[BHR26] C. Buschhaus, A. Hellwig, B. Rumpe:
WiRCA: Categorizing Research Software Code Based on Its Purpose.
In: Proceedings of the 2026 1st International Workshop on Software Engineering and Research Software, pp. 17–19, SERS '26,
DOI 10.1145/3786172.3788366, ACM, New York, NY, USA, Jul. 2026.

WiRCA: Categorizing Research Software Code Based on its Purpose

Constantin Buschhaus
RWTH Aachen University
Aachen, Germany
buschhaus@se-rwth.de

Alexander Hellwig
RWTH Aachen University
Aachen, Germany
hellwig@se-rwth.de

Bernhard Rumpe
RWTH Aachen University
Aachen, Germany
rumpe@se-rwth.de

Abstract

Software increasingly drives scientific contributions in research and must therefore be maintainable. It remains a significant challenge to effectively apply existing software engineering techniques for maintainability, such as separation of concerns, to long running research projects. While established methodologies offer metrics and tools to tackle separation of concerns in classical software, previous work has failed to address the unique constraints of the Research Software Engineering (RSE) domain. The WiRCA project proposes a novel approach that classifies research code based on its purpose to transfer and adapt the separation of concerns principles into RSE. We validated this approach by developing a demonstrator and applying it to the research software GemPy as a case study. Overall, this methodology and its associated tools can be used by researchers to improve the maintainability of their software projects.

CCS Concepts

• **Software and its engineering** → *Maintaining software.*

Keywords

Code Classification, Research Software Engineering, Software Maintainability

ACM Reference Format:
Constantin Buschhaus, Alexander Hellwig, and Bernhard Rumpe. 2026. WiRCA: Categorizing Research Software Code Based on its Purpose. In *1st International Workshop on Software Engineering and Research Software (SERS '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3786172.3788366

1 Introduction

In software engineering, the separation of concerns is an important measure for the maintainability of a project. Changes to an artifact cannot be made until all the concerns contained within it are understood. This problem is aggravated in RSE, since a high portion of researchers lack software engineering training [5], but increasingly write complex research software. Thus, they might be more prone to mix several concerns in one artifact, such as scientific code and technologic code. Additionally, they oftentimes lack the background training to understand technical aspects of a project, such as the database integration. The partitioning of research software projects into technologic and scientific code is therefore essential to help researchers focus on their domain.

For classical software engineering the basic concept of separating code based on its purpose and a accompanying methodology was already demonstrated by Quasar [4]. The project *Wissenschaftsinduzierte Retrospektive Code-Analyse(WiRCA)*, a RWTH-internal cooperation between the Chair of Science Theory and the Chair of Software Engineering, aims to apply this separation to research software. It focuses on identifying mixed purpose, hard to understand artifacts, which should be refactored to improve the project structure and thus maintainability. The resulting methodology and tooling has been implemented in a demonstrator and was applied to GemPy [2], a research software project for the 3D modeling and analysis of geological structures.

2 Code Categorization for RSE

Siedersleben proposed the idea to categorize code into the following categories for classical software engineering: domain, technologic, 0 code [3]. Additionally, code of these categories can be mixed resulting in two more categories: mixed code and, so called R code. Figure 1 shows how these categories overlap in an RSE project. The names are based on a blood-group metaphor to highlight the issues that arise when different code categories are mixed.

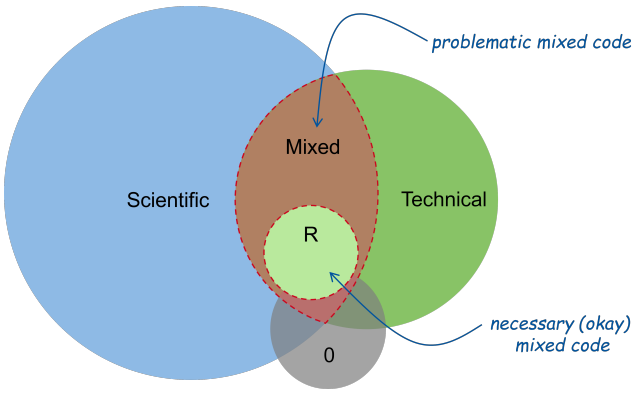

Figure 1: Code categories in an RSE project

In the following, we explain these categories and fit them into the context of RSE. Before that, we have to explain the terms **Technology** and **Scientific Domain**. A technology is a part of the environment in which a software runs in. Oftentimes, it is required infrastructure for the code, *e.g.*, the operating system or a database. In contrast, the scientific domain of an RSE project changes depending on the research topic, *e.g.*, physics, geology, or climate research.

SERS '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2404-6/26/04
https://doi.org/10.1145/3786172.3788366

Scientific Code (S) is code that is independent of technologies and dependent on the scientific domain. It often involves the manipulation of domain specific (scientific) data, *e.g.,* the calculation of a prediction based on measured data. **Technologic Code (T)** is not dependent on any scientific domain but dependent on a specific technology. The code can either be part of a technology or it constitutes as an interface to this technology, *e.g.,* code interacting with the operating system by reading a file. **Mixed Code (ST)** is dependent on the scientific domain and a technology. Usually mixed code is an indication of low cohesion and should be avoided. **R Code (R)** is a subcategory of mixed code. It comprises of code which translates between the scientific domain and a technology, which is unavoidable and thus cannot be considered harmful. Data import and export oftentimes results in R Code, *e.g.,* when importing from technologic data formats such as csv or json into a scientific object structure. **0-Code (0)** is code which is neither dependent on any technology nor any scientific domain. This code is easily reusable, but has no purpose on its own, *e.g.,* String manipulation code or vector/matrix manipulations that are independent of a domain or technology. 0-code can be used in technologic code and scientific code without causing problems.

To separate research code into these categories a hierarchical approach is used, that starts by classifying atomic code sections that are either logically or syntactically connected, such as methods or attributes. They are classified based on indicators for technological or scientific content, such as use of domain-specific vocabulary or usage of certain libraries. Scientific sections in particular can be identified by looking at citations of relevant papers in the comments. Other programming language constructs, *e.g.,* classes and modules, are hierarchically composed of these atomic snippets. Thus, their overall category can be determined by aggregating the categorizations of their components.

The aggregation of the categories defined in this approach focuses on the identification of mixed code and is defined below: If all sub-elements of an element are assigned to the same category, then the composed element is assigned to this category, too. If the sub-elements are of different categories, the composed element is considered mixed code, which should be avoided. The exception is the category 0-code, which can be added without effecting the aggregated category. In case that any of the comprising elements are mixed code, the composed element is also considered mixed.

The hierarchical nature of this approach allows for the identification of the smallest sections that are not properly separated by concern. For a high-level overview of the project and large modules, the portion of code applied to each category can be calculated. These insights can then be used to restructure the RSE project and improve its evolution.

For an application of this methodology consider Listing 1, which shows abbreviated Python code for climate prediction. The excerpt contains one class that consists of 3 functions. The scientific domain is climate research and two technologies are used: `pandas`, a common data processing library (l. 9 and 14), and `Excel` (l. 15). First, we categorize the functions in lines 4, 9, and 14, to then categorize the composed class. The function `climate_prediction` is scientific, because it implements a scientific algorithm of the domain. The function `read_data_from_excel` is technologic. It reads data from the technology Excel and transforms it into data of another

```
1  # Mixed category, since it is composed of scientific,
       technical, and R-Code
2  class ClimatePredictor:
3    # Scientific method
4    def climate_prediction():
5      """Implement algorithm from Leith AGCM"""
6      ...
7
8    # R-Code converting a dataframe(technical) to the
         internal data structure(scientific)
9    def historic_data(self, dataframe):
10     self.weather_data = transform_data(dataframe)
11     ...
12
13   # Technical code for interacting with excel
14   def read_data_from_excel(path) -> pandas.DataFrame:
15     res = pandas.ExcelFile.parse(path)
16     ...
```

Listing 1: Exemplary code excerpt containing different categories of code.

technology (pandas) independent of climate research information. Lastly, the function `historic_data` is categorized as R-code, because it converts a dataframe (technologic) into an internal data representation of weather data (scientific).

Since the `ClimatePredictor` class consists of scientific, technologic, and R (mixed) code, it itself is also categorized as mixed.

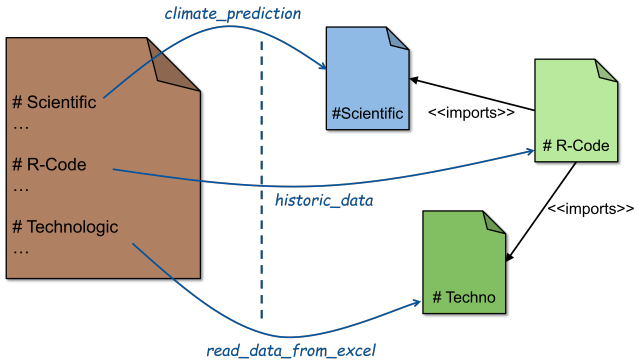

Figure 2: Refactoring of code from Listing 1

Our methodology now suggests to refactor the code by moving `read_data_from_excel` into another artifact. This will also improve reusability, since handling excel imports is probably also relevant in other parts of the project. For the `ClimatePredictor` class to be categorized scientific, we also have to remove the `historic_data` function. The result of this refactoring can be seen in Figure 2. This separation ensures that scientists only have to understand scientific code to understand the `ClimatePredictor` class, which improves its comprehensibility. Additionally, it will facilitate replacing the data importing technology in the future, since the scientific code can be left untouched.

3 WiRCA-Demonstrator Tool

To support our proposed methodology with tooling, we implemented a demonstrator that (1) automatically categorizes python code of a given RSE project, and (2) displays it in an interactive, explorable dashboard, and (3) allows researchers to manually refine

the initial automatic classification. The demonstrator is published and available as open-source software [1]. Internally the tool parses python code into its abstract syntax, *i.e.,* classes, functions, comments. These elements are then used to perform the hierarchical categorization as described above. Since it is impossible to define hard criteria for each of the categories, a Large Language Model (LLM) is used to automatically categorize the atomic elements.

To explore the categorization, the dashboard contains a graph showing the categories of the Python artifacts and their connections. Researchers can identify clusters of certain code categories, and most importantly outliers, that should be refactored. Additionally, the hierarchal classification of each artifact can be navigated to (1) manually refine the automatically assigned categories or (2) identify problematic mixed purpose code. Thus, the demonstrator allows the gradual adaptation of the categorization on the atomic and composed levels. In the future, we want to investigate how well the automatic categorization with LLM works and if our methodology actually provides helpful refactoring advice.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 459671966

References

[1] Constantin Buschhaus, Alexander Hellwig, Bernhard Rumpe. 2025. *WiRCA Demonstrator Tool.* Chair of Software Engineering, RWTH Aachen University, Aachen, Germany. doi:10.5281/zenodo.17292384

[2] Miguel De la Varga, Alexander Schaaf, and Florian Wellmann. 2019. GemPy 1.0: open-source stochastic geological modeling and inversion. *Geoscientific Model Development* 12, 1 (2019), 1–32.

[3] Johannes Siedersleben. 2003. Softwaretechnik. *Praxiswissen für Softwareingenieure. Hanser, München Wien* (2003).

[4] Johannes Siedersleben. 2004. *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar.* dpunkt.verlag, Heidelberg, Germany.

[5] Igor Wiese, Ivanilton Polato, and Gustavo Pinto. 2020. Naming the Pain in Developing Scientific Software. *IEEE Software* 37, 4 (2020), 75–82. doi:10.1109/MS.2019.2899838