libraries in coding tasks and document their usage | University of Potsdam | Link |

| ID | Description | Disciplines | Prerequisites | Evidence | Author | Source |
|------------------------|--|------------------|-----------------------|---|-----------------------|----------------------|
| ex_pro-gramming_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_pro-gramming_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_pro-gramming_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_pro-gramming_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_model_0 | 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 |

6.18.4 Sources & Implementations:

6.18.4.1 Curricula

- [Computing Curricula 2020](#)

6.18.4.2 Courses

- [Grundlagen der Programmierung](#)
- [Praxis der Programmierung](#)
- [Python for Psychologists](#)
- [Grundlagen der Informatik](#)

- [Grundlagen der Programmierung](#)
- [Programmierung](#)

6.18.4.3 Programs

- [UP Computational Science Master](#)
- [JMUW Informatik Bachelor](#)
- [RWTH Informatik Bachelor](#)

6.19 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

6.19.1 Module Competences

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

6.19.2 Sources & Implementations:

6.19.2.1 Courses

- [Computational Chemistry](#)
- [Computational Physics](#)

6.19.2.2 Programs

- [Uni Potsdam Master Bioinformatik](#)
- [RWTH Informatik Bachelor](#)
- [RWTH Informatik Master](#)

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.