RSE Curriculum

Gesellschaft für Informatik deRSE Julian Dehne
Florian Goth Jan Phillip Thiele Anna-Lena Lambrecht

1 WORK IN PROGRESS THIS IS NOT THE OFFICIAL STATEMENT OF THE COMMUNTIY 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.

2.1 Research skills

TODO add text here

Research skills are implemented in the following components:

- mnt_project (TODO work/elaborate on naming, add cross-reference)
- mnt_wildcard
- rse_thesis

Technical skills are implemented in:

- gen_datascience
- gen_programming
- gen_softwareengineering
- rse_softwareengineering
- rse_programming

(TODO check if technical training assumes too big a role)

communication skills are implemented in:

- rse_management
- mnt_project
- rse_theory

3 Ideas

Electronic Lab course. Heard of this in Erlangen for physics. Talks about ELN among other things.

3.1 Original Motivation

The target audience for such a master's programme would be students holding a bachelor's degree from a domain science, which we will call **home domain** in the following.

There is explicitly no restriction on the candidates' home domain: it may be from the STEM disciplines, life sciences, humanities or social sciences. Candidates with a bachelor's degree in computer science are also explicitly included, although we acknowledge that their master's programme should include adaptations to make their interaction effective with other domain scientists.

In order to give the future RSE the necessary breadth, we expect this to be a four semester curriculum.

The curriculum is formed from a combination of modules, some of which are core modules teaching essential skills that must be completed by all students. Other modules introduce more specialised concepts and skills.

During the master's programme, students should pick an RSE specialisation from the list in this paper and attend these additional modules to deepen their knowledge in the field.

Core modules are of course drawn from the three pillars of the RSE and can be categorised accordingly.

3.1.1 Software / Technical Skills

- **Foundational module**
  Introduction to programming: Emphasising use cases over programming paradigms, students learn at least two languages:
  - A language that facilitates prototyping and data processing e.g., Python or R

  - A language for designing complex, performance-critical systems e.g., C/Cpp
    This exposes them to computers in a hands-on fashion and is the foundation for DOCBB, DIST.

- **Computing environment module**
  Programming languages are not enough to work in a landscape of many interconnected software components. Hence, we require something like software craftsmanship:
  - Tools: Unix shell, version control systems, build systems, documentation generators, package distribution platforms, and software discovery systems
    This strengthens skills in DIST, DOCBB, SWREPOS, SRU.

- **Software engineering module**
  Develop foundational software engineering competencies:
  - Requirements engineering
  - Software architecture and design
  - Implementation, quality assurance, and software evolution
    Emphasising and strengthening DOCBB, DIST on a more abstract level.

3.1.2 Research Skills

- **Optional domain mastery module**
  Additional minor research courses; students with a home-domain already have the research part well-covered.

- **Research tools module**
  Teach tools used to distribute and publish software, and introduce domain-specific data repositories, gaining foundational knowledge in SRU, SP, DOMREP.

- **Meta-research module**
  Teach how research works: Introduce the research life cycle, the data life cycle, and the software life cycle abstractly.

3.1.3 Communication Skills

- **Project management methods**
  Teach project management methods that are useful in science, such as agile ones PM.

- **Communication skills module**
  Courses focusing on:
  - Interdisciplinary communication
  - Interacting across cultures
  - Communication in hierarchies

  - Supporting end users effectively
    All facets of the USERS skill.

- **Teaching module**
  Covers topics to effectively design courses and teaching material for various digital tools, strengthening the TEACH skill.

3.2 Hands-On Practice

RSE work also involves craftsmanship skills. Hands-on practice is integral.

- At least two **lab projects** are required within the mandatory curriculum.
- These should be team-based and involve a question from a domain science.
- Ideally, projects cover both the candidate's home domain and another domain.
- Projects should stem from collaborations with scientists within the institution, with RSE students taking on a consultant role.

This setup strengthens TEAM, TEACH, USERS and likely also MOD through interaction.

To emphasise exposure beyond their bachelor's domain, RSEs should support their non-home-domain project with introductory courses from that discipline. This encourages adapting vocabulary and thinking—an aspect of MOD.

3.3 Optional Modules and Specialisations

To align with the specialisations listed in this paper, example optional modules include:

- HPC engineering / parallel programming
- Numerical mathematics / scientific computing
- Web technologies
- Data stewardship
- AI models / statistics
- Community management / training

3.4 Master's Thesis

The programme concludes with a master's thesis that should:

- Be dual-supervised by an RSE project supervisor and a domain supervisor
- Answer a relevant research question strengthening NEW using computational methods

- Include software development as a required, gradable deliverable

The RSE supervisor ensures and grades the software craftsmanship. This ensures the effective application of RSE skills in an actual research environment.

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 General Study Process

5.1 Semester 1

Type	Description	SWS	ECTS
Seminar	RSE Nuts and Bolts I	2	3
Lecture	Wildcard Science I	2	3
Lecture	Basic Programming	2	1
Exercise	Basic Programming Exercise	4	4
Lecture	Mathematical Foundations of Data Science	4	6

Total ECTS: 17

5.2 Semester 2

Type	Description	SWS	ECTS
Lecture	Applied Programming	2	1
Exercise	Applied Programming Exercise	4	4
Lecture	Wildcard Science II	2	3
Lab	Wildcard Science Lab I	4	6
Lecture	Statistical Data Analysis	4	4
Lecture	Scientific Computing Basics	2	3
Exercise	Scientific Computing Basics Exercise	2	3

Total ECTS: 24

5.3 Semester 3

Type	Description	SWS	ECTS
Seminar	RSE Nuts and Bolts II	2	3
Lab	Wildcard Science Lab II	2	2
Exercise	Text2Data	4	4
Lecture	Computational Wildcard Science	2	3
Lecture	Software Engineering I	2	4
Exercise	Software Engineering I Exercise	2	2
Lecture	High Performance Computing	2	3
Exercise	High Performance Computing Exercise	2	3

Total ECTS: 24

5.4 Semester 4

Type	Description	SWS	ECTS
Thesis	RSE Master Thesis	10	30

Total ECTS: 30

Total Curriculum ECTS: 95

6 Complete Competences Table

TODO: replace this with Excel download in html and possibly remove for pdf

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
C01	Use a version control system to track software changes	CS, Bioinformatics		Push a merge request with documented code	Florian Goth	https://github.com/the-teachingRSE-project/RSE-Masters
C02	Conduct a ReproHack on domain-specific data	Physics, CS	C01	Submit a ReproHack report	Florian Goth	https://github.com/the-teachingRSE-project/RSE-Masters

7 Module Descriptions (Inline)

7.1 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:** 3

Lecture: Wildcard Science II

SWS: 2 **ECTS:** 3

Lab: Wildcard Science Lab I

SWS: 4 **ECTS:** 6

Lab: Wildcard Science Lab II

SWS: 2 **ECTS:** 2

7.2 Science Lab Module

Applied Research Software Engineering in MINT 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

SWS: 4 **ECTS:** 6

7.3 Example Module: Fundamentals of Computer Science

This is an example module to showcase the integration pipeline

7.3.1 Basics of Computer Science

7.3.1.1 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:** 1

Exercise: Basic Programming Exercise

SWS: 4 **ECTS:** 4

7.3.2 Applied Programming

7.3.2.1 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

7.3.2.2 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:** 1

Exercise: Applied Programming Exercise

SWS: 4 **ECTS:** 4

7.4 Module Competences

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

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_C5	Use basic functions and mechanisms of operating systems using UNIX/Linux as an example	Computer Science		Demonstrate file handling, permissions, and process control using UNIX/Linux commands	University of Potsdam	Link
ex_programming_mod1_C6	Create and refine simple algorithms using semi-formal notation	Computer Science		Submit pseudocode or flowcharts for given algorithmic problems	University of Potsdam	Link
ex_programming_mod1_C7	Evaluate and compare algorithms using runtime analysis	Computer Science	ex_programming_mod1_C6	Provide time complexity comparisons for multiple algorithmic solutions	University of Potsdam	Link
ex_programming_mod1_C8	Implement simple algorithms using imperative and functional programming styles (e.g., in Python)	Computer Science	ex_programming_mod1_C6	Submit code demonstrating both imperative and functional styles for the same problem	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
ex_programming_mod1_C9	Distinguish between programming paradigms and identify their characteristics	Computer Science	ex_programming_mod1	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

7.5 Sources & Implementations:

7.5.1 Curricula

- Computing Curricula 2020

7.5.2 Courses

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

- Grundlagen der Informatik

7.5.3 Programs

- UP Computational Science Master

8 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: Wildcard Science Lab

SWS: 4 **ECTS:** 6

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).

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

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

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.

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

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 visualisation, web tasks, and templating	Workshop Participants	Link
rse_tooling_3	Write and use Bash scripts for automation	Research Software Engineering		Submit shell scripts automating file manipulation or computational workflows	Workshop Participants	Link
rse_tooling_4	Apply testing, debugging, and logging techniques to ensure software reliability	Research Software Engineering	rse_tooling_2	Submit logs, test cases, and debugging documentation for a non-trivial Python or Bash project	Workshop Participants	Link
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

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
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

8.6 Sources & Implementations:

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)

8.7 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.

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.

8.8 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

8.9 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:** 4

Exercise: Software Engineering II Exercise

SWS: 2 **ECTS:** 2

8.10 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	Identifying 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

8.11 Sources & Implementations:

8.11.1 Curricula

- Computing Curricula 2020

8.11.2 Courses

- Software Engineering I

8.11.3 Programs

- UP Computational Science 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:** 4 **ECTS:** 6

8.12 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:** 4

Exercise: Data-oriented Programming

SWS: 4 **ECTS:** 6

Exercise: Text2Data

SWS: 4 **ECTS:** 4

8.13 Module Competences

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_datascience_1	Possess comprehensive, detailed, and specialized knowledge of selected fundamentals in the field of Data Science	Data Science		Demonstrate knowledge through theoretical exams and practical assignments	University of Potsdam	Link
gen_datascience_2	Demonstrate an in-depth understanding of selected Data Science methods	Data Science	gen_datascience_1	Apply Data Science methods in practical projects and case studies	University of Potsdam	Link
gen_datascience_3	Analyze novel data assimilation and inference problems, develop and implement solutions, and assess solution quality	Data Science	gen_datascience_2	Solve complex inference problems and present implemented solutions with evaluation	University of Potsdam	Link

ID	Description	Disciplines	Prerequisites	Evidence	Author	Source
gen_datascience_4	Develop new ideas and methods, weigh alternatives under incomplete information, and evaluate them considering different evaluation criteria	Data Science	gen_datascience_2	Present projects showcasing creative problem-solving and alternative evaluations under uncertainty	University of Potsdam	Link
gen_statistics_1	Possess comprehensive, detailed, and specialized understanding of the linear regression model based on the latest research	Data Science, Statistics		Apply linear regression models to practical problems and interpret results	University of Potsdam	Link
gen_statistics_2	Understand fundamental concepts and methods of nonparametric statistics	Data Science, Statistics	gen_statistics_1	Solve problems involving nonparametric methods and explain applied techniques	University of Potsdam	Link

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

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

8.14 Sources & Implementations:

8.14.1 Curricula

- Empfehlungen Masterstudiengänge Data Science

8.14.2 Courses

- Statistical Data Analysis

- Mathematical Foundations of Data Science
- Programmieren für Data Scientists Python

8.14.3 Programs

- UP Data Science

8.15 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.

8.15.1 Basic Scientific Computing

8.15.1.1 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.

8.15.1.2 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.

8.15.1.3 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

8.15.1.4 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.

8.15.1.5 Assessment

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

8.15.1.6 Prerequisites

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

8.15.1.7 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

8.16 Module Competences

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

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

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

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

8.17 Sources & Implementations:

8.17.1 Curricula

- EUMaster4HPC

8.17.2 Courses

- Viral Instructions Hardware
- HPC Computing

8.17.3 Programs

- HPC-carpentry

8.18 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: 10 **ECTS:** 30

9 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.