Adapting Established Software Engineering Techniques and Technologies into an Assistance System for Neuroscientists

Thorsten Arendt and Alexander C. Schütz
Department of Psychology, Philipps-Universität Marburg, Germany

In the information infrastructure project NOWA (NeurOscientific Workflow Assistance) of the collaborative research center CRC/TRR 135 *Cardinal mechanisms of perception: Prediction, Valuation, Categorization* we develop and combine tools for the sharing of research data. In this context, NOWA aims to create an organizational and technological framework that supports workflows throughout the entire research data lifecycle. This article addresses the conceptual and technical part of NOWA. We first discuss the specific requirements of the scientists and then compare them with proven solutions from the computer science and software engineering domain, respectively. These so-called best practices are then adapted to the specific needs within the CRC. Finally, we conclude the article by an overview of the planned technical and technological implementation of the NOWA assistance system.

1. NeurOscientific Workflow Assistance (NOWA)

The sensory organs are the "window to the world" since they enable the sensation of signals from the environment. But how does the brain process this information? How does perception work? The DFG funded collaborative research center CRC/TRR 135 *Cardinal Mechanisms of Perception: Prediction, Evaluation, Categorization* deals particularly with these questions. More precisely, twenty interdisciplinary working groups at the Justus-Liebig-Universität Gießen (JLU) and the Philipps-Universität Marburg (UMR) use a combination of behavioral experiments, physiological measurements and computational modeling to gain a comprehensive understanding of prediction, evaluation and categorization. The goal is to delineate these cardinal mechanisms behaviorally, to identify their underlying neural substrates and to explain their functions with computational models.

Already during the first funding phase of the CRC, the researchers increasingly pursued the implementation of the Open Science principles. In addition to the publication of research results in peer-reviewed articles, the researchers published relevant research data (Open Access) using the publicly accessible repository Zenodo[1] (Open Data). Doing this, they increased traceability and reproducibility of their results and so the overall transparency of their research. However, the high effort when selecting and packaging the research data and creating the meta data revealed the need for a more fundamental

[1] `https://zenodo.org/`

Thorsten Arendt und Alexander C. Schütz: Adapting Established Software Engineering Techniques and Technologies into an Assistance System for Neuroscientists. In: Heuveline, V., Gebhart, F. und Mohammadianbisheh, N. (Hrsg.): E-Science-Tage 2019 – Data to Knowledge, Heidelberg: heiBOOKS, 2020, 32–46. DOI: https://doi.org/10.11588/heibooks.598.c8415

approach. For the second funding phase, this led to the information infrastructure project NOWA (NeurOscientific Workflow Assistance) [2], which is in the focus of this article.

In NOWA we develop new and combine existing tools for the sharing of research data. In order to enhance reproducibility of all steps in the lifecycle of a scientific study – from planning experiments over collecting and analysing data to publishing them– this information infrastructure project aims at creating an organizational and technological framework, supporting workflows along the entire data lifecycle. The long-term goal is to share research data within both, the single working groups and the entire CRC, right from the beginning of the study and to make them publicly accessible at the "press of a button" at the time the results are published. NOWA is implemented in close cooperation with the researchers of the CRC and the local infrastructure facilities including the joint project HeFDI (Hessische Forschungsdateninfrastrukturen).[2] The University Computer Center of the UMR thereby provides the technical infrastructure.

This article particularly addresses the conceptual and technical part of NOWA. We first discuss the specific requirements of the scientists –such as collaborative working, management of large files, versioning of data and code, quality assurance at data and code level, meta data management, publishing research as well as automating these steps as effectively as possible – and then compare them with established solutions from the computer science and software engineering domain, respectively (e.g., version control using git and GitLab, continuous integration/testing, style guides, and packaging). These so-called best practices are then adapted to the specific needs within the CRC. Finally, in the conclusion of the article we provide an overview of the planned technical and technological implementation of the NOWA assistance system.

2. What Neuroscientists need

A neuroscientific study in the CRC involves typically a sequence of consecutive steps from designing experiments to collecting data and analysing data (see Figure 1). Since each study usually consists of several sequential experiments that are building-up on each other, this cycle is traversed several times. During each of those steps, different types of research data are produced: Meta data include information about the design of the experiment, the settings of technical equipment and software, personal data about the tested participants or information about the type of data transformation and analysis. Software consists of scripts for experiments and for data analysis as well as for computational modelling. Primary data refer to the immediate outputs of the experiments, while processed data refers to analysed and aggregated data and statistical results.

2.1. Sharing and Publishing Research Data

One objective of the CRC is to facilitate the collaboration of researchers within and beyond the CRC. Sharing data with researchers in and outside of the CRC poses additional requirements to research data management, which are specified below.

[2] https://www.uni-marburg.de/de/forschung/kontakt/forschungsdatenmanagement/projekte/hefdi-hessische-forschungsdateninfrastrukturen

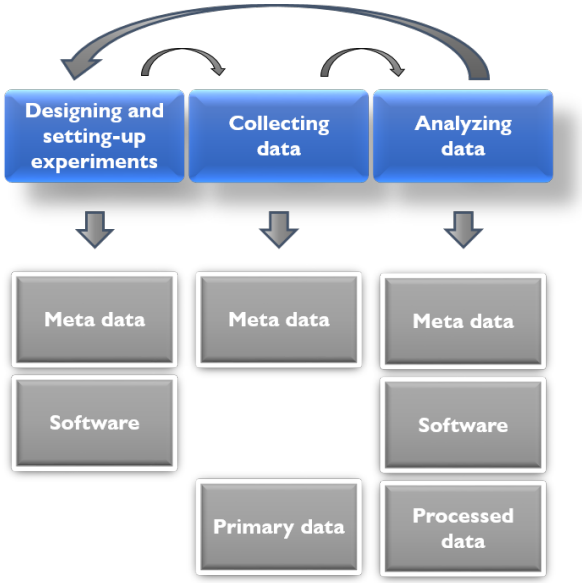

Figure 1.: The research process and where research data occur

Private and collaborative work Sharing data with other researchers requires comprehensive meta data regarding all aspects of a study, including the experimental design, the collection of primary data and the analysis of processed data, such that all involved researchers can comprehend the study material. The sharing of research data can be accomplished more effectively if those requirements are met from the beginning of the study. Working collaboratively on a study also requires a user management that allows to specify access privileges for different researchers and that keeps track of which researcher is working on the material. Of course, collaborative work also requires an easy exchange of research data and (semi-)automatic comparison of files and detection of modifications.

Publication Ultimately, all research data of the CRC should be made publicly available within the constraints of ethical regulations. To publish the data of a single study, this requires that all research data of the study are bundled and that each entity is tagged according to whether it can be made public or not. A direct export filter to existing research data repositories could facilitate the publication of data, such that only a single button-press is required.

2.2. Managing Research Data

An efficient management of research data is an important tool to guarantee the reproducibility of research. In the following, we highlight the most important requirements of research data management in the CRC.

Meta data, primary data, and code As mentioned before, different types of research data are produced during the life-cycle of a study. Several objectives need to be met by efficient research data management: For instance, different types of research data

need to conform with each other and redundancies in the data should be eliminated. In addition, the work flow should be optimized such that the additional work load imposed by research data management is minimized. The efficient reuse of materials, for instance analysis scripts, in other projects should be facilitated.

Data versioning A study in the CRC progresses usually through several stages, from an initial test phase, the presentation of first results at a conference to the publication of the final results in a scientific journal and the final data in a data repository. Usually, all research data evolve during this life cycle of a study. To make the research progress transparent and reproducible, it is important to keep track of all changes and to be able to return to any previous stage of the study and the associated materials. This goal cannot be achieved without a continuous versioning of all data that belong to the study. Ideally, versioning of all study-related material automatically generates a digital lab book that allows a complete overview of the work progress in the study.

Quality of data A further objective of NOWA is to maintain a high quality of research data to ensure the reproducibility of the research and results of the CRC. Several issues can arise during a study that will ultimately compromise the quality of research data: Meta data, for instance, might be incomplete because the necessary documentation is not carried out immediately. Primary data might be incomplete or might contain invalid data due to technical or organizational problems during data collection. Software might contain unknown limitations or bugs due to insufficient testing. An active research data management should include automated checks for data quality and highlight potential issues early on when they can be corrected easily.

2.3. Overview

To summarize (see Figure 2), researchers in the CRC need an efficient research data management spanning the whole data life-cycle from the design of the experiments over data collection and data analysis to the publication of data. The research data management should include all types of research data, such as meta data, primary and processed data and software. Primary goals of the research data management are to ensure a high quality of the data, to track different versions of data and to facilitate the collaboration between different researchers and working groups.

3. What Computer Science and Software Engineering provides

In this section, we discuss some of the challenges that have been identified in the domains of computer science and software engineering over the last 40 years, as well as solutions developed for them. The objective is to discover similarities to the needs of the researchers presented in the previous section in order to adapt the best practices from the domains of computer science and software engineering to the domain of neuroscience.

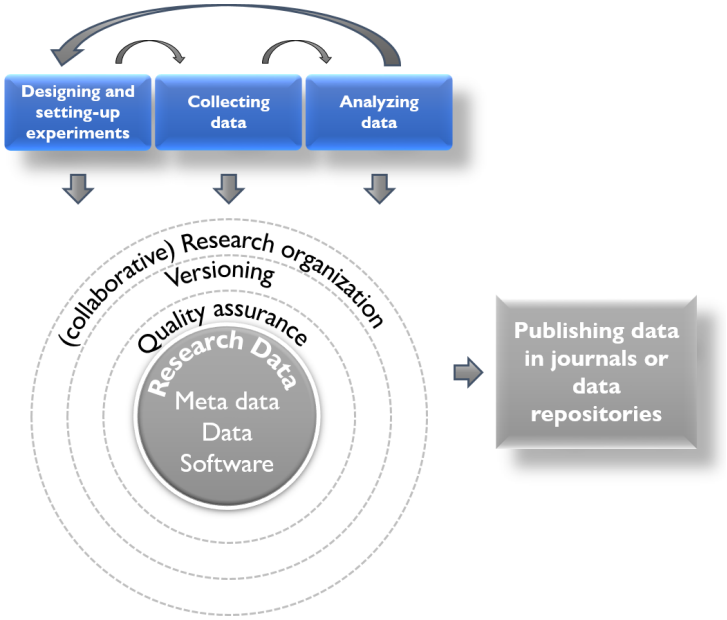

Figure 2.: Overview on what CRC/TRR 135 neuroscientists need

3.1. Collaboration and Versioning

Especially in recent decades, software systems have become ever larger and more complex. Consequently, the software engineers were confronted with a number of challenges in the area of so-called configuration management. For example, because large and complex software systems consist of a large number of collaborating components, the dependencies between these components must be managed. As these components evolve over time, each with its own extent and speed, the versions of the components and the entire system must be stored properly. Furthermore, old versions of files and the entire project should be easily and effectively recoverable, for example if the changes that are made lately prove to be insufficient. Additionally, for maintainability of the system, it would be very helpful to be able to easily understand how the components evolved between versions. Finally, it would be beneficial to have some sort of implicit backup system to counter the risks of a single point of failure when the code is managed on a single system. However, the core of these specific challenges is more general. Large and complex software systems are being developed by a large number of project members, many of whom are spread over different locations around the world. Therefore, the core of the challenges is to enable collaborative and distributed work within the software development project.

In order to meet the challenges in the field of configuration management, the computer science community developed so-called Version Control Systems (VCS) [14]. Such a system tracks changes to project (a set of files) and for each change, it records the date and time, the person who made the change, and the differences between the file contents before and after the change. As a result, the system can re-create a consistent snapshot of the project at any point in time. Two different approaches to version control systems have evolved over time. In the centralized approach, the VCS consists of a server that stores the only master copy of the entire project. Examples for centralized VCSs are Concurrent

Versions System (CVS)[3] and Subversion (SVN)[4]. However, since the centralized approach depends on a server, working on the project is only possible if there is a network connection. Furthermore, this approach carries the risk of a single point of failure. In contrast, the distributed approach generally does not have a server. Each user has a full copy of the whole project with the full history on the computer. Popular distributed VCSs are Mercurial[5] and git [6]. The distributed approach represents the state of the art.

In terms of supporting collaborative and distributed work within a software development project, a number of useful solutions have been established: groupware, computer-supported collaborative work, CASE tools, collaboration platforms, etc. [13, 9]. All of these solutions share the core features co-ordination, collaboration, and community building –with communication as cross-cutting function. Furthermore, these systems can be classified into their spatial and temporal dimensions. The spatial (or geographic) distribution in the software development process distinguishes spatially close co-located and distributed organizational units. In the temporal distribution, the focus is on whether the cooperation, in particular the communication, takes place with a time delay. Here, information and software artifacts can be either IT-buffered (asynchronous) or concurrently (synchronously) exchanged respectively edited. A combination of the two aforementioned concepts represent web-based collaboration platforms having a VCS as core technology. Here, state-of-the-art platforms are GitHub[7], GitLab[8] and Bitbucket[9].

3.2. Data, Meta Data and Software

Digital data has been stored in files of various formats for decades. However, this syntactic separation is often in contrast to the semantic contexts within the data. In order to summarize data semantically, applications such as ZIP and RAR were developed early on. They package a set of heterogeneous files into an archive and simultaneously compress their contents. This is similar with software systems. Even small applications consist of a large number of components that provide the required functionality only in interaction and depend on each other to a different extent. Appropriately developed software packagers [6, 8] carry out the combination of these components –in the corresponding versions– into an installable and executable application.

In most cases, data has a content-related structure. Uniformly structured and logically related data volumes are managed in databases. The essential task of a database is to store large volumes of data efficiently, without contradictions and permanently and to provide the necessary subsets in different, needs-based forms of presentation for users and application programs. At the next level, a data repository is an infrastructure of databases that collect, manage and store varying data sets. In fact, a data repository can be seen

[3] `https://savannah.nongnu.org/projects/cvs`
[4] `https://subversion.apache.org/`
[5] `https://www.mercurial-scm.org/`
[6] `https://git-scm.com/`
[7] `https://github.com/`
[8] `https://gitlab.com/`
[9] `https://bitbucket.org`

as a general term that comprises several concrete ways to collect and store data[10]. Such a repository can be implemented in one of the following ways. Data warehouses are large data repositories that aggregate data from multiple sources or segments of a business, without the data being necessarily related [7]. Data lakes are large data repositories that store unstructured data that is classified and tagged with meta data. An example can be found at [12]. Finally, data cubes are lists of data with three or more dimensions stored as a table [11]. The common structure of data can be specified using so-called meta data. Meta data are often described as information about data, i.e., the information required to understand data, including data set contents, context, quality, structure, and accessibility. The meta data explains where the data originated, how it was captured, and what it represents. In most instances, meta data are descriptions of things, whether physical objects (books, items in a warehouse) or information objects (spreadsheets, web pages, publications, etc.). Meta data repositories store data about data and databases. An example of a meta data repository for technical information in digital medical images can be found at [15].

Today, information and knowledge in the form of data and programs are widely published on the World Wide Web. Often, this information is in a packetized and archived format, as described above. However, even at this higher level, there are relationships between data (sets) stored at different locations in the internet. In order to make these relationships more explicit, this information can be coupled through the concepts of Linked Data [5] and the Resource Description Framework (RDF)[11], respectively.

3.3 Software Quality

Software of inferior quality is hardly accepted by the users. In extreme cases, this can mean that the software is rarely or not used at all. As a consequence, achieving a high quality is one of the major challenges in software development. But what does software quality exactly mean? To explicitly describe the term software quality, several so-called quality models for software products gradually evolved over the last 50 years. All of these quality models have in common that they describe certain quality aspects that own a software. These quality aspects are structured using a tree-based hierarchy to form the model. For example, the ISO 9126 standard[12] specifies six independent, high-level quality characteristics that are further sub-divided into 21 sub quality criteria. Following this standard, a software should meet the intended needs, perform well under stated conditions, and provide appropriate performance relative to given resources. Moreover, it should be understandable and easy to use, be modifiable with minimal effort, and transferable to another environment. Altogether, the need to develop high quality software resulted in establishing a software quality assurance process that ensures that the developed software meets and complies with defined or standard quality specifications.

The evaluation of software quality can be effectively automated using the state-of-the-art continuous testing & integration approach [18]. This approach works as follows. When

[10] `https://www.cbronline.com/opinion/what-are-data-repositories`
[11] `http://www.w3.org/TR/rdf-concepts/`
[12] `https://www.iso.org/standard/22749.html`

developer check in their changes to the code repository the VCS triggers a continuous integration (CI) process on a dedicated server. This server builds the software, runs several specified (static and dynamic) tests and analyses, returns the results to the developer, and potentially releases the new version of the software. In modern version control platforms like GitLab the CI server is integrated and and represents a central component for the assurance of the software's quality.

Already in the early years of software engineering, the developers noticed that they re-implemented many similar and even identical algorithms repeatedly. This led to the notion of software reuse representing the process of creating software systems from existing software rather than building software systems from scratch. Especially the upcoming paradigm of object-oriented software development forwarded and empowered the software reuse approach. For enabling software reuse, appropriate code is outsourced into so-called libraries, which can then be integrated into the actual software system to be implemented. Such software libraries can be provided either internally within the software company only or externally, e.g., as open source. A survey on software reuse libraries can be found in [19], for example.

Coding does not enforce problems as long as the code is syntactically correct. However, software code also has other quality characteristics than just syntactic correctness. Here, the use of standards and guidelines –e.g., for the Java programming language[13]– with a consistent style helps to develop high-quality code. A consistent style improves the readability, and therefore, maintainability of code. Furthermore, it facilitates sharing of code among different programmers, especially teams of programmers working on the same project. Finally, it saves development time, once the guidelines are learned, by allowing programmers to focus on the semantics of the code, rather than spend time trying to determine what particular format is appropriate for a given situation. Compliance with these guidelines can be verified by appropriate tools, e.g., during the CI process. In 1998, Kent Beck and Martin Fowler developed the concept of code smells [4] which has been further adapted to other software artifacts such as software models [1]. Code smells represent suspicious parts that are potential candidates for improvements, i.e., they are not synonyms for problems but are worthy of an inspection. In recent years, also approaches to use machine learning techniques to detect code smells have been developed (see [10] for an overview). Finally, the concept of quality assurance has been also adapted to assessing the quality of meta data. A comprehensive overview in this field can be found at [20].

3.4 Adaptation

In this section, we compare the needs of the scientists within the CRC with those of the computer science and software engineering domain. In doing this, we reuse solutions and best practices developed to the challenges of software engineering and adapt them to the domain of neuroscience. These adaptations will then provide a roadmap for the current and future work in building up the NOWA system.

As can be easily seen, the requirements of researchers and those of computer scientists are similar in many ways. In the remainder of the section, we discuss the common

[13] `https://google.github.io/styleguide/javaguide.html`

requirements in the three categories of data management, collaborative work, and quality assurance of artifacts. In the category data management, the correspondence is quite obvious. Both domains deal with data, meta data, and software code. The same holds for the category collaborative work. Members of the team want to share their artifacts with other people. Furthermore, researchers might ask the following questions similar to those in configuration management:

- Which version of our research data is part of a particular publication?
- How did our analysis script evolve during the data analysis, and why?
- Who made the last change in the current version of our research paper, and which one?

Finally, there is also a large overlap in the category quality assurance. The main goal of researchers is that their research is correct and trustworthy. This implies that research results are reproducible through the information provided about the methods used and the associated research data. As a consequence, the research data must be of high quality. For example, the data must match the corresponding meta data, or the previously mentioned analysis scripts must be implemented correctly. Figure 3 shows which best practices from the software engineering domain can be adapted and applied to which challenges in the CRC. Table 1 summarizes, in a before and after comparison, how the application of these best practices will impact the current research workflows in the CRC – again along the above categories and the common goal of simultaneously publishing research results and research data.

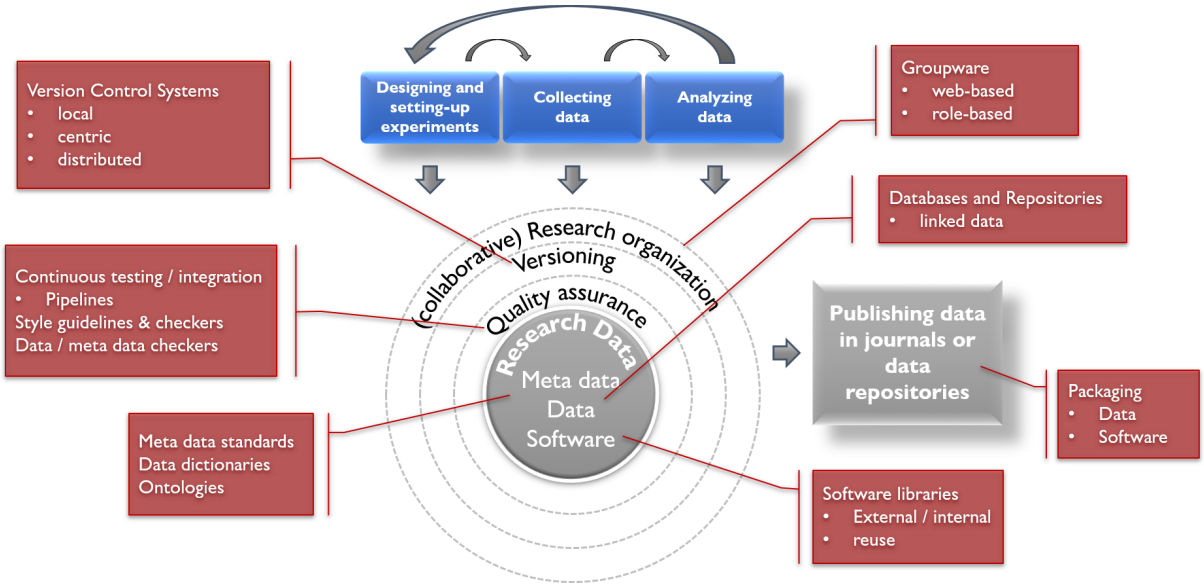

Figure 3.: Adaptation of best practices from the domains of computer science and software engineering to the needs of the researchers within the CRC

With regard to research data management, concrete data can be stored in databases in special repositories with a given structure. The data can then be linked with each other

if necessary. This reduces the current high number of less-structured files and introduces more unified, clearer and more basic file structures. Meta data can be centrally managed and stored with the help of standards, data dictionaries or ontologies[14]. As a result, the previously rather implicit or multi-managed meta data become explicit and through the standardization clearer and more comprehensible. With regard to the research software to be created, more software libraries can be used. These include external standard libraries, but also libraries built internally in the working groups or in the entire CRC. Thus, the more error-prone and time-consuming copy-paste programming can be reduced and the software can be created faster and more comprehensible.

With regard to collaboration, we can adapt a combination of the concepts groupware and version control systems. More specifically, we will use a local instance of the version control collaboration platform GitLab as the basis for the NOWA assistance system. By using GitLab, documents can be shared in a central yet distributed environment. The complexity that has resulted from the previous use of different methods for file exchange (e.g., via e-mail or file hosting services) is thereby greatly reduced. Furthermore, data and documents are managed more clearly, since the different file names used so far for the versioning are omitted.

With regard to the quality assurance of research artifacts, various methods and techniques can be adopted from software development, in particular, of course, for the creation of the necessary research software. The so far only fundamentally existing quality considerations such as syntactic correctness checks are supplemented by more sophisticated methods such as the use of style guides. However, the effort to check the artifacts for various new quality assurance techniques must be kept to a minimum –after all, the actual research is still the focus of the researchers. This can be done in particular by using the continuous integration technology, which is already integrated in the version control collaboration platform GitLab.

Finally, with regard to the joint publication of research results and data, the methods of packaging used in software engineering, both for data and for software, can be adapted. The previous complexity in the challenge of compiling the relevant research data can be significantly reduced by using the tagging feature provided by the VCS GitLab. Combined with the use of a structured quality assurance process started early in the research workflow, researchers within the CRC can ensure the overall reproducibility of their results.

[14] These best practices were not covered in this article.

	Before	After
Research Data	• High number of less-structured files • Implicitly given meta data • Copy-Paste programming	• Unified file structures • Explicitly given, standardized meta data • Software reuse
Collaboration	• Document exchange via Email or file hosting services • File versioning by different names with different nomenclatures	• VCS server with central and distributed repositories • Versioned files with permanent names
Quality Assurance	• Basic quality (syntactical checks only, redundancy, etc.) • Nearly no automation	• High quality (conformity, comprehensibility, reusability, etc.) • CI pipeline within VCS
Publication	• Complex (time-consuming, error-prone, etc.) • Low utility (data reuse difficult)	• Effortless due to using tagged data versions • High utility (data reuse easy)

Table 1.: Before/after comparison when adapting best practices from software engineering

4. Current and Future Work

As already mentioned in the previous section, we will set up a local instance of the GitLab web application at the UMR University Computer Center that is based on the distributed version control system git. This will represent the core of the NOWA workflow assistance system. By using the possibilities given by GitLab, a continuous and distinctive versioning of the active research data in the CRC projects will be supported in the best possible way. In particular, the Git LFS (Large File Storage)[15] extension optimally manages large files such as fMRI data. Additionally, using GitLab will facilitate improved collaborative work within the interdisciplinary working groups of the CRC. For this purpose, we will prepare and integrate specific security and authorization concepts. Parallel to the set-up of the assistance system, we analyze the research workflows within the CRC in close collaboration with the researchers involved. These workflows will be consecutively optimized by the possibilities provided by GitLab. Together with good practices for the general use of GitLab we will pass on the improved workflows to the researchers in regular training courses.

Since GitLab also provides a continuous integration and testing subsystem, our workflow assistance system will provide measures to ensure the quality of research data on top of this. On the one hand, we integrate existing quality assurance techniques for the software code that is currently used within the CRC research processes. On the other hand, we plan to develop further –possibly project-specific– quality assurance techniques together with the researchers which will then also be integrated into the assistance system. Moreover, NOWA will provide a collection of reusable libraries for research software such as data analysis scripts. The basis for this can be, for example, established libraries for the languages Python or MATLAB as provided in [17, 21] or [16, 22], respectively. These are then successively extended by own implementations. NOWA will initially make these libraries available to the researchers of the CRC and, ultimately, in a public repository of the GitLab instance.

In addition to the opportunities provided by GitLab, the workflows of the researchers will be supported by additional services offered in the infrastructure facilities of the participating universities. On the one hand, the collaborative work can be supplemented by the Sync&Share solution HessenBox, which is currently being tested. On the other hand, active research data management can be optimally supported by the use of HeRDMO, the HeFDI-administered prototypical installation of the research data management solution RDMO (Research Data Management Organizer)[16] also funded by the DFG. This tool captures all relevant planning information in data management plans and manages all data management tasks throughout the entire research data lifecycle. In passive research data management, the publication of research results and associated research data can then finally be achieved via the existing publication servers in conjunction with the data repository DSpace[17] currently under test at the UMR computer center.

The mere publication of research results and data of any kind within the World Wide

[15] `https://git-lfs.github.com/`
[16] `https://rdm1organiser.github.io/`
[17] `https://duraspace.org/dspace/`

Web together with their linking in a standardized format often does not guarantee the reproducibility of the underlying research. Current approaches and efforts to overcome these challenges include, for example, the concept of so-called research objects [3][18] as well as the publication of all relevant information in a reproducible article of the eLife Science Magazine[19]. In the future, we will also pursue and integrate these approaches in our NOWA assistance system.

Acknowledgements

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 222641018 – SFB/TRR 135 TP INF.

Bibliography

[1] Thorsten Arendt. *Quality Assurance of Software Models - A Structured Quality Assurance Process Supported by a Flexible Tool Environment in the Eclipse Modeling Project*. Doctoral thesis, Philipps-University Marburg, pp. 1–385 (2014). DOI: 10.17192/z2014.0357

[2] Thorsten Arendt, Ortrun Brand, Christian Krippes, Andreas Gabriel, Matteo Valsecchi, Clemens Helf, Karl R. Gegenfurtner, and Alexander C. Schütz. *NeurOscientific Workflow Assistance (NOWA)*, Poster presented at DINI Jahrestagung 2018, Bielefeld, Germany (2018). DOI: 10.17192/es2019.0002

[3] Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, Matthew Gamble, Danius Michaelides, Stuart Owen, David Newman, Shoaib Sufi, and Carole Goble. *Why linked data is not enough for scientists*, In: Future Generation Computer Systems, vol. 29, pp. 599 – 611 (2013). DOI: 10.1016/j.future.2011.08.004

[4] Kent Beck and Martin Fowler. *Bad Smells in Code*. In: Refactoring: Improving the Design of Existing Code, 2nd Edition, Addison-Wesley Professional (2018)

[5] Christian Bizer, Tom Heath, and Tim Berners-Lee. *Linked Data: The Story so Far*, In: Semantic Services, Interoperability and Web Applications: Emerging Concepts, ed. Amit Sheth, pp. 205–227 (2011). DOI:10.4018/978-1-60960-593-3.ch008

[6] Alexandre Decan, Tom Mens, and Philippe Grosjean. *An empirical comparison of dependency network evolution in seven software packaging ecosystems*, In: Empirical Software Engineering, vol. 24, pp. 381–416 (2019). DOI: 10.1007/s10664-017-9589-y

[7] Barry Devlin and Lynne Doran Cote. *Data Warehouse: From Architecture to Implementation*, Addison-Wesley Longman Publishing Co., Inc. (1996).

[18] `http://www.researchobject.org/`
[19] `https://elifesciences.org/`

[8] Shouki A. Ebad and Moataz Ahmed. *Software Packaging Approaches – A Comparison Framework*, In: Software Architecture, Springer Berlin Heidelberg, LNCS, vol. 6903, pp. 438–446 (2011)

[9] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. *Groupware: Some Issues and Experiences*. In: Communications of the ACM, vol. 34, pp. 39–58 (1991). DOI: 10.1145/99977.99987

[10] Francesca Arcelli Fontana, Mika V. Mäntylä, Marco Zanoni, and Alessandro Marino. *Comparing and experimenting machine learning techniques for code smell detection*. In: Empirical Software Engineering, vol. 21, pp. 1143–1191 (2016). DOI: 10.1007/s10664-015-9378-4

[11] Steven Geffner, Divakant Agrawal, and Amr El Abbadi. *The Dynamic Data Cube*, In: Advances in Database Technology – EDBT 2000, Springer Berlin Heidelberg, LNCS, vol. 1777, pp. 237–253 (2000)

[12] Rihan Hai, Sandra Geisler, and Christoph Quix. *Constance: An Intelligent Data Lake System*, In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD), ACM, pp. 2097–2100, (2016). DOI: 10.1145/2882903.2899389

[13] Tobias Hildenbrand, Franz Rothlauf, and Armin Heinzl. *Ansätze zur kollaborativen Softwareerstellung*. Arbeitspapier, Universitätsbibliothek Mannheim (2006)

[14] Konrad Hinsen, Konstantin Läufer, and George K. Thiruvathukal. *Essential Tools: Version Control Systems*, In: Computing in Science and Engineering, vol. 11, pp. 84–91(2009). DOI:10.1109/MCSE.2009.194

[15] Hans-Erik Källman, Erik Halsius, Magnus Olsson, and Mats Stenström. *DICOM Metadata repository for technical information in digital medical images*. In: Acta Oncologica, vol. 48, pp. 285–288, Taylor & Francis (2009). DOI: 10.1080/02841860802258786

[16] Mario Kleiner, David H. Brainard, and Denis Pelli. *What's new in Psychtoolbox-3*. In: Perception, 36 (2007). DOI: 10.1068/v070821.

[17] Florian Krause and Oliver Lindemann. *Expyriment: A Python library for cognitive and neuroscientific experiments*. In: Behavior Research Methods, vol. 46, pp. 416–428 (2014). DOI: 10.3758/s13428-013-0390-6

[18] Mathias Meyer. *Continuous Integration and Its Tools*. In: IEEE Software, vol. 31, pp. 14–16 (2014). DOI: 10.1109/MS.2014.58

[19] A. Mili, R. Mili, and R.T. Mittermeir. *A survey of software reuse libraries*. In: Annals of Software Engineering, vol. 5, pp. 349–414 (1998). DOI: 10.1023/A:1018964121953

[20] Jung-Ran Park and Yuji Tosaka. *Metadata Quality Control in Digital Repositories and Collections: Criteria, Semantics, and Mechanisms*. In: Cataloging & Classification Quarterly, Routledge, vol. 48, pp. 696–715 (2010). DOI: 10.1080/01639374.2010.508711

[21] Jonathan Peirce, Jeremy R. Gray, Sol Simpson, Michael MacAskill, Richard Höchenberger, Hiroyuki Sogo, Erik Kastman, and Jonas Kristoffer Lindeløv. *PsychoPy2: experiments in behavior made easy.* In: Behavior Research Methods, vol. 51, pp. 195–203 (2019). DOI: 10.3758/s13428-018-01193-y

[22] Sabine Verboven and Mia Hubert. *LIBRA: a MATLAB library for robust analysis*. In: Chemometrics and Intelligent Laboratory Systems, vol. 75, pp. 127–136 (2005). DOI:10.1016/j.chemolab.2004.06.003