EPiC Series in Computing

Volume 107, 2025, Pages 364–373

Proceedings of EUNIS 2025 annual congress in Belfast

“It works on my research”: An argument for prioritized software testing in research software

Lars Michel[1*], Tom Carnein[1] and Markus von der Heyde[2†]
[1] University of Potsdam, Germany
[2] SemaLogic UG, Germany
lars.michel@uni-potsdam.de, tom.carnein@uni-potsdam.de,
markus.von.der.heyde@semalogic.de

Abstract

As science deals with the ever-growing complexity of research fields, research software provides the tools to observe, assist and manage their processes and results. However, the majority of research software tends to prioritize the validation of theories and the creation of prototypes over the delivery of products, often overlooking the crucial aspect of quality assurance. This can result in products that are susceptible to flaws and errors. The risk of failing to deliver correct results not only undermines user trust, but also damages the reputation of researchers and developers in a team, risking losses in funding and public trust in research. The implementation and emphasis on software engineering practices, particularly software testing, can mitigate these risks by prioritizing the identification and elimination of errors and faults in software development while also enhancing observability and transparency throughout the coding cycle. In part, the project “CAVAS+” is used as an example of these practices. A thorough examination of the current state of research software development and its challenges highlights the importance of software quality and reliability in research software, ultimately contributing to a more robust research software ecosystem.

1 Background

Scientific inquiry, driven by hypothesis testing and experimentation, has greatly expanded our understanding of the world. However, many phenomena remain beyond direct observation due to limitations of scale, time and space. To overcome these limitations, scientists increasingly rely on simulations and computational models. At the core of this approach is research software which is powered by advanced computing technologies and often involves a multitude of libraries or microservices.

[*] ORCID - Lars Michel: https://orcid.org/0009-0007-7523-1507
[†] ORCID - Markus von der Heyde: https://orcid.org/0000-0002-6026-082X

The definition of the term “research software” does not share a universally agreed definition, though for this context, it is described as “source code files, algorithms, scripts, computational workflows and executables that were created during the research process or for a research purpose” (Gruenpeter et al., 2021, p. 16). As such, these software tools are developed to overcome challenges in the various domains of science. An example includes the measurements of attributes of celestial objects in astrophysics and simulations of their behavior over large periods of time: Where humans would not be able to see far or live long enough to observe the outcomes, programs developed on astronomical theories are able to quickly calculate and estimate the possible outcomes in their stead. Simulations of this nature represent a significant advantage of software, offering capabilities such as modeling, data processing, statistical analysis, visualization and experimental design. Notably, these benefits are achieved in a more efficient manner compared to traditional software-less methods.

However, as software complexity increases, it becomes increasingly challenging for “pure scientists” to develop and maintain these on their own. Recent studies indicate that most scientific teams that use software report a lack of adequate training in software development, primarily due to their educational backgrounds in scientific fields: “many scientific software developers obtain their knowledge from other scientific developers […]. This lack of formal training often leaves scientific software developers blind to much of the field of software engineering that could provide much greater control over the quality of their code” (Heaton and Carver, 2015, p. 208). The intricate nature of certain software-dependent domains contributes to a high likelihood of errors, which are often difficult to identify. On top, development limitations with the relation to science frequently result in additional challenges, including, but not limited to “limited interoperability, dependence on closed proprietary tools, management of large data sets, and computational challenges” (Bernoth et al., 2024, p. 283).

To create successful research software, it is indicated that the combination of in-depth scientific knowledge with engineering skills is essential: “scientific software developers believe that software engineering practices could increase their ability to develop quality software and software engineers agree that scientific software developers need adopting software engineering practices would allow them to produce higher quality software” (Heaton and Carver, 2015, p. 217) Scientists must be able to translate their research findings into data and algorithms while software engineers and developers must be able to turn these findings into practical software solutions. Alternatively, scientists could also acquire software development skills of their own to taking on a hybrid role that combines both the provision and implementation of scientific software. However, neither option is commonly represented within the scientific community. While research software repeatedly accelerates the discovery and innovation of the domain, it is also often considered a secondary tool that does not take a large priority. This can result in a lack of qualitative design and well-tested functions in the software, leading to unreliable results that compromise the integrity of scientific research and risk inaccuracy or falsehood in discoveries, resulting in criticism.

In order to mitigate the risks of faulty research software, information about the current state of research software and their shortcomings has been gathered to formulate recommendations to developers, researchers, and stakeholders involved in scientific computing. The aim is to increase the overall reliability and trustworthiness of these programs. Common issues caused by the lack of software engineering practices such as missing agile processes, insufficient testing or poor documentation, will be addressed to increase communication between scientists and software engineers and ensure sufficient training in computational methods for researchers. The increased priority of software testing improves the efficiency and correctness of implemented algorithms and facilitates the reuse of the software for further experiments in the same domain or variations of them in other domains. Transparency and communication between developers and the stakeholders (or the public) establishes more trust in the process through exchange of information and quick response to issues.

2 Recommendations to increase quality and reliability

Reliability is a critical factor in the effectiveness and longevity of software. However, research software often exhibits shortcomings in development and maintenance, resulting in suboptimal quality in the final product (or prototype). While standard IT practices are present in development environments focused on science and proofs, they often do not meet the higher standards set by software engineering principles. A survey of research software developers, conducted by Eisty and Carver in 2022, highlights that these deficits can be mitigated by introducing or emphasizing software engineering practices, particularly in training and software testing. The context of research software development also demands an increased focus on trust and communication, as potential failures can permanently tarnish the reputation of otherwise adequate work.

2.1 Increased training in software engineering practices

Where commercial software standardized development strategies such as defined requirements, version control and regular testing, scientific software prioritizes short-term goals like functionality and rapid prototyping over long-term value like maintainability and scalability. This makes sense because less experienced researchers typically have a shorter-term interest in the software or its results, but contribute a major part of the academic software. As commercial vendors and academia are pressured by constraints in time and resources additional quality checks may appear to detract from the final research output or product. In order to avoid inaccuracy or even false information in the output, it is recommended that at least some software engineering methods, which are not too unfamiliar with scientific habits, can be integrated without too many drawbacks.

This can largely be addressed by providing skills for proper software development in advance as “later modifications become increasingly difficult and error-prone” (Heaton and Carver, 2015, p. 209). Research shows that “researchers are rarely purposely trained to develop software” (Carver et al., 2022, p. 6) and that they are “either unaware of their need for or may not have access to sufficient formal training in software development” (Carver et al., 2022, p. 7). If researchers develop their software engineering skills early and well enough, they will be able to dedicate the same level of effort to the research software development as they do to their own research field. Specialized training in development competencies “should provide research software developers with hands-on experience so they are able to understand and utilize existing testing techniques and tools in their projects, where appropriate” (Eisty and Carver, 2022, p. 25).

Addressing the issue at the outset of the project cycle ensures that subsequent stages such as the initial design stages or the coding process itself, are supported by a robust foundation. This approach facilitates the creation of software that is more structured and easier to maintain, aligning with both scientific objectives and long-term usability requirements: “As research software developers better understand the existing tools and techniques, they will also be able to identify gaps that can be filled by modifying existing tools and techniques or by creating new ones” (Eisty and Carver, 2022, p. 25).

However, the same studies also mention that training in software engineering is available, but does not have a positive effect on development for different reasons. While it is possible to consider the training practice to be impractical or even irrelevant to the otherwise specialized software in research, a larger issue lies in time constraints: “while training may be available, respondents do not have adequate time to take advantage of it” (Carver et al., 2022, p. 19) – An issue that could be solved through adjustments in management, ensuring team members of a scientific project have the necessary time to learn and apply engineering skills to their project for quality assurance purposes.

2.2 Higher priority and quality of testing

Crucial to the developed software is the guarantee of high quality. Written functions must fulfill their intended purpose, transforming required inputs into expected outputs during execution, ideally in a short time. This is achieved through the dedicated stage (or parallel development, depending on the overall project development strategy) of software testing and conclusive debugging where potential errors in the source code are identified and eliminated to ensure a reliable code structure.

In a study about the claims of software engineering practices in science, it is asserted that “the effectiveness of the testing practices currently used by scientific software developers is limited” (Heaton and Carver, 2015, p. 212). While common responses do acknowledge that testing is done for the sake of “correctness of software”, other answers such as “easiest or least effort required for these tests” or even in favor of skipping testing by saying “it is usually clear whether the software is working as intended” are more unorthodox. These statements hint at a badly arranged, sometimes even the absence of a validation process, a necessary condition to certify correct results and deliver the answer whether the overall output represents the solution to a problem.

A peculiar perspective emerges when considering integration and regression testing in particular, known as the control procedure for how different software components interact with each other during source code updates: “many scientists who successfully test their code are actually using integration testing already, but they just think of it as using the scientific method. For example, every time a model is changed, the scientists treat it as a new experiment and test it, using the previous results as a control” (Heaton and Carver, 2015, p. 213). This perspective encourages testing by framing it as “updated experiments”, enabling a quick look into early results and follow-up adaptations depending on them. Similar parallels can be drawn when talking about the processes of verification and validation. While verification is typically defined as insurance that the software is built correctly and meets specified requirements, for scientists, “this same concept is described as ensuring that the computational model […] matches the mathematical model” (Heaton and Carver, 2015, p. 213). The same can be said for validation, the assurance that the software fulfills the intended purpose and its user requirements, where scientists are also “ensuring that the mathematical model […] matches the real world” (Heaton and Carver, 2015, p. 213). Taking these parallels into account, the implementation of proper test suites appears similar to scientific processes and motivates their application in the development process.

Concerns have been raised regarding studies and responses suggesting that testing is not only lacking, but also more challenging in the context of scientific software development compared to traditional software development as the correct results are often undefined or unclear. Poorly written test cases result in imprecise reports: “The problem is that when the oracle and test results do not match, the scientist does not know whether the problem lies with the theory, the theory’s implementation, the input, or if the oracle itself is flawed” (Heaton and Carver, 2015, p. 213). Aforementioned validation is often not given: “[The] lack of experimental validation means that a scientist may not even have an expected answer” (Heaton and Carver, 2015, p. 213). This highlights the need for clearer oversight during development, with consideration for future projects.

2.3 Practices in regards to trust and reputation

As effective quality management becomes both more challenging and more crucial, the consequences of failure are becoming more widespread. A 2024 US government technical report, endorsed by “leading technology companies, academics, and civil society organizations” (The White House, 2024), highlights these concerns. While the report primarily focuses on cybersecurity issues such as memory safety, rather than research software specifically, it highlights “the multifaceted nature of the software measurability problem” (The White House, 2024) which presents a challenge to software in general.

Despite decades of efforts from major institutions to eliminate entire classes of software defects, fundamental quality issues persist, requiring ongoing large-scale interventions. This raises a critical question: If even rigorously tested, commercially developed software struggles with reliability and security, how can research software, often built under far fewer constraints and oversight, be immune to similar problems? The challenges of assessing and improving software quality are not limited to cybersecurity, but extend to any domain where software plays a crucial role.

A recent example of a widely used commercial software failing to meet expectations is the event known as the 'CrowdStrike incident' in July 2024 which affected Windows operating systems worldwide, causing widespread disruption. The incident impacted a broad range of sectors by wrecking "havoc on airlines, health care systems, banks and scores of other businesses and services around the world" (Satariano et al., 2024). Public trust in the company immediately decreased, measurable by the stock price, which "plunged 13% on Monday […] after Wall Street analysts downgraded the stock on concerns over the financial fallout from a global cyber outage last week" (Reuters, 2024).

Beyond failures in commercial software, research software faces its own trust-related challenges. Two notable instances are the retractions of research related to COVID-19 which demonstrate how a lack of transparency in software can lead to a loss of confidence in scientific findings.

"The Lancet published, then retracted, analysis of the effects of hydroxychloroquine or chloroquine treatment for COVID-19. In the retraction, Prof. Mehra indicates that the data and software on which the analysis was based were not made available for independent peer review or replication" (Lee et al., 2021). Common software practices, such as making the code open source with basic documentation or a descriptive test suite, could have helped prevent this issue, possibly leading to less loss of trust in the scientific community.

Another significant example of this phenomenon is a "simulation by scientists at Imperial College London, one of several models that helped to sway UK politicians into declaring a lockdown" (Chawla, 2020). Even though the research model turned out to be correct, the massive criticism it received could have been mitigated through the adoption of enhanced coding practices.

While the direct impact of these retractions is difficult to quantify, they exemplify how flaws in software can undermine trust in both scientific research and policy decisions - an issue that remains relevant, as skepticism toward COVID-19 measures persists to this day.

3 Reliability practices in CAVAS+

In light of the ongoing debate surrounding research software, the authors introduce their current project, along with the practices they have implemented to ensure reliability. The project is titled "Computer Assistance for the Validation and Accreditation of Study Regulations to improve studyability (CAVAS+)".

It is evident from the title that CAVAS+ is an organizational development and training project dedicated to the creation of software prototypes that assist educational and structural purposes. More precisely, it is an approach to model study and examination regulations and their frameworks with a formal specification language, SemaLogic, and to convert them into a clear, machine-interpretable structure. The result is an advanced program that can continuously check for consistency and contradictions during the writing process and transparently visualize the interpreted representation for subsequent processes. This potentially reduces the effort involved in downstream checking and correction loops and enables the rules contained in the study and examination regulations to be applied directly in subsequent digital systems, thereby increasing process efficiency (Lucke, 2023).

It is important to note that, while the software developed within CAVAS+ is considered scientific software on grounds of being developed and discussed in university context and aiming to answer a research question, it is technically not research software itself as it does not generate concrete knowledge through data analysis or processing. Rather, CAVAS+ conducts research in order to develop such software.

Much like research software, CAVAS+ recognizes the importance of well-written software and strives to hold the quality of written code and functionality of components to a high standard. In its current state (as of February 19th 2025), the lines of test-related code in the projects amounts to an average of 25.34% (see Figure 1). A variety of software testing practices have been implemented, ranging from basic ones that are to be expected from programs fulfilling their intended function to more advanced ones that would go beyond the fundamental principles of a typical software development project. Software testing can be categorized in different ways, but a key approach is based on levels. It is possible to test both a single function in the project structure and all project components as a whole with different interpretations for their results. Common in CAVAS+ are three types of testing which address different levels of the microservice architecture. Unit tests, as the lowest layer, "searches for defects in, and verifies the functioning of software (e.g., modules, programs, objects, classes, etc.) that are separately testable." (Graham et al., 2007, p. 44). These tests evaluate written functions at the individual level, with minimal consideration for import, and are designed to ascertain the function's alignment with its intended purpose. Above it are integration tests which "tests interfaces between components, interactions to different parts of a system such as an operating system, file system and hardware or interfaces between systems" (Graham et al., 2007, p. 45). These tests ensure that different microservices communicate well with each other. An illustrative example is the request of "module data" describing details of a study regulation from the "External Data Service" component, followed up by transforming the data in the "Student Use Case Controller", then forwarding that to "CavasGUI" to display the result as a solution in the front end. It is possible to verify whether the expected results are displayed in the front end, as this indicates that all services are communicating correctly with each other. At the top level, system test and acceptance tests, would test the project as a whole and determine its readiness for realistic application. While some groundwork exists, development is not yet advanced enough to address this last level.

Less easy to fit into a layer, but still highly relevant, is a testing method that works in tandem with the code as it is written and updated. To ensure that project components still fulfill their core purposes

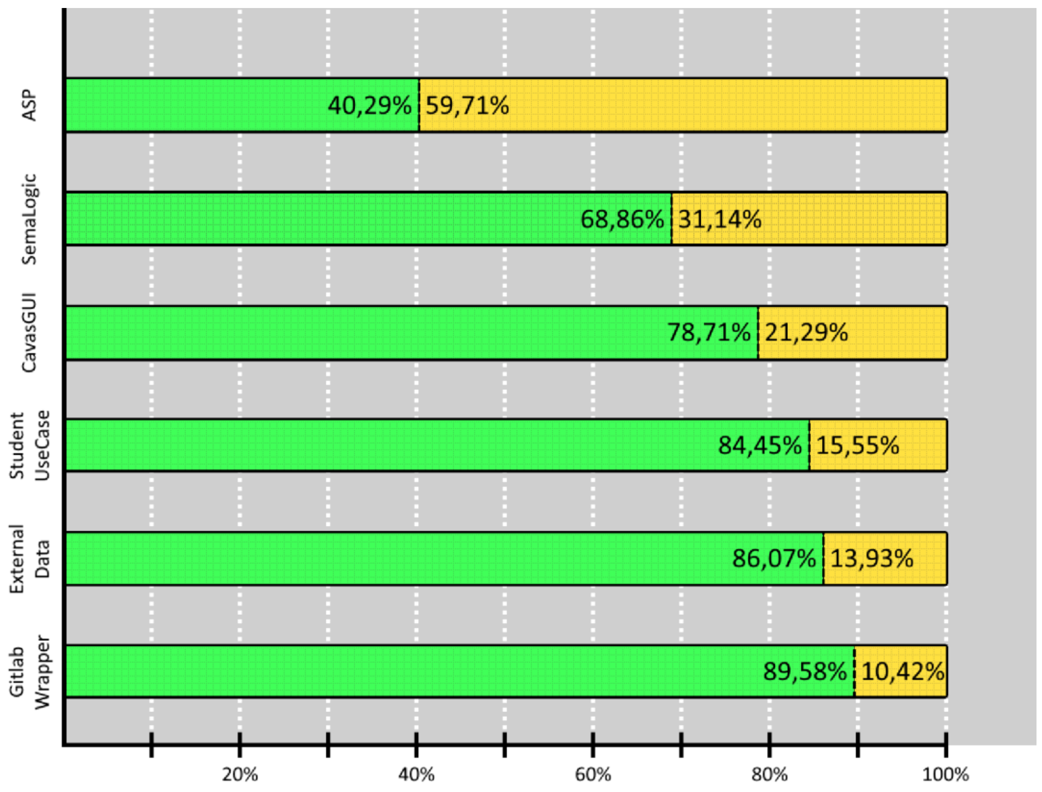


Figure 1: Percentages of lines of code written in subprojects services and APIs in CAVAS+. Green: Development related code. Yellow: Testing related code. The outlier in ASP API is (partly) due to containing several SemaLogic models in its test suites.

after each update of the code, regression testing has been added into the pipelines of each project (CAVAS+ uses GitLab to collaborate on and maintain code). Regression testing is defined as the process "to verify that modifications in the software or the environment have not caused unintended adverse side effects and that the system still meets its requirements" (Graham et al., 2007, p. 52).

Survey results such as those from Carver et al. (2022) demonstrate that the current testing level is comparable to that of testing conducted in research software. While these results can be considered sufficient for each of the sub-projects under the comprehensive field of CAVAS+, it should be noted that this method is prone to errors. Developers tend to be biased towards their own developments: "it is difficult to find our own mistakes. So, business analysts, marketing staff, architects and programmers often rely on others to help test their work. This other person might be a fellow analyst, designer or developer." (Graham et al., 2007, p. 30). In CAVAS+, that "other person" is represented by the dedicated sub-project "Quality Management". The purpose of this sub-project is to oversee other actively developed sub-projects and ensure that they write their tests as well as keep them up to date and to the required standards. Quality management is currently working as the project's independent integration test and will later generate and evaluate a system test of all components. As for its team members, they participate in the short meetings occurring twice a week to follow developments of the project components, communicate potential issues that may arise and discuss options to manage these issues using test suites of various levels. One such exemplary communication result is the accommodation of each user story (a planned software feature based on user requirements) with a relevant test suite. The quality management developers also contribute to CAVAS+ as well through added work for a “Test Dashboard” that is currently underway. This is a web page that provides a display of the current state of all relevant components of the project. It allows not only other project members to identify problems during development, but also future maintainers to pinpoint sources of issues in the running prototype.

While the current state of quality assurance in CAVAS+ appears satisfactory, it has not always been the case. As a research project, developing an AI-based support software for study regulations was not as straightforward as it is usually the case for dedicated product development at companies. In traditional software engineering, discussions regarding the direction of a potential software product take place in the early stages of development. During these phases, requirements are typically structured and well-defined, and they generally remain consistent throughout the remainder of the development cycle. Agile software development has transformed the industry by adopting a user-centric approach, delivering minimal viable products in short development cycles. Similarly, in scientific research, priorities can evolve when initial plans prove unsuccessful. CAVAS+ uses a similar approach: The project's development leads to discoveries that define the potential direction. Only after these discoveries do requirements become clearer and development in all areas become more consistent. Arguably, this too can be seen as an argument for the necessity of a dedicated team of quality management as they can quickly adapt priorities in testing in line with changes to the project.

4 Conclusion

In the current climate of scientific complexity, traditional measurement tools are reaching their limits. However, technological advances have enabled the development of new methods for calculating and simulating complex phenomena. These innovations, in particular research software, have enabled the generation of highly precise results in scientific research. However, this is not yet the norm, with researchers often treating software as a means of proving a theory, rather than developing it for longevity and extensibility. Modern software engineering practices, which emphasize the creation of long-lasting, flexible software from the ground up, are therefore encouraged. Scientists require special training in software engineering methods such as appropriate design principles or acceptable documentation. In order to ensure the accuracy of research results, it is essential that written functions operate without flaws. This means that software testing and debugging must be given high priority during the development process. Increased communication and application of common software practices in the created software ensure that the work and effort of properly researched results cannot be harmed by overlooked faults. By strengthening engineering practices and addressing previously unconsidered gaps in our development process, it can be ensured that research software meets the same standards as commercially developed software in terms of quality, reliability and longevity. This will also enable newer continuation projects to build on our software as a solid foundation, further enhancing its value and impact.

References

Bernoth, J., Riedel, C., Wiepke, A., & Laban, F. A. (2024). *From Receiving to Characterizing: Improving Training Strategies for Research Data/Software Management by another domain*. https://doi.org/10.18420/DELFI2024_25

Carver, J. C., Weber, N., Ram, K., Gesing, S., & Katz, D. S. (2022). *A survey of the state of the practice for research software in the United States*. PeerJ Computer Science, 8, e963. https://doi.org/10.7717/peerj-cs.963

Eisty, N. U., & Carver, J. C. (2022). *Testing research software: A survey*. Empirical Software Engineering, 27(6), 138. https://doi.org/10.1007/s10664-022-10184-9

Graham, D., van Veenendaal, E., Evans, I., & Black, R. (2006). *Foundations of Software Testing: ISTQB Certification*. Int. Thomson Business Press. Retrieved from https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf

Gruenpeter, M., Katz, D. S., Lamprecht, A.-L., Honeyman, T., Garijo, D., Struck, A., Niehues, A., Martinez, P. A., Castro, L. J., Rabemanantsoa, T., Chue Hong, N. P., Martinez-Ortiz, C., Sesink, L., Liffers, M., Fouilloux, A. C., Erdmann, C., Peroni, S., Martinez Lavanchy, P., Todorov, I., & Sinha, M. (2021). *Defining Research Software: A controversial discussion (Version 1)*. Zenodo. https://doi.org/10.5281/ZENODO.5504016

Heaton, D., & Carver, J. C. (2015). *Claims about the use of software engineering practices in science: A systematic literature review*. Information and Software Technology, 67, 207–219. https://doi.org/10.1016/j.infsof.2015.07.011

Lee, G., Bacon, S., Bush, I., Fortunato, L., Gavaghan, D., Lestang, T., Morton, C., Robinson, M., Rocca-Serra, P., Sansone, S.-A., & Webb, H. (2021). *Barely sufficient practices in scientific computing*. Patterns, 2(2), 100206. https://doi.org/10.1016/j.patter.2021.100206

Reuters. (2024). *CrowdStrike shares tumble 13% on IT outage impact*. Retrieved from https://www.reuters.com/technology/crowdstrike-shares-set-extend-losses-outage-effects-linger-2024-07-22/

Satariano, A., Taylor, D. B., Tumin, R., & Kaye, D. (2024). *Outage for Microsoft Users Knocks Out Systems for Airlines and Hospitals in Chaotic Day*. The New York Times. Retrieved from https://www.nytimes.com/live/2024/07/19/business/global-tech-outage

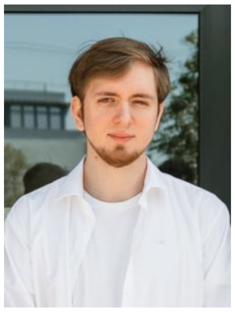
The White House. (2024). *BACK TO THE BUILDING BLOCKS: A PATH TOWARD SECURE AND MEASURABLE SOFTWARE*. Retrieved from https://web.archive.org/web/20250118014817/https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf

The White House. (2024). *Statements of Support for Software Measurability and Memory Safety*. Retrieved from https://web.archive.org/web/20250116115017/https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/memory-safety-statements-of-support/

von der Heyde, M., Goebel, M., Lindow, S., & Lucke, U. (2024). *Einsatz symbolischer KI in Hochschulen durch formale Modellierung von Studien- und Prüfungsordnungen* [Use of symbolic AI in universities through formal modeling of study and examination regulations]. Informatik Spektrum, 47(3), 87–96. https://doi.org/10.1007/s00287-024-01577-9

Author biographies

Lars Michel studied both the Bachelor program in Informatics/ Computational Science as well as the Master program in Computational Science at the University of Potsdam. He started working as a scientific assistant in March 2022 at the Chair of Complex Multimedia Application Architectures in the Institute of Computer Science at the same university, parallel to his master studies. After finishing and achieving his degree, he was promoted right away to work full time as a scientist and developer since January 2024. He is currently involved with the CAVAS+ project with a focus on quality management and software testing. Besides that, he also shows interest in general multimedia, software management and virtual reality. In his free time, he is editing media for content as well as participating in and contributing to the fighting game community.



Tom Carnein is a student enrolled in the Bachelor of Informatics/Computational Science program at the University of Potsdam since October 2020. In March 2022, he started working as a scientific assistant on the CAVAS+ project, where he has contributed to frontend and backend development, focusing on database management, data analysis, transformation, and API integration. Beyond his academic and professional work, he enjoys developing software in his free time, including Android applications, video games, and various other types of software, which has given him broader insights into software design, architecture, and development in general. Currently, he is spending time abroad in Japan, further expanding his perspectives on technology, cross-cultural collaboration, and international research environments.

Dr. von der Heyde received his PhD with topics in cognition research at the Max Planck Institute for Biological Cybernetics in Tübingen. Since 2011, Dr. von der Heyde has been advising colleges, universities, and public cultural and research institutions on a wide range of digitalization topics (governance, organization, strategy, research data management, information security, IT service management) as part of vdH-IT, and conducts independent research on these topics (see ResearchGate). Since 2018, he has been an Adjunct Professor at the School for Interactive Arts and Technology (SIAT) at Simon Fraser University, Vancouver. Dr. von der Heyde is also active as a volunteer in a variety of non-profit organizations (GI, ZKI, EUNIS, Educause). In 2020, he founded SemaLogic UG to use semantic and structural logic technologies to automatically map and validate natural language regulatory texts. The application of these technologies to study regulations and accreditation is currently being implemented with partners from the university environment. Further details can be found on LinkedIn or Google Scholar.