Combining Repository & Process Mining for Better Scientific Software

Thorsten Sommer*

November 19, 2021

1. Challenge

The typical development of scientific software begins as a prototype for own needs, or those of the own organizational unit. For such prototypical development, developers often forgo recognized techniques from professional software development, such as requirements analysis, user stories, documentation of use cases, development of an appropriate architecture, and consultation with potential users. This leaves issues such as IT security, quality, usability, maintainability and scalability out of consideration. Complicating matters further, scientific software is often developed by scientists from all disciplines without the necessary training and support (cf. Haupt et al., 2018). At the same time, scientists working as software developers cannot be expected to become computer scientists in parallel. After all, the challenges in computer science are so complex that computer scientists spend a lifetime working and researching on them every day. In consequence of these conditions, so-called "technical debt" arises, see also Besker et al., 2019; the following four symptoms (S1 to S4) might result from the described situation.

S1: IT Security

During early development, IT security is deliberately ignored. The initial aim is to demonstrate feasibility or to solve a problem under high time pressure. Years later, the program continues to be used without the security aspects ever having been checked and implemented (cf. Siavvas et al., 2020). Ultimately, this leads to the overall IT security of the research organization being weakened.

S2: Oracle Problem

In early development, quality was laid out for the one immediate use case that needed to be solved. Under the conditions of this first use case, the software has always worked and produced correct results. After years of use, the software finds more users and gradually becomes a product. The original developer is no longer part of the organization, therefore the knowledge of how the software works has been lost; the software is now a black box for most users. The quality and correctness for the new conditions is and was not checked, so possibly unnoticed wrong results are produced (cf. Vogel et al., 2019). This issue is called the "oracle problem" and "large variability", see Vogel et al., 2019.

S3: Productivity

Where quality was not given sustained attention in scenario S2, the focus here is on usability. The software was developed for the initial task. Exactly for this purpose the software is usable ideally. Over time, more scientists, also from other disciplines, work with it. This gradually changes the use case. This can lead to lower productivity compared to the original use case (cf. Macaulay et al., 2009).

S4: Blackbox

For most scientists, universities are transit stations on their careers. In countries such as Germany, the maximum duration of stay is even regulated by law, for example by the "Wissenschaftszeitvertragsgesetz" of 1999. Under these conditions, the prototypical development described at the beginning is a problem that becomes apparent in the medium and long term. During the initial development, no or inadequate documentation is produced. In the best case there are publications about the software. Years later, after the original developers are no longer part of the organization, the knowledge about the implementation is lost. The scientific HPC community has already recognized this issue for themselves and is trying to counteract it (Anzt and Flegar, 2019). However, this issue also occurs in a similar form when scientists spend a lifetime developing software and then retire. Further development becomes expensive or even impossible in both cases.

2. Opportunity

Nowadays, a common method to make these challenges manageable is the introduction of software development guidelines (cf. Haupt et al., 2018). Three of the four symptoms described here are thereby mitigated, but not eliminated. In particular S2 cannot be mitigated in this way, since the initial developer could not know which future use cases would occur. This is similarly true for

*thorsten.sommer@dlr.de; ORCID 0000-0002-3264-9934; German Aerospace Center (DLR), Institute for Software Technology, Intelligent and Distributed Systems Group

the other three symptoms. For example, no developer can know what IT security attacks unknown today will occur in the future.

In support of the software development guidelines, I therefore propose the development of a generic framework that uses repository and process mining to interactively support developers. Repository mining uses data science methods to, for example, derive information about the software development process (cf. Kurnatowski et al., 2020). Hence, similarity to other projects (El Baff et al., 2021) and IT security vulnerabilities (Schreiber et al., 2021) can be determined automatically.

Process mining can be used to derive information about processes and thus also about their use. In the software domain, there is the approach of working with telemetry data. The software under investigation sends user events to a server. By evaluating these user events, insight can be gained into the use of software and the emerging issues, for example in the area of usability.

Combining these approaches results in a solution that scales to any number of scientific software and can be applied to both new and legacy software. Symptom S1 can thus be completely eliminated. In the case of S2, telemetry can at least detect when new use cases make inputs that potentially lead to incorrect results so that scientists can be warned. Likewise, the loss of productivity in the case of S3 can be detected so that new requirements can be assessed and the software can be adjusted in terms of its usability. In the sense of the black box in S4, repository mining can be used to recognize retrospectively how the software was created step by step in order to gain knowledge about its implementation. Additionally, an unknown software architecture can be visualized in such a way that the viewer can explore the architecture to improve its understanding (cf. Schreiber and Misiak, 2018).

In order to achieve this goal, the framework to be developed would have to be able to handle the most common programming languages, such as C, C++, C#, Python and Java. The repository mining part of the framework must be able to analyze these languages in order to identify e.g. IT security vulnerabilities and programming flaws. For the process mining aspect, the framework needs an embedded web server. For maximum interoperability, language-neutral interfaces would then be provided using the OpenAPI standard, allowing integration into new and existing scientific software with little effort. If such a generic framework would be maintained as an open source solution by the scientific community, it could establish itself as a standard for scientific software development in the future.

3. Timeliness

Approaches such as repository and process mining are relatively recent. In the meantime, enough experience exists so that the framework considered here can be implemented. In addition, thanks to open source, the worldwide software engineering ecosystem has now reached a state in which not only the own code but almost all of its external dependencies can be analyzed by means of repository mining, so that a holistic overall picture can obtained with the approach presented here.

4. References

Anzt, H., Flegar, G., 2019. Are we Doing the Right Thing? — A Critical Analysis of the Academic HPC Community, in: 2019 IEEE Intl. Parallel and Distributed Processing Symposium Workshops (IPDPSW). Presented at the IEEE Intl. Parallel and Distributed Processing Symposium Workshops, IEEE, Rio de Janeiro, Brazil, pp. 739–745. DOI: 10.1109/IPDPSW.2019.00122

Besker, T., Martini, A., Bosch, J., 2019. Software developer productivity loss due to technical debt—A replication and extension study examining developers' development work. Journal of Systems and Software 156, 41–61. DOI: 10.1016/j.jss.2019.06.004

El Baff, R., Santhanam, S., Hecking, T., 2021. Quantifying Synergy between Software Projects using README Files Only, in: Proceedings of the 33rd Intl. Conference on Software Engineering and Knowledge Engineering. Knowledge Systems Institute Graduate School, Pittsburgh, USA. DOI: 10.18293/SEKE2021-162

Haupt, C., Schlauch, T., Meinel, M., 2018. The Software Engineering Initiative of DLR - Overcome the obstacles and develop sustainable software. Presented at the ACM/IEEE Intl. Workshop on Software Engineering for Science, ACM, Gothenburg, Sweden, pp. 16–19.

Kurnatowski, L. von, Stoffers, M., Weigel, M., Meinel, M., Wasser, Y., Rack, K., Fiedler, H., 2020. Scientific Software Engineering: Mining Repositories to gain insights into BACARDI, in: 2020 IEEE Aerospace Conference, AERO 2020. Presented at the IEEE Aerospace Conference, IEEE, Big Sky, MT, USA, pp. 1–10.

Macaulay, C., Sloan, D., Jiang, X., Forbes, P., Loynton, S., Swedlow, J.R., Gregor, P., 2009. Usability and User-Centered Design in Scientific Software Development. IEEE Software 26, 96–102. DOI: 10.1109/MS.2009.27

Schreiber, A., Misiak, M., 2018. Visualizing Software Architectures in Virtual Reality with an Island Metaphor, in: Chen, J.Y.C., Fragomeni, G. (Eds.), Lecture Notes in Computer Science. Presented at the VAMR 2018: Virtual, Augmented and Mixed Reality: Interaction, Navigation, Visualization, Embodiment, and Simulation, Springer Intl. Publishing, Las Vegas, USA, pp. 168–182.

Schreiber, A., Sonnekalb, T., Heinze, T., Kurnatowski, L., Gonzalez-Barahona, J.M., Packer, H., 2021. Provenance-Based Security Audits and Its Application to COVID-19 Contact Tracing Apps. Lecture Notes in Computer Science 12839, 88–105.

Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., Tzovaras, D., 2020. Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises. Enterprise Information Systems 1–43. DOI: 10.1080/17517575.2020.1824017

Vogel, T., Druskat, S., Scheidgen, M., Draxl, C., Grunske, L., 2019. Challenges for Verifying and Validating Scientific Software in Computational Materials Science. 14th Intl. Workshop on Software Engineering for Science, IEEE, Montreal, Canada, pp. 25–32. DOI: 10.1109/SE4Science.2019.00010