Scientific Developers v/s Static Analysis Tools: Vision and Position Paper

Rohan Krishnamurthy*, Thomas S. Heinze*, Carina Haupt†, Andreas Schreiber†, and Michael Meinel†
*Institute of Data Science
German Aerospace Center (DLR)
Jena, Germany
rohan.krishnamurthy@dlr.de, thomas.heinze@dlr.de
†Simulation and Software Technology
German Aerospace Center (DLR)
Berlin and Cologne, Germany
carina.haupt@dlr.de, andreas.schreiber@dlr.de, michael.meinel@dlr.de

***Abstract*—Usability and the use of automated static analysis tools in the software development process have been an evolving subject of research in the last decades. Several studies shed light on issues like high false positive rates and low comprehensibility, which hinder tool adoption for even software engineers. Yet, the tools' perceived usefulness and ease of use play a much larger role when it comes to untrained software developers, as is usually the case in scientific software development. In this paper, we outline a multi-stage interview study to learn more about how scientists come to accept and use static analysis tools.**

I. Introduction

Developing software is a vital part of scientific research today. While research results and their reproducibility are depending on the quality of research software, the usage of software engineering practices is still not widely spread among research facilities and individual researchers [1], [2]. This also applies to the use of static analysis tools, which could though provide an effective measure to the development of quality software, e.g., by identifying defects and code smells or enforcing common coding standards. In particular scripting languages, which are frequently used for scientific software, should benefit from static analysis tools [3], since safety guarantees as provided by statically typed programming languages are otherwise missing due to their dynamic nature.

Unfortunately, known issues like high false positives rates, low comprehensibility of analysis results, and missing process integration restrain the use and acceptance of static analysis tools also among professional software developers. The identification of factors which help in increasing the tools' acceptance is therefore an even more important premise for their widespread adoption in the scientific software domain. In this paper, we outline our multi-stage interview study to learn more about how scientists come to accept and use static analysis tools when developing software.

DOI 10.1109/CHASE.2019.00029

II. Technology Acceptance and Use

There exist different models for explaining user adoption and use of IT in the research literature, where Davis' Technology Acceptance Model (TAM) [4] is among the most prominent ones [5]. In TAM, an user's intention to use a certain IT system, and thus its actual usage, is predicted by basically two variables: perceived usefulness and perceived ease of use. The former one denotes the degree to which an individual believes that the use of the system will help in accomplishing a task. The latter one denotes an individual's belief in how less effort is required for using the system. The model also states that the influence of external variables, e.g., the system's UI design, are mediated by these two variables.

Several studies have shown empirical support for TAM [5]. Later extensions identified additional variable determinants (TAM2, TAM3 [5]) or integrated them with other models into the Unified Theory of Acceptance and Use of Technology (UTAUT) [6]. In UTAUT, social influence is, besides perceived usefulness and ease of use, the other main construct determining an user's intention to use a system, while facilitating conditions, e.g., training, also predict actual system usage.

III. Studies on Static Analysis Use

Use of static analysis has been researched by several authors and also has drawn wider attention [7] lately. Due to space constraints, we here just sketch a few representative studies. Also note, that we are not aware of work that addresses the use of static analysis in scientific software development.

Johnson et al. conducted interviews with 20 experienced software developers [8]. The authors were interested in the shortcomings of static analysis tools. Besides the high number of false positives and poor understandability of analysis results, both relating to perceived usefulness and ease of use, they found the weak integration with developers' workflows as barrier for wider adoption. The same line of thought is

followed in [9], where 375 Microsoft developers answered questions on their perspective on static analysis tools. The interviews showed again the importance of precision and comprehensibility. Process integration, configuration, and a supporting team policy, i.e., facilitating conditions, have also been identified as crucial for the adoption of static analysis tools. In addition, the beneficial effect of an instantaneous analysis has been revealed, which is further emphasized with respect to just-in-time analysis and IDE integration in [10]. Social influence and facilitating conditions, in terms of organizational factors and communication channels used to spread static analysis tools, are also discussed in [11] and [12].

Beller et al. combined developer surveys with repository mining and found the common but not ubiquitous use of static analysis tools in popular open source projects [3]. The authors also stressed the importance of process integration and found that the static analysis tools are more widespread among scripting languages compared to statically typed languages. While, on the one hand, a scripting language seemingly requires more static analysis, on the other hand, its dynamic nature makes static analysis hard and the high false positives rate a bigger issue. As another observation, they found that static analysis tools are rarely customized, showing either the good fit of the tools' default configurations or rather indicating missing expertise, or at least awareness, among analysis users.

IV. Research Agenda

As a research organization, the German Aerospace Center (DLR) has a rich source of scientific software developers, who come from different domains and backgrounds and thus provide a heterogeneous population in regard to their knowledge and expertise in software engineering [2]. Our main objective is to better understand, what factors influence the usage of static analysis tools for this particular group and what are resulting opportunities for intervention to increase static analysis adoption. We will develop our research in a three-stage interview study to inspect these questions:

Stage 1: At DLR, we have the opportunity to conduct a pilot study, limited to a maximum of 50 participants coming from scientific software development. We are interested in the first place in the developers' software engineering background, used programming languages, and notions of code quality. Though, prior experience in using static analysis tools, which can range from not present at all to the use of simple linters or more advanced analysis, and sources for developers' initial exposure to the tools, will also be asked for. The collected data will be used to prepare the other stages of our study.

Stage 2: We will then prepare a workshop on static analysis tools and small development tasks which are related to their usage, e.g., refactor a code snippet while keeping the number of introduced defects low. The tasks will be tailored specifically to the domain and programming languages of the scientific developers, where we cover statically typed as well as dynamic languages. In the workshop, developers will be asked to perform the tasks and report in a qualitative interview about their experiences in doing so. We are in particular interested in the tools' perceived usefulness and ease of use:

- How do scientific developers perceive false positives, bad comprehensibility, batch-style analysis?
- Which kinds of tools are perceived most effective?
- Do developers apply evasive strategies, e.g., straight on ignoring tool warnings, introducing bypasses?
- Are there differences in the experiences for statically typed and dynamic programming languages?

The experiment will comprise a control group, which is asked to do the same tasks without static analysis tools, such that we can compare the effectiveness in task solving for both groups.

Stage 3: In order to learn even more about the adoption of static analysis tools by scientific software developers, we will ask the workshop attendees to participate in a long-term survey. Voluntaries will be asked to deploy a live survey component into their development platform, either implemented as a plugin in a popular IDE, such as Visual Studio, Eclipse, PyCharm, or as a hook into a CI pipeline, if available. The live survey will be triggered when a developer applies the supported static analysis tools within the development platform and will interrogate about comprehensibility of analysis results, etc. We plan to provide repetitive assistance in terms of newsletters and reminders in addition. This way, besides gaining more detailed data on usefulness and ease of use of static analysis tools, as perceived by the participants, we hope to also address questions about the effect of other variables, i.e., facilitating conditions and social influence:

- How good do the static analyis tool integrate with workflows in scientific software development?
- What is the effect of training and social environment?
- What is the team/developer's notion of code quality?
- Are there configurations/policies used for static analysis?

Acknowledgment

We would like to thank the reviewers for their constructive and helpful suggestions on improving the paper.

References

[1] A. N. Johanson and W. Hasselbring, "Software Engineering for Computational Science: Past, Present, Future," *Computing in Science and Engineering*, vol. 20, no. 2, pp. 90–109, 2018.

[2] C. Haupt, T. Schlauch, and M. Meinel, "The Software Engineering Initiative of DLR: Overcome the obstacles and develop sustainable software," in *SE4Science@ICSE*. ACM, 2018, pp. 16–19.

[3] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software," in *SANER*. IEEE, 2016, pp. 470–481.

[4] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[5] V. Venkatesh and H. Bala, "Technology Acceptance Model 3 and a Research Agenda on Interventions," *Decision Sciences*, vol. 39, no. 2, pp. 273–315, 2008.

[6] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly*, vol. 27, no. 3, pp. 425–478, 2003.

[7] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from Building Static Analysis Tools at Google," *Commun. ACM*, vol. 61, no. 4, pp. 58–66, 2018.

[8] B. Johnson, Y. Song, E. R. Murphy-Hill, and R. W. Bowdidge, "Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" in *ICSE*. IEEE, 2013, pp. 672–681.

[9] M. Christakis and C. Bird, "What Developers Want and Need from Program Analysis: An Empirical Study," in *ASE*. ACM, 2016, pp. 332–343.

[10] L. N. Q. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, and E. R. Murphy-Hill, "Just-in-time static analysis," in *ISSTA*. ACM, 2017, pp. 307–317.

[11] E. R. Murphy-Hill and G. C. Murphy, "Peer Interaction Effectively, yet Infrequently, Enables Programmers to Discover New Tools," in *CSCW*. ACM, 2011, pp. 405–414.

[12] S. Xiao, J. Witschey, and E. R. Murphy-Hill, "Social Influences on Secure Development Tool Adoption: Why Security Tools Spread," in *CSCW*. ACM, 2014, pp. 1095–1106.