Towards public understanding of software through modeling

Robbert Jongeling
robbert.jongeling@mdu.se
Mälardalen University
Västerås, Sweden

ABSTRACT

We consider the public understanding of software and its engineering to be imperative for citizens in the rapidly digitizing world. In this vision paper, we explore the possibilities of using software and systems engineering models for communicating quality aspects of software-intensive systems to their users. We provide a scope, vision, and research challenges in this direction, aiming to facilitate transparency about design choices and relevant quality attributes in areas such as safety, security, and privacy. Ultimately, we envision that these efforts can contribute to an improved public understanding of software systems.

CCS CONCEPTS

• **Human-centered computing** → **Collaborative content creation**; • **Software and its engineering** → **Maintaining software**; **System modeling languages**.

KEYWORDS

Transparency, Explainability, Public understanding of software and software engineering, Software and systems modeling

ACM Reference Format:
Robbert Jongeling. 2024. Towards public understanding of software through modeling. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3652620.3688560

1 INTRODUCTION

When we buy a bar of chocolate, we may find different kinds of information on it to help us make an informed decision about whether or not we want to buy and eat it. For example, we may see the list of ingredients, nutrition information, a health label, the best-before date, and perhaps even quality labels indicating, e.g., the source of the cacao. Moreover, we may find information on where to turn in case something is wrong, such as the contact information of the manufacturer. Users of software or software-intensive systems[1], on the other hand, typically have none or only very limited information to help them make an informed decision about whether or not they want to use the particular software. For example, users may not be aware of how the software ensures their privacy, how the software ensures security, how the software is tested, when the software was last maintained (if at all), who is responsible for the creation and maintenance of the software, or who were involved in gathering the requirements for the software.

We believe understanding software is imperative to prevent unintended outcomes due to e.g. limited consideration for minority groups due to their inability to affect the requirements for developed software, misplaced trust in software, or misinterpretation of the output of software systems. Moreover, understanding software becomes vital to allow people to participate in an increasingly digitized society,. To prevent these unintended outcomes, we assert it is vital for software systems to be transparent about their properties, and for users to be able to properly interpret this information.

We know that users are interested in software quality aspects, as they have reported so, e.g., in reviews of mobile applications [6, 9]. We argue for an improved transparency of software to empower users to critically assess it, which also requires some understanding of software engineering activities. In this paper, we provide a vision on how we can utilize models to support improving the public understanding of particular software systems, and the public understanding of software engineering in general.

Finkelstein already raised the issue of public understanding of software engineering almost thirty years ago, and argued for ten general aspects of software engineering that need to be understood before more technical aspects of particular software can be understood [7]. Not much more is detailed in the literature about how this shall be achieved, although both the ACM and IEEE make mention of public understanding in their ethical codes. The ACM code of ethics and professional conduct encourages computing professionals to contribute to improving the public understanding of computing (explicitly in Section 2.7 and indirectly, in 3.6) [4]. Members of the IEEE commit among other things "to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems" [10].

Public understanding becomes increasingly more important, for two main reasons. Firstly, software is ubiquitous and its complexity is increasing, due to increased functionality and due to increased interactions with other systems. Secondly, we put increasingly high expectations on users to engage in the creation, maintenance, or customization of software. For example with the further development of low-code and no-code environments, *Citizen Developers* are expected to engage in the development of software. Initially, Citizen Developers were considered domain experts without specific programming expertise but with high affinity with technology [1]. Pushing this idea even further, some visions of a future Software Engineering 2.0 envision an even broader range of citizens involved in development and collaborating with AI agents such as LLMs

[1]In the remainder of this paper we refer to either of these simply as "software"

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688560

to develop software [13]. While these agents may support activities in requirements, programming, testing, and others, there are fundamental aspects that make software engineering an engineering discipline that require knowledge and understanding beyond technical skills to be able to judge the consequences of engineering choices.

In this vision paper, we propose an approach to reuse software and systems engineering models, such as software design models (e.g. UML class diagrams and sequence diagrams), and system models (e.g. in SysML), by transforming them into models to be used for communicating aspects of software to users. By indicating their information needs, the users can participate in the creation of these models, specifically aiming to enhance their ability to gather information they need to make informed decisions about using the software or not.

The remainder of this paper is organized as follows. Section 2 provides more background on transparency, public understanding, collaborative modeling and related work on modelling for the general public. Section 3 details the proposal forming this vision paper. Section 4 discusses relevant aspects, generalization, and limitations of these ideas. Section 5 concludes.

2 BACKGROUND AND RELATED WORK

Software transparency does not stand on its own but is contributed to by activities that all help to bring it about: accessibility, usability, informativeness, understandability and auditability [11]. The notion of transparency can refer to digital processes and public information provision, as well as to software. Transparency is also desired by users, surveys have shown this fact in particular in the context of AI systems [5]. These findings are in majority from the requirements engineering community. Although it is mentioned that modeling with non-experts such as users requires an increased transparency [18], it remains an interesting aspect to bridge between the modeling and requirements engineering communities.

It has been argued that decision making about software is made more difficult by the scale and complexity of modern software, that often even exceeds experts' ability to deal with [3]. Moreover, the authors (of [3]) argue that this complexity should not be hidden to the general public but that instead the general public shall be able to make decisions related to software. In our view, that would first require public understanding of software engineering and insight into aspects of specific software systems.

Collaborative modeling activities have been proposed in which a broad set of stakeholders are involved in creating engineering models [8, 16]. These approaches are typically focused on engineering activities and do not involve users. Similar approaches, but with even broader sets of roles among involved people and developed artefacts are within *crowdsourcing*, which includes first steps towards involving the general public in software engineering activities. For example, there have been works involving users in eliciting requirements, scaping UI design, testing the software and reporting bugs, and even to a limited extent in architecture and implementation tasks [14]. For the more technical tasks such as implementation, the crowd from which is sourced consists of people already familiar with programming. In our work, we are not directly aiming for general public participation in these processes, although it could be one of the future outcomes.

Discussions at a recent Dagstuhl seminar resulted in some action points on research on collaboration in MDE, including the question why models are not more used for communication across boundaries of specific fields [12]. In our vision, the question of this range can be extended to reach even to the general public. Indeed, models bring an opportunity to provide some insight into important aspects of the software under development. They are used in our field for many engineering tasks and in fact one very common use of models and diagrams is for the communication between stakeholders [17]. Therefore, we aim to explore if models can also be used to communicate quality aspects of the software to users in which they are interested. For example in app reviews, users have mentioned the following quality aspects: usability, reliability, portability, compatibility, performance efficiency, security, and functional suitability [9].

A relevant question is then what type of information needs to be conveyed to users. A recent study has surveyed users to find out their needs for explanations about software they encounter in their everyday use [6]. Understandably, most concerns are about the interactions with the software and its behaviour, that is, understanding how to perform certain tasks or expressing frustrations with encountered behaviour of the software. Given the focus of the survey on the most recently used software by the respondents, these are in majority consumer-oriented software such as communication apps. When considering other types of software, such as safety-critical systems, the concerns may be different. Nevertheless, the survey [6] also contains some concerns by users on privacy and security concerns. Although we are interested in explaining software to users, we are not focusing solely on explainability of AI algorithms, which is concerned with understanding why opaque algorithms arrive at certain suggestions.

3 PUBLIC UNDERSTANDING OF SOFTWARE

In this section, we outline the scope and requirements for our proposal, and the research challenges that need to be bridged to achieve the vision.

3.1 Scope of this proposal

Finkelstein argued that understanding specifics of software requires an understanding of the underlying challenges and limitations of software engineering activities [7]. While models could be used for either of these activities, in this work, we limit our scope to the understanding of software. Within that scope, still we can distinguish several activities, for example understanding how a particular software operates, having insight into quality aspects of particular software, or understanding interactions between various software systems. In this paper, we focus in particular on how models can be used to communicate quality aspects to users. Figure 1 shows an overview of the scope of this work, as a small part within bigger shells.

We are in particular focussing on cyber-physical software-intensive systems. The focus on these systems is motivated by their impact through e.g. safety and security concerns that require these systems to attain various certifications such as compliance with the

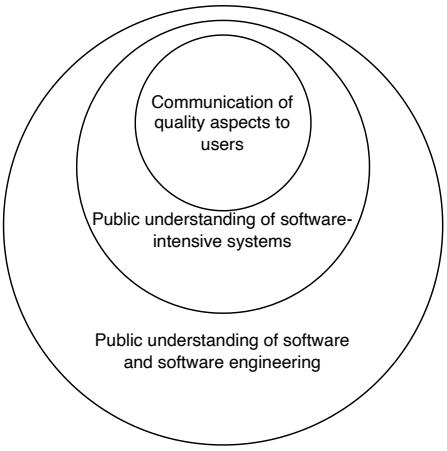

Figure 1: The scope of the research road-map proposed in this work is within the smallest circle

ISO-26262 standard. This standard is quite general and prescribes activities to be contained in the development process, but does not define how these shall be implemented.

Failures of this type of systems may have a large impact that general public shall be protected against, which is why they commonly need to be certified to achieve e.g. a certain safety integrity level (SIL). However, in our view, certification alone is not sufficient to prevent unintended effects from software, especially since we are expecting increasingly much autonomy from citizens in their interactions with software. Therefore, we need to promote transparency and improve understanding of these systems by their users.

When we discuss quality aspects of software, we consider for example questions about the system from the user's perspective. Extrapolating from the previously discussed results of [6], we imagine questions such as and not limited to the following:

- How do I know my data is secure?
- How do I know the software is properly tested?
- Who is responsible for maintaining this software?
- How do I know what the software updates do?
- Where can I report any failures of the software?
- Who were involved in establishing the requirements for this software?
- How do I know the requirements treat me equally?
- How do I know that the software will act in the correct way in my particular situation?

These questions are thus broader than the transparency of AI algorithms and extend to other opaque information about software. The proposed questions are an initial illustration of what could be relevant for users to know, which is an area of ongoing and future research.

3.2 What is required for such "user communication" models

In this section, we consider what the requirements of using models for communicating aspects of software to users should be. We refer to such models as communication models.

Personalized and dynamic. From considering science communication literature, we know it is relevant to consider pre-existing "mental models" of persons [2]. That is, people place newly gained knowledge within the context of what they already know and this has consequences for how to best present new information to them. In our view, we imagine that considering a mental-models approach may create the need for personalized explanations. Fortunately, we are in the software domain and so we have the possibility to customize explanations based on input from the user, when that includes the user's prior knowledge and current information needs.

Figure 2 illustrates our considerations for the inputs of communication models: the user's knowledge, the user's information needs, and the state of the system. Communication models cannot be static entities but rather need to be updated frequently, given that the input elements (in circles in Figure 2) are subject to change. Indeed, the software itself is typically subject to evolution. Moreover, with the user learning from previous explanations, both their prior knowledge and their information needs change over time. Therefore, it is relevant to understand how to personalize and update these communication models.

Input and output forms for users. Before anything else, users have an information need about the software that they want answered. At these input and output ends of the approach, we thus need to consider the question how users can input their "questions" and how the answer should be formatted. After the input, we know what those models shall contain, but we do not yet know in what form the models are best understandable for users. We should not expect the general public to learn UML or other modeling languages, so it is a future research effort to determine a suitable format for these communication models. These do not need to be diagrammatic and may instead consist of natural language text.

One option is to reuse other models used for communicating to the general public. Consider for example a standardized "label" such as we see on the chocolate bar, or such as energy labels that are commonly seen on packages of household appliances. These energy labels are standardized by the European Union[2] and regulate exactly what information must be captured on the label, and what values of the measured energy usage warrants what label value. We can imagine a similar scale for various aspects of software systems, where we could e.g. consider an A-G scale of cybersecurity of a particular product. While such scales can be defined, the reality is that we need to consider rather many quality items, with each their own label.

There is a trade-off between these standardized labels and the provision of personalized explanations. Standardized labels may increase broad knowledge among the public and be recognized by many people. The strength of these labels is that they provide an intuitive insight into relevant aspects of the product. On the other hand, the labels provide necessarily limited quantitative information within a narrow range of possibilities and cannot provide additional qualitative explanations to the users. Moreover, there is no room for personalized explanations as we have included in our vision. Further research is required to understand what type of information presentation about software systems is best for users.

[2]https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02019R2015-20230930

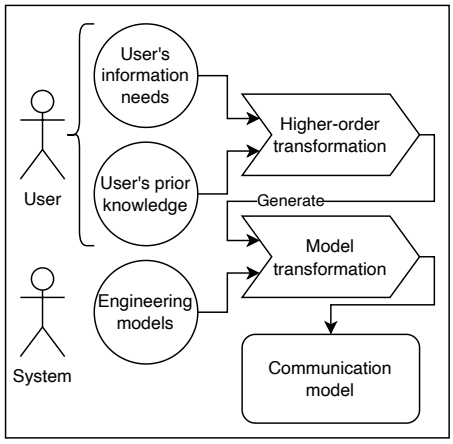

Figure 2: Conceptual overview of our vision. A user's information needs and prior knowledge are captured in models that are input to a higher-order transformation, from which a model transformation is then generated which takes system design models and outputs a communication model.

3.3 Towards automated generation of communication models

Figure 2 summarizes the vision proposed in this paper. The user's information needs and prior knowledge could be captured in models. These models are then used as input to a higher-order transformation that generates a model transformation to convert engineering models into communication models suitable for this particular user. The technical implementation of such transformations can be challenging, especially given the variety in modeling languages used for the engineering models that would need to be transformed. An obvious idea is to use large-language models (LLMs) to play the role of the transformations in this case, and gather the user's information needs and prior knowledge through natural language input. Furthermore, input and output from users could then be in the form of natural language texts. Some work in this direction has been proposed [15], in which the authors propose using an LLM as the enabler of proving explanations of the functionality of low-code tools. Other options are also viable and we need further research to determine which would work best for this type of communication.

For communicating aspects of the software, we propose to use views on existing software and system engineering models. To do so, we need user input on their information needs, and then transformations to create views that remove sensitive or IP-related information and just show the system aspects with respect to the information need of the user. The goal is thus to not generate completely new models, but rather to re-use existing models that are already used for the development of a particular software. Given their original purpose as engineering models, these models may contain (i) more information than is needed for communication and that should not be made public, but also (ii) not enough information about the quality aspects that need to be communicated. Therefore, we see a need for engineers to annotate the models with additional information and to highlight what can and what cannot be part of the communication models. Generating the communication model from the system design models then requires a further transformation from the model into a format that can be understood by users.

3.4 Summary of research challenges

From the description in this section, we now summarize the research challenges towards realising the vision of more transparent and understandable software.

(1) How can we collect user input on both their information needs and their prior knowledge? And how to keep track of the changing prior knowledge so that we do not explain to the user things that they have learned by then?
(2) How can we allow engineers to annotate engineering models such that they may be re-used for outward communication without revealing intellectual property? What shall be annotated? Is it possible to annotate engineering models with the required information?
(3) How can we create automated mechanisms to generate the communication models from existing engineering models?
(4) How can we format output models? Should we customize the form of communication models for various users to maximize their understanding? Or would it instead be better to create a singular format that can provide consistently information to all users?
(5) How can we measure success of the communication models? How can we ensure that the generated communication models sufficiently improve the understanding of the user?

4 DISCUSSION

In this section, we discuss possible extensions of the scope of our proposal, and its challenges and limitations.

Relevance of public understanding of software and its engineering in collaborative and participatory modelling. Beyond the domain of cyber-physical systems, we can see a broader applicability of the need for an increased public understanding of software and software engineering. It is an open question to which extent the communication models we are proposing in this paper should be one-way, or if they could include means for the general public to participate in the development of systems too, by providing their input using these kinds of models. For now, such participation is not in the scope of this proposal. The collaborative aspect is most clearly shown in capturing the user's prior knowledge and information needs in models, to establish what type of information from the engineering models shall be conveyed to the user.

Is public understanding of software engineering a prerequisite? Properly interpreting the uncertainty coming from statements about quality aspects of software requires an insight into how software engineering works and why creating software is so complicated, maintenance is necessary, and failures unavoidable. Indeed, in his argument, Finkelstein states such knowledge as a prerequisite for understanding software systems [7]. In our work, we envision

that we can aim to bring the understanding in both of these categories simultaneously, by both including explanations of software systems, and by doing so educating the people of the limitations and uncertainties of summarized values for quality aspects. Any communication of quality aspects of software shall also include explanations of the scale at which these are measured and the limitations of any such measurements. We must thus in parallel explain that there are always unknown unknowns in software implementations and that it is impossible to completely check all possible states the software can be in to ensure that no unintended things can ever happen.

Public understanding of modeling? In this paper, we consider using models to improve the public understanding of software. Closely related to that, one may consider the public understanding of modeling itself. This could require, e.g., understanding of abstraction and notations. Alternatively, simple analogies may convey a sufficient explanation. For example, we expect that the general public knows about blueprints and understands that these are models of to-be-constructed buildings.

Moreover, the users will never have access to confidential engineering models directly, so to further public understanding it is not primarily important to teach the general public modeling in the model-driven engineering sense. Nevertheless, we can and do ask the question: what and how can we communicate using models? So, this work becomes more towards the suitability of models as a vessel for communication to the general public. This is relevant, especially when we consider the type of modeling in this work as a participatory activity in which we ask the members of the general public for their input in order to create the communication models.

Feasibility and limitations of this proposal. The proposal in this paper is necessarily abstract and mostly consists of a vision of future research directions. Consequently, also the feasibility of the general proposal as seen in Figure 2 is not evaluated. In general, we see a clear motivation for this work and the potential to address it with model-based technologies, but further research is needed to work out this vision in detail.

Already, we can imagine some limitations of the proposal. For example, the idea relies on engineering models to contain to some extent the relevant information that the user may ask for. This is not always in place. Moreover, it might not be relevant for engineers to add this information if the only purpose is this outward transparency. This touches also a next possible limitation: adoption by industry. Without further regulating software transparency, companies may not be so motivated to adopt these measures, even though it could make them look good (if they are working with the quality aspects in a good way).

5 CONCLUSION

We see an important responsibility for the research community to contribute to the public understanding of software and software engineering. In this work, we consider how the software and systems modeling domain can support one aspect of this broader scope, namely communicating quality aspects of software systems to their users. This paper presents a high-level proposal to achieve this goal and highlights several early challenges in this direction. We believe existing engineering models can be central in providing users insight into quality aspects of software systems, and in keeping these insights updated throughout the further maintenance and update life-cycle of software.

We look forward to discussing this topic with the community and collaboratively take steps towards improving the public understanding of software and its engineering.

REFERENCES

[1] Björn Binzer and Till J Winkler. 2022. Democratizing software development: a systematic multivocal literature review and research agenda on citizen development. In *International Conference on Software Business*. Springer International Publishing, Cham, 244–259. https://doi.org/10.1007/978-3-031-20706-8_17

[2] Wändi Bruine de Bruin and Ann Bostrom. 2013. Assessing what to address in science communication. *Proceedings of the National Academy of Sciences* 110, supplement_3 (2013), 14062–14068. https://doi.org/10.1073/pnas.1212729110

[3] Magiel Bruntink and Jurgen Vinju. 2014. Looking Towards A Future Where Software Is Controlled By The Public and Not The Other Way Round. *ERCIM News* 99 (2014), 1. https://inria.hal.science/hal-01110831

[4] ACM Code 2018 Task Force: Executive Committee. 2018. ACM code of ethics and professional conduct. https://www.acm.org/code-of-ethics.

[5] Karolina Drobotowicz, Marjo Kauppinen, and Sari Kujala. 2021. Trustworthy AI services in the public sector: what are citizens saying about it?. In *Requirements Engineering: Foundation for Software Quality: 27th International Working Conference, REFSQ 2021, Essen, Germany, April 12–15, 2021, Proceedings 27*. Springer, 99–115. https://doi.org/10.1007/978-3-030-73128-1_7

[6] Jakob Droste, Hannah Deters, Martin Obaidi, and Kurt Schneider. 2024. Explanations in Everyday Software Systems: Towards a Taxonomy for Explainability Needs. In *2024 IEEE 32nd international requirements engineering conference (RE)*.

[7] Anthony Finkelstein. 1996. Improving public understanding of software engineering. https://discovery.ucl.ac.uk/id/eprint/1136. *IEEE Software* 13, 6 (1996), 20–21.

[8] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. 2017. Collaborative model-driven software engineering: a classification framework and a research map. *IEEE Transactions on Software Engineering* 44, 12 (2017), 1146–1175. https://doi.org/10.1109/TSE.2017.2755039

[9] Eduard C Groen, Sylwia Kopczyńska, Marc P Hauer, Tobias D Krafft, and Joerg Doerr. 2017. Users—the hidden software product quality experts?: A study on how app users report quality aspects in online reviews. In *2017 IEEE 25th international requirements engineering conference (RE)*. IEEE, 80–89. https://doi.org/10.1109/RE.2017.73

[10] IEEE. 2020. IEEE Code of Ethics. https://www.ieee.org/about/corporate/governance/p7-8.html.

[11] Julio Cesar Sampaio do Prado Leite and Claudia Cappelli. 2010. Software transparency. *Business & Information Systems Engineering* 2 (2010), 127–139. https://doi.org/10.1007/s12599-010-0102-z

[12] Grischa Liebel, Jil Klünder, Regina Hebig, et al. 2024. Human factors in model-driven engineering: future research goals and initiatives for MDE. *Software and Systems Modeling* (2024). https://doi.org/10.1007/s10270-024-01188-8

[13] David Lo. 2023. Trustworthy and synergistic artificial intelligence for software engineering: Vision and roadmaps. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 69–85. https://doi.org/10.1109/ICSE-FoSE59343.2023.00010

[14] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. 2017. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software* 126 (2017), 57–84. https://doi.org/10.1016/j.jss.2016.09.015

[15] Francisco Martínez-Lasaca, Pablo Díez, Esther Guerra, and Juan de Lara. 2024. LowcoBot: Towards Chatting With Low-Code Platforms. https://fmlasaca.dev/pubs/lowcobot.pdf. In *LLM4MDE Workshop at STAF*.

[16] Michiel Renger, Gwendolyn L Kolfschoten, and Gert-Jan De Vreede. 2008. Challenges in collaborative modelling: a literature review and research agenda. *International Journal of Simulation and Process Modelling* 4, 3-4 (2008), 248–263. https://doi.org/10.1504/IJSPM.2008.023686

[17] Harald Störrle. 2017. How are conceptual models used in industrial software development? a descriptive survey. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 160–169. https://doi.org/10.1145/3084226.3084256

[18] Charlotte Verbruggen and Monique Snoeck. 2023. Practitioners' experiences with model-driven engineering: a meta-review. *Software and Systems Modeling* 22, 1 (2023), 111–129. https://doi.org/10.1007/s10270-022-01020-1