Discussing the Future Role of Documentation in the Context of Modern Software Engineering (ISoLA 2022 Track Introduction)

Klaus Havelund[1⋆], Tim Tegeler[2], Steven Smyth[2], and Bernhard Steffen[2]

[1] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA
klaus.havelund@jpl.nasa.gov
[2] TU Dortmund University, Dortmund, Germany
firstname.lastname@cs.tu-dortmund.de

Abstract. The article provides an introduction to the track: *Programming - What is Next?: The Role of Documentation*, organized by Klaus Havelund and Bernhard Steffen as part of ISoLA 2022: the 11th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Software has to run on machines, but it also has to be understood by humans. The latter requires some form of documentation of the software, which explains what it does if the human is a user of the software, and how it does it if the user is a programmer who wants to modify the software. Documentation is usually the neglected artifact. This track attempts to focus attention on documentation as a first-class citizen of the software development process.

Keywords: Documentation · Domain-specific Languages · Modeling · Programming

1 Motivation and Background

In 1973, Hoare made the following statement in his famous *hints on programming language design*:

> *"The purpose of program documentation is to explain to a human reader the way in which a program works, so that it can be successfully adapted after it goes into service, either to meet the changing requirements of its users, to improve it in the light of increased knowledge, or just to remove latent errors and oversights. The view that documentation is something that is added to a program after it has been commissioned seems to be wrong in principle and counterproductive in practice. Instead, documentation must be regarded as an integral part of the process of design and coding."* [10]

⋆ The research performed by this author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

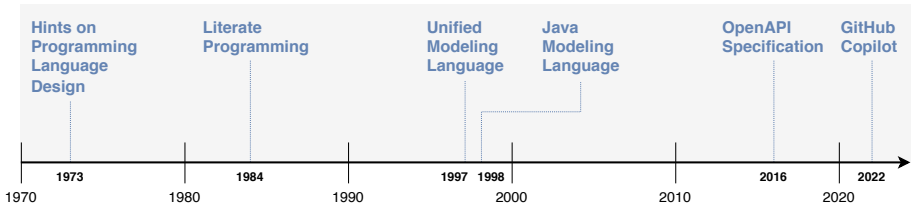

Fig. 1. Selected milestones in the historical evolution of software documentation

Despite the maturity of this statement, it is our observation that software documentation is often still considered a burden and merely realized as plain source code comments or concluded in a post-development fashion [4] as standalone documents. We believe this results from a *love–hate* relationship that we as software engineers have with software documentation. On the one hand we cherish well documented software and on the other hand we don't enjoy creating and maintaining the documentation of our own projects.

Almost 50 years after Hoare's statement, software documentation has gone beyond helping programmers to orient themselves during the programming and has many facets today. Not only gained it importance with the increasing collaboration of highly distributed teams, but it became a structured means that is more than human-readable text:

- Classic approaches such as Javadoc [11] are legitimate successors of Knuth's literate programming [14]. They were instrumental in enhancing comments by the adoption of structured tags that carry processable metadata of commented source code and the automatic generation of documentation in a browsable output format (e.g. HTML).
- JML [15, 12] goes beyond Javadoc. In particular, it supports design by contract by allowing pre and post-conditions to be written as annotations.
- Standardized modeling with UML introduced graphical notations to support documentation and understanding of object-oriented software [17, 7].
- Projects like the OpenAPI initiative [18] support a documentation-first approach to describe (RESTful) APIs in a human and machine readable format that allows automatic service discovery.
- Web-based documentations, e.g. [1], have evolved to software products themselves, making use of version control software and distinct build processes. This trend lead to the need for frameworks and libraries to generalise the creation of documentation, e.g. [2].
- GitHub Copilot [3] generates complete source code from comments, with the result of blurring the lines between documenting and programming.

We think this evolution (cf. Figure 1) has the potential to overcome our prejudices against documenting and finally position documentation in the center of software development. The aim of the track is to review current pragmatics and discuss the future role of documentation in the context of modern software engineering.

2 Contributions

In this section we present the contributions of the PWN track in the order of their presentation at ISoLA in three sessions which, incrementally, consider enhanced roles of documentation from mere additions to the code, via code/documentation conglomerates to various forms of executable documentation.

2.1 Session 1: Duality between Documentation and Code

This session addresses the duality between documentation and code. A seamless interplay of both entities is vital for the success of software projects; in particular for projects relying on the continuous collaboration of various stakeholders. The following papers lay the foundation for this track by discussing how abstraction in form of models can be used to represent software properties and reducing the need for additional documentation.

Coherent Description of Software System Properties. Broy [5] introduces his paper by discussing the term software documentation from different points of view and underscores that the development of software systems involves a wide-ranging set of challenging tasks, including the understanding of technical entities, maintaining the source code and operating the overall system. He argues that documentation is not only useful for each of these tasks but badly needed for software system development and evolution, especially in the context of DevOps-driven projects. Therefore continuously improving documentation, while verifying its consistency, is a key issue. One practical way of enriching documentation for this purpose is the use of models to abstractly describe important properties (e.g. functionality and architecture) of the considered software systems.

Models as Documents, Documents as Models. Stevens [20] reflects on similarities and differences between both terms. Documentation and models are broad concepts and overlap in software engineering. Models can serve to document certain parts of a software project and, in the context of model-driven engineering documents can be viewed as models as well. The paper approaches the questions “What counts as a model?” and “What counts as a document?”.

Using Supplementary Properties to Reduce the Need for Documentation. Programming languages serve two purposes in general. Firstly to instruct computers by being executed and secondly to describe the intended computational processes to humans. However, practice has shown that source code can be difficult to understand, even for experts. Thus, source code requires to be documented by supplementary documentation using natural language, diagrams, specifications and models. Madsen and Møller-Pedersen [16] dedicate this paper to the challenge of extending the expressive power of programming languages

in order to reduce the need for such supplementary documentation on the one hand, and to reduce the need for additional (documentational) languages on the other hand.

2.2 Session 2: Synergies between Documentation and Code

We often think of documentation as text written in natural language, explaining how code works or how it is used. However, documentation can also be understood more formally. This session focuses on such more formal forms of documentation, including visualization of code, assurance arguments, and models. Visualization of code can be viewed as visual documentation. An assurance argument is a semi-formal argument about the correctness of a system, and can be viewed as a structured form of documentation. Similarly, a design model is a form of documentation. The papers in this session discuss the tight integration of these forms of documentation and code, either by concretely linking artifacts with tooling, or by providing precise semantics of the documentation that supports correct code development and generation from a model.

Pragmatics Twelve Years Later. In 2010, Fuhrmann et al. introduced *modeling pragmatics* as a term for tools that combine the best of the textual and graphical worlds. A key enabler for this combination in practice was the ability to automatically synthesize customized (graphical) views from (possibly textual) models. Now, von Hanxleden et al. [8] reflect on their proposal twelve years later with the example of the recently developed coordination language Lingua Franca and discuss the obstacles, opportunities, and the outlook for modeling pragmatics in general.

Assurance Provenance. Modern society relies heavily on high-assurance systems software for transportation, medicine, finance, defense, and other areas. However, the rigorous requirements and processes for such safety-critical systems do not mesh well with modern agile software development practices, which focuses on 'good enough' software 'fast enough'. Karsai and Balasubramanian [13] argue that current CI/CD pipelines should be extended by Continues Assurance (CA) high-assurance software systems enabling rapid re-analysis and re-evaluation. For this, integrated tooling is essential: Developers need assistance for managing and maintaining complex systems.

Formalization of the AADL Run-Time Services. Hatcliff et al. [9] note that documentation is not only needed for a specific program or model but on the meta-level for the definition of modeling languages, e.g., for the Architecture and Analysis Definition Language (AADL). AADL is a modeling language focussing on strong semantics for real-time embedded systems. The definition of modeling languages require expressive documentation in order to allow all stakeholders to reason about the modeled system. However, in case of the AADL standard, the

documentation of run-time services is semi-formal. This allows divergent interpretations of the definition which renders the implementation of analysis or code generation functionality difficult. The authors show how rule-based formalization of key aspects of the AADL semantic can be documented to enable functions for realistic, interoperable, and assurable implementations.

2.3 Session 3: Executable Documentation

Executable Documentation is an umbrella term which includes all parts of a documentation that can be executed directly, create executable components (e.g., higher-level models or even running products), or components that manually or automatically generate parts of the documentation.

Test-First in Action. Smyth et al. [19] present two test-first scenarios. At first, a test-driven development approach tailored to the needs of programming beginners is illustrated. A focus is set on a test-guided, auto-didactic exploration through appropriate automatic diagram syntheses for runtime visualization for an immediate, tangible feedback. The test-based guidance can be automated, e.g., via randomized testing, or used to foster student-tutor interaction. In the second scenario, a similar test-first approach can be leveraged to improve modern web development. Instead of relying on static templates, concrete test instances are used to instantaneously visualize the final product in a WYSIWYG style. This radically reduces the time for tedious development cycles.

Runtime Verification as Documentation. In contrast to a runtime verification monitor that returns a Boolean verdict, Dams et al. [6] present three monitor examples that document system behaviour. For this, they combine runtime verification with techniques from data science. The three examples cover state reconstruction from traces, a data analysis of operations on a distributed database, and timed debugging. While the considered notion of dynamic documentation is rooted in runtime verification, such systems produce rich data, which require analysis that goes beyond Boolean verdicts. The authors argue that monitors written in a high-level programming language allow for arbitrarily complex data processing, which is often needed in industrial contexts.

From Documentation Languages to Purpose-Specific Languages. Domain-specific notation and documentation languages are usually used in the requirements and design phases. Tegeler et al. [21] illustrate how to turn these languages into fully-fledged purpose-specific modeling languages. These languages are designed to tightening the semantic gap between the *what* should be done and the *how* is it implemented, while also reducing the need for handwritten *why* specifications. The approach is illustrated within the DevOps scenario. Here, typical graphical CI/CD workflows are turned into an integrated modeling environment, called Rig. At a larger scale, Rig specifications, that are themselves

executable documentations, can be regarded as means to automatically provide corresponding running systems which can be considered the 'ultimate' (executable) documentations where the understanding of all stakeholders should converge.

The PWN track closes with a panel discussion regarding the role of documentation in the context of modern software engineering.

References

1. Laravel - the PHP framework for web artisans. https://laravel.com/docs, [Online; last accessed 31-August-2022]
2. VuePress. https://vuepress.vuejs.org, [Online; last accessed 31-August-2022]
3. Your AI pair programmer. https://github.com/features/copilot, [Online; last accessed 31-August-2022]
4. Aghajani, E.: Software documentation: automation and challenges. Ph.D. thesis, Università della Svizzera italiana (2020)
5. Broy, M.: Software system documentation: Coherent description of software system properties. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
6. Dams, D., Havelund, K., Kauffman, S.: Runtime verification as documentation. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
7. Fernández-Sáez, A.M., Caivano, D., Genero, M., Chaudron, M.R.: On the use of UML documentation in software maintenance: Results from a survey in industry. In: 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 292–301 (2015). https://doi.org/10.1109/MODELS.2015.7338260
8. von Hanxleden, R., Lee, E.A., Fuhrmann, H., Schulz-Rosengarten, A., Domrös, S., Lohstroh, M., Bateni, S., Menard, C.: Pragmatics twelve years later: A report on Lingua Franca. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
9. Hatcliff, J., Hugues, J., Stewart, D., Wrage, L.: Formalization of the AADL runtime services. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
10. Hoare, C.A.: Hints on programming language design. Tech. rep., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE (1973)
11. Javadoc. https://docs.oracle.com/en/java/javase/13/javadoc/javadoc.html, [Online; last accessed 02-September-2022]
12. JML. http://www.eecs.ucf.edu/∼leavens/JML, [Online; last accessed 02-September-2022]
13. Karsai, G., Balasubramanian, D.: Assurance provenance: The next challenge in software documentation. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
14. Knuth, D.E.: Literate programming. The computer journal **27**(2), 97–111 (1984)
15. Leavens, G.T., Baker, A.L., Ruby, C.: Preliminary design of JML: a behavioral interface specification language for Java. SIGSOFT Softw. Eng. Notes **31**(3), 1–38 (2006). https://doi.org/http://doi.acm.org/10.1145/1127878.1127884

16. Madsen, O.L., Møller-Pedersen, B.: Using supplementary properties to reduce the need for documentation. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
17. Object Management Group (OMG): Documents associated with Object Constraint Language (OCL), Version 2.4. https://www.omg.org/spec/UML/2.5.1/ (dec 2017), [Online; last accessed 08-February-2019]
18. OpenAPI Initiative: OpenAPI specification v3.1.0. https://spec.openapis.org/oas/latest.html (Febraury 2021), https://spec.openapis.org/oas/latest.html, Accessed 2022-03-25
19. Smyth, S., Petzold, J., Schürmann, J., Karbus, F., Margaria, T., von Hanxleden, R., Steffen, B.: Executable documentation: The real power of test-first. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
20. Stevens, P.: Models as documents, documents as models. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]
21. Tegeler, T., Boßelmann, S., Schürmann, J., Smyth, S., Teumert, S., Steffen, B.: Executable documentation: From documentation languages to purpose-specific languages. vol. Proc. ISoLA 2022. Springer International Publishing (2022), [in this volume]