Communication based on Tools being the Message: Methodological Support for Software Project Tool Selection

Floris Wijbrands, Sietse Overbeek and Gerard Wagenaar

Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, the Netherlands.

Contributing authors: f.h.f.wijbrands@students.uu.nl; {s.j.overbeek, g.wagenaar}@uu.nl;

Abstract

Documentation is an essential part of many software projects. Unfortunately, documentation is often neglected due to a lack of interest, negatively affecting projects. Automating the process of software documentation could help in solving this problem altogether but is, as of right now, nearly impossible to achieve. This paper will introduce a software project tool selection method called SoPro-TSM that allows automatic documentation to be a part of the selection process. The idea behind SoPro-TSM is based on a concept called "tools are the message". This concept implies that, in long-lasting software projects, everything needed for documentation could be found in the tools used within the project. This paper uses this concept to introduce a method that allows for a software project tool selection that considers the message represented within the tools. Achieving this would lead to better insight into which parts can be automated. Two literature reviews and stakeholder interviews at the IT company Incentro were conducted to develop the method. The data collection results included a set of requirements and method fragments that were combined and modeled in a Process-Deliverable Diagram (PDD). After validating this PDD through several validation criteria, the concluding method could help in facilitating automated documentation.

Keywords: Software project tool selection; Automated documentation; Method construction; Process-Deliverable Diagram; Tools are the message

1 Introduction

Software development projects contain many variables that can impact the project in different ways. For example, projects may involve many or a few stakeholders, employ different development processes, or contain differences in programming paradigms like low-code versus traditional coding. Regardless of the nature of a project, one thing that does remain is the need for documentation. While its exact structure, as well as the effort that is put into it, may change, documentation remains essential for each project. Allowing stakeholders to understand and be up-to-date on the status of a project through documentation is critical for solid communication, and project success [38]. When different stakeholders are involved in a project, documentation needs could change from person to person. While one stakeholder might want to know about the programming decisions made during a project, another might want to know more about specified requirements. The process of documenting any of these things is often not dissimilar, coming down to someone having to

spend time manually noting precisely what happened [62]. In general, this documentation task is not seen as very hard or complicated yet deemed quite tedious [19]. Due to this tediousness, mistakes can be made, or documentation can be lacking completely if a project goes on for a long time while neglecting documentation. This can form an issue especially if one of the stakeholders requests documentation when it is not present or of high enough quality [1, 2, 20]. Even in this case, relevant information regarding the project can sometimes still be extracted from its tools if these tools kept track of the correct data. Theunissen et al. [66] elaborate on this by saying that *tools can contain the message* in this case. This implies that as long as an accommodating set of tools was used during a project, information that needs to be presented to project stakeholders should be accessible within these tools regardless of any manual documentation. This notion contains the potential to improve project documentation by reducing the need for manual documentation. While the idea of tools containing the message sounds promising, it needs to be sure that tools contain the right message before an attempt can be made at automating software documentation. A comparison needs to be made between what information stakeholders need and what information is contained within tools used for software projects. Software project documentation can be at least partially automated if a set of tools is selected and used within a software project that satisfies the information needs of stakeholders. Creating a method that facilitates tool selection so that this is possible is the aim of this project. As constructing a method is, in essence, a design science problem, the guidelines provided by Wieringa [76] were used for this research. Abiding by these guidelines, a formal problem statement was created to clarify the research goal:

This thesis aims to ***improve*** *software documentation* ***by constructing*** *a generalizable method for tool selection* ***that facilitates*** *the selection of tools being used for the automation of software project documentation. This method should at least consider the different types of stakeholders involved in a project and their needs to provide us with a tailored set of tools that contains as much relevant information as feasibly possible. This is done* ***in order to*** *save time and effort for stakeholders involved in a software development project as well as limit human error.*

The rest of this paper will be structured as follows: Sect. 2 will introduce the applied research approach. Sect. 3 will produce the results of the literature reviews. Sect. 4 will provide the results of the conducted interviews. Sect. 5 will describe the process and result of creating the initial method. Sect. 6 will describe the results of the validation process. Sect. 7 will cover triangulation and threats to validity. Finally, Sect. 8 will cover the conclusions and future work.

2 Research approach

As the problem statement shows, a method needs to be constructed for tool selection that can help in facilitating automated documentation. The accompanying research question that should be answered during this research is therefore as follows:

MRQ: *What methodological support can be provided for facilitating automatic software documentation via tool selection?*

Similar to the problem statement, the approach used to answer this research question was based on design science as described by Wieringa [76]. Wieringa proposes a design cycle containing three primary phases that should provide a resulting treatment upon completion. As shown in Fig. 1, the design cycle can be executed iteratively; however, this research opted for only one complete iteration due to time constraints.

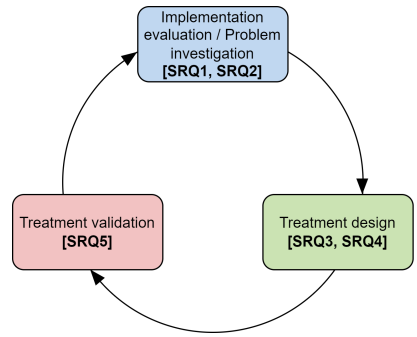

Fig. 1 Adaptation of the design science engineering cycle [76]

The first of the phases described by Wieringa is called *Implementation evaluation / Problem investigation*. While it might seem like this phase should be split into two phases, implementation evaluation and problem investigation essentially become the same thing through iterations and are therefore depicted as such. During this first phase, the problem space was investigated. This not only meant looking at any previous treatments and their effectiveness but also formalizing what phenomena needed to be improved. The second phase, *Treatment design*, is where the research artifact was designed. In this case, the artifact was an approach for tool selection, as mentioned in the problem statement. During this phase, an initial version of the method was produced. In the final phase, *Treatment validation*, the created treatment was studied to see if it could achieve its goals in a real-life setting. Verifying this allowed us to fully answer the main research question and complete the project. The application of the selected research methods to the phases of the design cycle is summarized in Fig. 2. A summarizing Process-Deliverable Diagram (PDD) can be found in Appendix A.

2.1 Problem investigation

As there are no previous implementations of this specific method to evaluate, the first phase focuses solely on the problem investigation. Wieringa [76] explains that a problem investigation aims to look at the problem in detail before establishing requirements for the treatment. As such, the goals of a problem investigation can be described as identifying, describing, explaining, and evaluating the to be treated problem. Two sub-questions were posed to capture the essence of what should be investigated during this phase.

- **SRQ1:** *What are the current characteristics of software project documentation?*
- **SRQ2:** *What decision factors currently go into the tool selection process for software projects?*

2.1.1 Exploratory literature review

An exploratory literature review was performed as a first step towards completing the problem investigation. This review served to provide a basic understanding of the problem space and start answering the research questions that were created. The full extent of the review was performed by searching through Google Scholar and using snowballing to find an expansive array of sources. The results of this review can be found in Sect. 3.1.

2.1.2 Interviews

Expert interviews were conducted to add to the literature review and finish off the problem investigation phase. These interviews were performed at three different branches of the IT company Incentro. More information on this can be found in Sect. 2.4. The selection process for the interviewees included a brainstorming session with a team lead and operations manager at Incentro to identify stakeholders and potential interview participants. Within this session, ten participants were selected, eight of whom ended up available for an interview. An overview of the final participants can be found in table 1. The interviews themselves were conducted in a semi-structured way, including an interview protocol that can be found in Appendix B.1. All participants also signed the informed consent form that is shown in Appendix B.3.

2.2 Treatment design

To guide the construction treatment design, two additional sub-questions were posed.

- **SRQ3:** *What method can be constructed for eliciting documentation needs in different software projects?*
- **SRQ4:** *How can we use the method from SRQ3 to select a complementing set of tools?*

2.2.1 Systematic literature review

Before starting the construction of the eventual method, one final systematic literature review was performed. The research protocol for this review can be found in table 2 and follows the guidelines provided by Okoli [47]. While there was no specific research question coupled with this review, its main purpose was to find applicable method fragments as well as criteria for the to be constructed method. To decrease threats to validity, the research queries were reviewed several times before deciding on a final set. Consideration was also made on which databases to apply the queries.

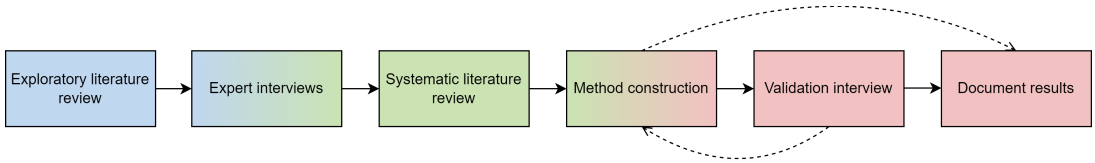

Fig. 2 Summarization of the research process

Table 1 List of interviewees

Name	Branch	Occupied function	Exp. in years
IntervieweeA	Data analytics and OutSystems (DOS)	Project manager	0.5
IntervieweeB	Data analytics and OutSystems (DOS)	Data engineer/developer	8
IntervieweeC	Digital Information Services (DIS)	Solution architect/project lead	12
IntervieweeD	Data analytics and OutSystems (DOS)	Project manager/delivery manager	3
IntervieweeE	CLOUD	Cloud architect/IT consultant	8
IntervieweeF	Digital Information Services (DIS)	Developer	10
IntervieweeG	Data analytics and OutSystems (DOS)	Solution architect/project lead	11
IntervieweeH	Data analytics and OutSystems (DOS)	Data engineer/developer	5

While opinions on the best approach differ, there is evidence to suggest that using Google Scholar as a sole database can provide a 100% coverage of research and this option was therefore chosen [24]. The results for this review are described in Sect. 3.2.

2.2.2 Process-Deliverable Diagrams

The creation of two methods as demanded by the research questions can be a complicated thing. Method engineering research provides some guidance as to how to construct such methods; however most of these projects describe situational methods, whereas this project aims to create a generalizable method. As such, strictly applying these methods to this research was deemed inappropriate. Instead, the expert interviews were used to extract method requirements. Method fragments were then used to create a method that fits these requirements. To represent and validate the constructed method, a PDD will be used. While the concept of a PDD has been covered in various works, the guidelines presented by Weerd et al. [70] appear to be the most complete and will therefore be referred to for this project. As its name implies, a PDD is a diagram including two main components, one on each side of the model. The first side of the model concerns the process and displays all activities that will take place within the method, similar to a UML activity diagram. The second side is similar to a UML class diagram. It concerns the deliverables or concepts that result from the aforementioned activities. By creating a PDD, the order of activities and their relation to resulting deliverables should become more apparent. To ensure everything within the model is understandable, accompanying activity and concept tables were made to explain all the terms used within the PDD.

2.3 Treatment validation

For the final phase within the design cycle, one more sub-question was posed.

- **SRQ5:** *How can we validate and eventually implement the newly constructed method?*

Within the treatment validation phase, it needs to become clear whether the constructed method is feasible and would support stakeholder goals when implemented. Wieringa [76] recommends doing this through the use of a validation model. As the name implies, a validation model is a model of the created artifact that can be used to validate the use of this artifact. By studying the validation model, design theories can be created which can help in predicting the effects that implementing the created artifact would have in the real world. Wieringa proposes several methods to validate a design; however, due to the availability of experts as well as recommendation by Weerd et al. [71], expert opinion was considered the most

Table 2 Systematic literature review protocol

Component		Description
Purpose		Find criteria and/or method fragments of software project documentation processes. Find criteria and/or method fragments of software project tool selection processes.
Practical screen	*Inclusion criteria*	1. The research is applicable to the software industry. 2. The result is not a duplicate of other research. 3. The research is not an older version of another result. 4. A digital copy of the research is available to the researcher. 5. The research is written in a language in which the researcher is proficient.
Search strategy	*Search terms*	"software project documentation", "software development documentation", "software project", "software development", "tool selection", "method", "criteria", "framework"
	Resources	Google Scholar
	Search queries	1. ("software project documentation" OR "software development documentation") AND ("criteria" OR "method" OR "framework") 2. ("software project" OR "software development") AND "tool selection" AND ("criteria" OR "method OR "framework")
	Search limits	To limit the amount of outdated or irrelevant research as well as to limit the search results to a reasonable amount, only query results from 2010 and up were considered.
Study quality assessment		1. The research is part of a peer-reviewed journal or conference proceedings [60]. 2. The research contains a clear scientific method. 3. The research is elaborated to an extent where it is deemed understandable.
Synthesis of extracted data		Methods, as well as criteria, are grouped and compared for overlap.

practical and accurate option available for this project. According to Wieringa [76], validation through expert opinion is done by presenting the design of an artifact to one or multiple experts who can imagine how this artifact will interact with imaginary problem contexts. In this case, the validation model would exist in the expert's mind. As long as the experts understand the artifact, can imagine realistic problem contexts, and make reliable predictions of the effects it will have, this method can work well for validation. To provide some structure to this validation, evaluation criteria defined by Prat et al. [51] were used during validation. Since their original list of criteria is quite extensive and not all of them are relevant for this research, the selection of criteria that was used for this research can be found in table B1. Obtaining expert opinions can be done in multiple ways. For this research, the proposed artifact was presented to practitioners at Incentro as well as a method engineering expert at Utrecht University during semi-structured interviews. As an alternative to single-person interviews, focus groups were also considered as recommended by Venable et al. [72] for ex-ante, naturalistic evaluation. This idea was, however, discarded due to a lack of significant benefits [26, 63] and scheduling constraints.

2.4 Research environment

This research was performed as a graduation project for the Master's program Business Informatics at Utrecht University. As a result of this,

resources available through the university were used where possible. One example of this was the availability of a method engineering expert for the treatment validation. Aside from its ties to the university, this research was carried out in collaboration with the Dutch IT company Incentro. Incentro is a company currently active in the Netherlands, Spain, and Kenya that works on various digitalization projects across the world. While it would have been beneficial for this research to work with multiple companies, Incentro provided a unique benefit in its structure that made it a good alternative. This is because each branch at Incentro essentially works as an autonomous cell with its own policies, processes, and projects. In this way, looking at different branches within Incentro is much like looking at different companies. Throughout this research, the scope of the interviews will focus on the larger, ongoing projects within these branches and the experiences from those working within them to maintain focus on the relevant subject matter. The selected branches for this research project were Digital Information Services, Data & OutSystems, and CLOUD. These branches work on software-related projects ranging from creating small or large data solutions for other companies to implementing existing systems. The varying environments for projects, as well as the availability of field experts, made Incentro a solid environment in which to perform this research.

3 Literature review results

This section will discuss the results obtained from the two literature reviews performed throughout this project. The exploratory review aimed to create a foundation for the knowledge base on which to build, and the systematic review served to supplement and finalize this knowledge base. The results of the exploratory review are primarily descriptive and therefore depicted as such. The systematic review aimed to collect method fragments and criteria from various research, which could be summarized in tables.

3.1 Exploratory review

Throughout the exploratory literature review, several topics are covered that together provide an overview of the research domain as well as highlight the gap in research that this research aims to fill. The first part of this review covers the concept of software documentation in general and goes into the current characteristics of its processes. The second part refers to the current efforts towards automating the process of software documentation. Finally, the concept of tools are the message is introduced to show how a new approach could make steps towards fully automating software documentation via tool selection.

3.1.1 Documentation characteristics

At its core, the process of software documentation can be described as "collecting, organizing, storing, and maintaining a historical record of programs and other documents used or prepared during the different phases of the life cycle of the software" [9, p. 2563]. While this description captures part of the essence of what software documentation is, the exact purpose or goal of documentation is still often debated. One elementary way to describe the role of documentation in a project is that it should communicate information from software engineers to each other and their project's stakeholders [12]. Alternatively, Ambler [6], and Cockburn [10] proposed that the role of software documentation should not be to provide information but more to convey knowledge, meaning a reader should be able to understand the system instead of just be presented with tidbits of information. Aghajani et al. [2] substantiate this school of thought by performing a survey study among 146 professional practitioners on the usefulness of software documentation. They found that explanatory documentation like code comments can be especially beneficial for introducing new team members to a project and allowing them to get familiar. While neither of the described roles of software documentation is more accurate than the other and a perfect description does not exist, it is essential to a project's success to realize that different stakeholders might require different things from their documentation [74].

Because software documentation can be such a broad concept when looking at a full software project, it might prove insightful to split it up into

smaller parts. Sommerville [61] did this by splitting documentation into two main categories and two subcategories:

- **Process documentation** can be described as all documents that record the process of development and maintenance. They are most often relevant for managing parties and include project plans, schedules, and quality documents. These types of documentation are often created during the beginning of a project and serve as a guide for the rest of the project.
- **Product documentation** includes all documentation that has to do with the product in development. Within this category, a further distinction can be made between system documentation and user documentation.
  - **System documentation** describes the product from the developers' point of view and is used during development and maintenance. Some documents belonging to this term include requirements documents, system architecture, component functionality, validation documents, and maintenance guides.
  - **User documentation** is constructed to provide a simple description of the product intended for its end-users. This user documentation usually includes things like manuals.

A summarizing model can be found in Fig. B2.

While documentation can be used throughout a project, research by Hager [28] suggests that maintenance is the single most crucial phase for documentation usage. The structural integrity of the documentation that is used during any phase of a project becomes more critical if a project lasts a long time or even indefinitely. One reason for this is that good documentation makes handing over projects to new teams or introducing new team members to the project much more accessible [23]. As one can imagine, this is important for more extensive projects since there is much more to explain, which cannot all be done in person. Another subconscious benefit of software documentation was posed by Dagenais et al. [13] in the form of improved code quality. They stated that if every change made by a developer was documented, a form of embarrassment-driven development would ensue. The idea behind this is that the developer subconsciously feels like they are performing a demo of some sort because what they are doing is documented. This, in turn, means that they will try and create the best code they can for this demo.

Considering the previous explanation of the software documentation, it might seem like everything is already going smoothly. In reality, there are unfortunately some issues regarding the execution of software documentation that cause it to be less effective than it could potentially be [1].

Perhaps the most detrimental factor to software documentation is the lack of effort from developers. While the importance of software documentation is mostly undisputed among participants of a software project [78], the task itself is often still neglected. This problem can potentially impact the documentation in three significant ways: up-to-dateness, accuracy, and completeness [22]. Survey research by Forward et al. [20] found that most documentation is rarely updated, and 68% of the 48 participants found across several high-tech companies stated that documentation was always outdated. Documentation regarding software testing was the only thing that seemed to be updated frequently enough. It is important to note that, even though documentation was often considered outdated, participants still found themselves referring to it quite often. In practice, this could result in mistakes that can even be more costly than not having documentation at all, according to research by Poston [50].

Capers [34] found that flawed process documentation was often the most costly when not done properly. They analyzed approximately 250 large software projects and found common problems in the documentation among those that eventually failed, including poor planning, milestone tracking, and quality control.

Another thing that can impact the usability of software documentation is structure. Research by Das et al. [14] and Treude et al. [68] found that the overall quality of documentation becomes worse as a project goes on. Their reason for this is that there is often simply too much. New documents are created regularly, which means that eventually, even if everything is up to date, it might be hard for a stakeholder to find what they are looking for unless a solid structure is in place. Along the same lines, working in a single document can also cause it to become too long and convoluted

[18, 69], causing the same problem as mentioned before. Ideally, stakeholders should be easily able to find documents and, after finding them, quickly identify and anticipate what is in them [32].

The amount of documentation is not the only problem that arises as a software project continues. The phase of a project, if not handled correctly, can also cause some trouble in the documentation. As a project progresses through different phases, like construction or maintenance, documentation needs may change [6, 10]. Documentation during the construction of software will often require a different substance than the documentation during maintenance of the same software simply because of the tasks and people involved. These differences should be taken into account when attempting to automate documentation. A final problem issue in software documentation is that it is hard to establish standards. While most people would agree that the documentation is important, it is hard to pinpoint precisely which elements of documentation are important [15]. One reason for this is that the importance of some aspects in the documentation might change depending on who is asked. Another could be the project phase or even the nature of the project itself. Taking extra steps to elicit this information might prove to remedy this problem somewhat.

3.1.2 Automating software documentation

After looking at what software documentation is and what its current issues are it, is reasonable to say that automating the documentation process could prove very useful. This conclusion has been drawn by researchers in the past and several initiatives have been made towards this goal in different areas of documentation [20].

For example, Hu et al. [31] developed a method called DeepCom, which uses natural language processing (NLP) to add comments to large amounts of code. Other research projects [41, 44] focus more on global documentation by developing methods able to summarize methods and classes respectively to provide programmers with quick insight into code. The use of previously existing text summarization techniques has also been studied by Haiduc et al. [29], who used a combination of different techniques on source code and found that together, they were able to generate a satisfying summary of the code. Finally, McBurney et al. [42] developed a technique using topic modeling that can make a hierarchical summary of all topics within the code, allowing users to look at the highest level functionality without having to look at each topic individually.

A critical part of the software construction phase is unit testing. During this process, developers test a specific part of the software to see if everything works as intended. Because unit testing is a task that needs to be performed quite often, attempts have been made to automate it. The result of these attempts was often that there is not much time to be saved on automated unit testing since the developers have a hard time understanding the output of these automated tests [21]. This is another area where automated documentation can provide aid. This sentiment is shared by Panichella et al. [48], who produced TestDescriber, an automated documentation approach for summarizing automated unit tests. Another effort in this area was made by Li et al. [35], who designed a similar approach called UnitTestDescribe. UnitTestDescribe uses statistic analysis, NLP, backward slicing, and code summarization to generate documentation for test cases.

After unit tests are performed and documented, changes in code on a personal machine can be committed to keep everyone up to date. While doing this, adding a commit message to describe what changes were made is a crucial part of the documentation. Especially for maintenance, knowing what was changed in which commit is essential. As is a trend among the previously mentioned topics, these messages can also be automated. Several studies have been performed to approach this goal and tools like ChangeScribe [11] and ARENA [45] have been developed that automatically write commit messages. Both these approaches have been tested and were judged to often include more helpful information in messages than those written by developers themselves.

Aside from automatically documenting output, formally documenting input is also an option. Multiple research projects [36, 39, 54] opted for this idea by developing methods and techniques to automatically summarize incoming bug reports to only show relevant information. While this does not necessarily fall within the scope of this project, it is good to realize that these things exist, and it

might be interesting to incorporate them in future research.

Alternatively, some attempts have also been made at automating documentation for people other than programmers specifically. For example, Rodeghero et al. [57] proposed a method for automatically extracting user stories from conversations between a developer and client. If done correctly, this would eliminate the need for taking notes within these conversations.

3.1.3 Tool selection and tools being the message

We have seen in the previous section that many attempts at the automation of documentation have been made. Some of them are included in industry-standard tools, and others are not, but much progress has been made nonetheless. What we can learn from all these attempts at automation is that, realistically, everything we need to know for the software documentation is often available in the tools that are used for the project. This idea that tools can contain the message that needs to be documented is also presented by Theunissen et al. [66]. They adapted this idea from McLuhan et al. [43], whose original statement was that the medium is the message. In this case, the phrase tools are the message can be seen as an application of the phrase "the medium is the message." Since both concepts are similar and tools are the message is more recent and applicable to this research, this is the phrasing that will be used from now on.

The research done by Theunissen et al. [66] concludes a few things:

1. There is a strong relationship between the tools used and the type of information produced. This considers the content, format, and relevance (why it is stored).
2. Tools are organized into tool stacks which in turn are organized into software development ecosystems.
3. The variety of tools refers to the amount of structure for information.
4. The previous three conclusions make a difference in creating, retrieving, communicating, and understanding "the message" which in our case is just software documentation.
5. The focus on information from specific tools for each stakeholder decreases the complexity of the ecosystem and makes it easier to understand.

Of these conclusions, the first one is the most relevant for this project. It shows that different forms of information are produced depending on the tools used. When using the information that is stored in tools for automation, it is therefore critical to select the right tools. Otherwise, the message contained in the documentation will change.

While the idea that tool selection can have an impact on the success of a project is not unreasonable, surprisingly little research has gone into how tools are currently selected and how this could be done better [3]. There is, however, no shortage of studies that evaluate what companies and practitioners find most important during the tool selection process. Ease of use [4, 46, 55, 56, 64, 73] and pricing [55, 56, 64] are the aspects that come by the most often as a contributing factor in tool selection. Nevertheless, the need for better reporting capability is also something that is desired by practitioners, especially in agile software development spaces [7]. While these requests do not necessarily mean that this aspect is more or less important than any other factor, it does mean that selecting tools for documentation and reporting automation could be very beneficial. Some other notable criteria for tool selection include: the possibility of collaboration [3, 46, 56], vendor responsiveness [64], reliability [55], and security [46, 55].

3.1.4 Summary

From this first part of the literature review, a few conclusions can be drawn that are part of answering SRQ1 and SRQ2. Starting at the beginning, there is essentially no such thing as the perfect documentation approach. There are differing views on what documentation of a project should include and what can make documentation good or bad. Yet, ultimately, it will differ from person to person and from project to project. We also know that in many projects, the current documentation is not good enough. The documentation lacks in areas such as up-to-dateness, accuracy, completeness, and structure. An ideal implementation of automated documentation would aim

Table 3 Guiding principles for documentation elicitation

Documentation priciple	Description	Reference
Stakeholder preference identification	Make sure stakeholder preferences regarding documentation are identified.	[15, 16, 59]
Characterization	Characterise documentation preferences of common stakeholders within a project to save time.	[16]
Protocol comparison	Compare the documentation protocol to protocols of other, successful, projects.	[33]
Checklist compilation	Create forms or checklists that can be verified.	[27]
Role dedication	Assign dedicated roles and reviews to enforce agreements.	[52, 53]
Cyclic documentation refactorization	At the end of a project cycle, refactor documentation and decide what will be needed for future cycles.	[65]
Preemptively choosing a text format	Choose a documentation format that will work for the project, preferably formats that withstand time like plaintext.	[75]

to alleviate these problems. Practitioners' statements and statistics show us that the problems in documentation can be costly and waste significant amounts of time, which are things any project team would rather avoid. Several initiatives have already been made to pursue the goal of documentation automation. While some have been effective, there is little effort to automate more than just a single aspect of documentation at one time. Tools are the message provides us with a reason why this has not yet been done and gives some idea as to how progress can be made in this area. To fully automate documentation via tools, we first need to select the right set of tools or software development ecosystem to facilitate. Devising a method that can consistently provide this selection has not yet been done for this purpose and could therefore carry significant academic and practical potential. While the selection of tools for automated documentation could have a high priority, there are several other criteria , like ease of use, pricing, and security, that are often just as important.

3.2 Systematic review

To finalize the knowledge base needed for constructing an initial treatment, a systematic literature review was conducted as per the protocol described in table 2. This review essentially consisted of two parts; to search for criteria and method fragments for documentation needs elicitation and to search for criteria and method fragments for tool selection. Unfortunately, while there were a fair amount of method fragments to be found for tool selection, documentation needs are a subject that is glanced over by most literature. The topic of software documentation has been researched, and different approaches towards documenting are readily available; however, none of these approaches specifically mention how to decide what should be documented. Because of the lack of method fragments to be found, a list of principles and criteria that should be kept in mind during the process was compiled instead. As shown in table 3.

For tool selection, several applicable methods were found. While using every single method within the treatment would be inefficient and unnecessary, a summarizing description of each method can be found in table 4.

4 Interview results

Alongside the literature reviews, interviews were used to supplement the knowledge base. Three main topics were discussed during these interviews. First off, the topic of software documentation in general and its current implementation was discussed. Next, automated documentation was discussed, including any experiences already had as well as the consequences of implementing some form of an automated system. Finally, some effort was put into eliciting goals or requirements for the

Table 4 Tool selection methodologies

Code	Description	Reference
M1	A method applying the Analytical Hierarchy Process (AHP) to the tool selection domain in three stages: structuring complexity, measuring on a ratio scale, and synthesizing results.	[3]
M2	A method aiming to identify the most important features of a tool and then filter out tools that do not contain these features. Any tools left are subject to a standard comparison on selection criteria.	[64]
M3	A method featuring a semi-automated tool recommender which uses AHP to obtain the weight of each selection criteria and then uses Multi-Attribute Utility Theory (MAUT) to recommend a tool or toolset.	[49]
M4	A method using a weighted decision matrix to represent the availability of certain desired features within tools and select optimal candidates.	[58]
M5	A method containing four phases for the selection of tools. These phases include justifying and initiating the need for tools, developing specification and evaluation criteria, investigating tools and selecting the best candidate toolsets using an algorithm, and implementing and maintaining the solution.	[37]
M6	A method providing business process-oriented tool selection in four steps. These steps include the identification of relationships among project goals, critical success factors and criteria, the identification of relevant tools, the development of an AHP model, and final decision making.	[8]
M7	A method providing a seven-step guide for selecting tools that support traceability. These steps include agreeing on the problem, understanding the problem, identifying stakeholders, determining requirements and constraints, designing a requirement management system, assessing and selecting tools, and planning the tool introduction.	[25]
M8	A method providing a decision framework allowing users to prioritize tool features using the MoSCoW technique to assign weights to the criteria.	[17]

to be designed treatment. To allow for traceability of the evidence produced within the interview, a coding scheme was created. The scheme for the first set of interviews can be found in table C2.

4.1 Documentation process

As three different branches were interviewed throughout this project, three slightly different takes on project documentation were discussed. Out of the described processes, those of DOS and DIS were the most similar. This is a logical consequence of the fact that these two branches only recently split up and used to be a single branch. Participants from both branches also mention that while they do document, there is no clear process, which sometimes causes problems [iv-1, iv-2, iv-3, iv-6, iv-7]

The DOS documentation process starts with a meeting with the customer to discuss the project parameters. One of these parameters could be the documentation; however, if the client themselves do not find this very important then it is often not discussed to a significant extent. For technical documentation, agreements are made between developers depending on what they encounter during the project [iv-2, iv-8]. This documentation is often separated over multiple channels like Teams, Google Drive, or Confluence making it sometimes hard to find [iv-7, iv-8]. Aside from doing this technical documentation, most of the interviewees mentioned that documentation is rarely updated. Unless someone specifically asks for it, it remains outdated until the end of a project where it is updated for the final delivery [iv-2, iv-4, iv-7, iv-8].

DIS attributes even less value to documentation as, based on the nature of their projects, their documentation can be more inherent. As such, most of the documentation is written near the end of the project [iv-6]. An exception to this is in-code documentation which is often based on mutual agreements [iv-3]. One major problem DIS

projects encounter is that documentation needs are often formalized too early within a project and might later prove useless [iv-3].

The documentation process within CLOUD entails four documentation types [iv-5].

- **Solution documentation** is created at the start of a project and concerns whatever is discussed at day-0. This includes project plans which are updated whenever a change is made.
- **Development documentation** is created during the project itself. Writing this documentation should be done per feature. Whenever a new feature is introduced, accompanying documentation should also be produced. What this documentation entails is often subject to the author's interpretation.
- **Installation and configuration documentation** is produced whenever a new feature or update is rolled out and includes everything that is needed for the installation and configuration of a product.
- **Maintenance documentation** is built towards the end of a project and should include everything that is required by the maintenance team. Some communication about this documentation is necessary; however, most of it is often again subject to the author's interpretation.

Due to a lack of structure within the documentation process, many improvements could be made according to the participants. Some of the current issues faced at Incentro are a lack of standards [iv-1, iv-2], spread out documentation [iv-2, iv-7, iv-8], lack of effort from individuals [iv-5, iv-7, iv-8], miscommunications [iv-3], and a lack of completeness [iv-3, iv-7, iv-8].

4.2 Automated documentation tools

While every participant had at least heard of attempts of automated documentation, not all of them actually had experience with them. Those that did were mildly enthusiastic but did identify some issues that caused them not to be incorporated in their current work. The first and most common complaint was that the initial implementation of the automation was too much work, and there were no volunteers to do it [iv-2, iv-5, iv-8]. Because of this, using automated tools would only be helpful for tasks that take a long time or are done continuously. A second problem encountered was that automatic documentation was hard to implement in projects with little structure. This is especially a problem within shorter projects at Incentro, as identified earlier [iv-8]. Even if some form of structure was present, it could still prove challenging to make people follow this structure [iv-4]. Another problem with previous attempts was that there was no focus within the documentation, and it became too elaborate to be beneficial [iv-6, iv-7].

While the mentioned problems currently prevent implementing automated documentation, most participants did recognize its potential upsides. Some participants mentioned the ability to prevent human error [iv-2, iv-7], while others found the ability to optimize handovers more important [iv-6, iv-8]. Most of all, the possibility of saving time [iv-1, iv-4, iv-5, iv-7, iv-8] and making sure documentation is up-to-date [iv-2, iv-4, iv-7, iv-8] were mentioned.

Opinions on the most important stakeholders for such implementations differed as well. Some found it most important to the client [iv-5, iv-8], others found developers to be the primary beneficiary [iv-2, iv-3, iv-7], and some said that all newcomers to a project were most likely to benefit from a solid implementation [iv-6].

4.3 About the method

Overall, most participants agreed that the treatment method should be executed by someone internal. They based these conclusions on variables such as cost, maintaining internal knowledge, and the odds of accepting the new way of working. Other than this, opinions on how to execute the method were divided. Half of the participants found some form of training to be acceptable [iv-1, iv-2, iv-3, iv-5], while the other half felt that a simple guide should be enough [iv-4, iv-6, iv-7, iv-8]. While all participants recognized the eventual end goal of automating documentation, they also identified several goals that could be achieved within this treatment. The most mentioned goal of the method was to allow for some form of standard or minimal requirements within the documentation [iv-1, iv-3, iv-6, iv-7]. Along the same lines, many participants found creating a predefined way of working for documentation to be important [iv-2, iv-7, iv-8]. Other, less-mentioned

Table 5 Elicited requirements.

Code	Description	Source
AR1	The treatment should include some efforts in creating or maintaining documentation standards.	[27, 33, iv-1, iv-4, iv-7, iv-8]
AR2	Stakeholders should be included in the process to project their needs.	[15, 16, 59, iv-3, iv-6]
AR3	Stakeholders should be included in the process to verify conclusions.	[iv-1]
AR4	If automated documentation fails to be implemented, a manual takeover should be possible.	[iv-4]
AR5	Upon completion of the treatment there should be clear agreements on the application of documentation	[iv-2]
AR6	Once candidate tools are selected, no more time should be spent looking for the perfect tool.	[iv-7]
AR7	There should be room for evaluation included in the treatment.	[65]
AR8	The treatment should produce documentation that is suitable for onboardings and project handovers	[iv-2, iv-6, iv-7]
IR1	The treatment should limit interfering with the usual way of working.	[iv-4, iv-5]
IR2	The treatment should be beneficial even if not all employees participate.	[iv-4, iv-5]
IR3	The treatment should account for the changing documentation needs during different phases of a project	[6, 10, iv-5, iv-3]
IR4	The treatment should not require any prerequisite knowledge to execute.	[iv-1, iv-2, iv-3, iv-4, iv-5, iv-6, iv-7, iv-8]
IR5	The treatment should contain dedicated roles of the people who execute it	[52, 53]
IR6	The treatment should produce the right amount of documentation.	[iv-1, iv-3, iv-7]

goals were to simplify work [iv-1, iv-5], create acceptable documentation in one try [iv-3, iv-4], and improve efficiency [iv-2]. Finally, some challenges were identified that could limit treatment's performance. Most of all, getting the stakeholders to accept the new way of working could provide struggles [iv-1, iv-4, iv-5, iv-6, iv-8]. This is especially true if the treatment is invasive and not easy to execute [iv-8]. Other than this, the cost of changing tools caused some worry [iv-2], as well as spending too much time finding the perfect tool [iv-6].

5 Creating the initial method

After completing the knowledge base through the interviews and literature review, method construction could be started. A shortlist of requirements for the treatment was extracted from the knowledge base, which can be found in table 5. Within this table, requirements labeled starting with an A should be related directly to an activity within the PDD, and requirements labeled with an I are inherent to the treatment. A new methodology was then created abiding by these requirements where possible using method fragments that were found during the interviews and literature review. The resulting PDD can be found in Figure 3.

The proposed method contains three major phases. The first phase is executed upon starting a new project and includes the steps necessary to discover the documentation needs for a project, corresponding to SRQ3. This process includes defining the project itself, interviewing stakeholders, and creating documentation standards. Following the documentation needs elicitation, a preferred set of tools should be selected

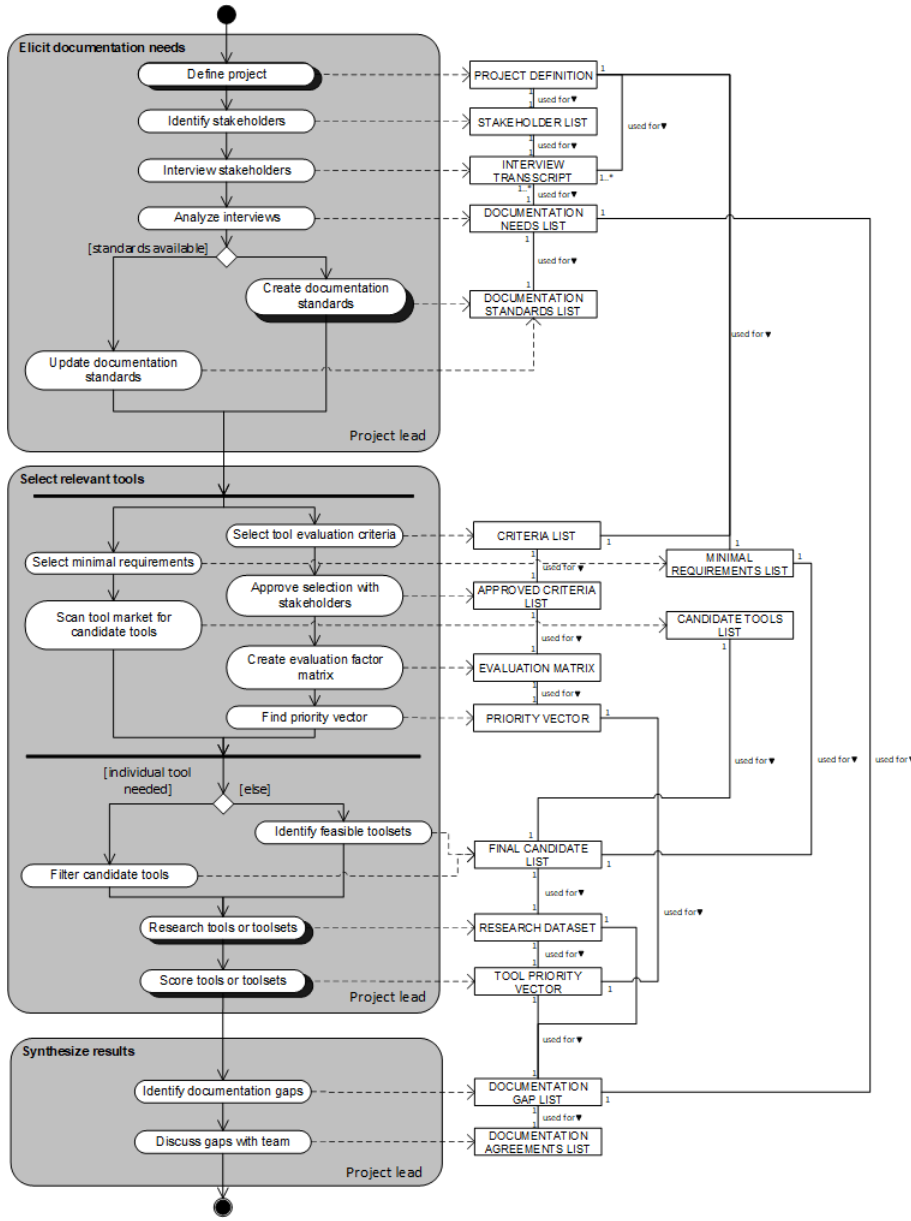

Fig. 3 PDD for the main method

corresponding to SRQ4. This next phase prescribes looking for relevant evaluation criteria like automation prospects or tool costs and scoring them on relevance using AHP. Alongside this, minimal requirements for the tool or toolset should be selected to provide a list of candidate tools. Finally, the tools should be researched and scored on each of the evaluation criteria to find the final preferred toolset. The method is concluded when synthesizing the results by identifying what documentation can or cannot be automated using the selected toolset and communicating these gaps in automation with the project team.

6 Validation

To validate the designed treatment, four expert interviews were conducted. Three of the experts interviewed were domain experts working at

Incentro as a developer [viv-2], delivery manager [viv-3], or project manager [viv-4]. The final expert worked as a teaching assistant for the Method Engineering course taught at Utrecht University [viv-1]. During the interviews, the experts were presented with the initial PDD and asked to what extent the proposed treatment satisfies each criterion in table B1. To guarantee a comparable output between participants, a five-point Likert scale was used during the interviews. While the exact scores for each criterion can be found in E, an average score was compiled in table E6. Within this table, a score of 4 would mean that the participants completely agree that the criterion was fulfilled. In contrast, a score of 0 would mean they completely disagreed. As can be seen, most criteria obtained a score of at least three, meaning that the participants, on average, agreed that they were fulfilled.

Table 6 Average scores for each validation criterion

Criterion	Score
Efficiency	3.5
Utility	2.75
Operational feasibility	2
Absence of side effects (people)	3
Absence of side effects (organization)	3
Ease of use	3
Completeness	3
Adaptability	3.75
Modifiability	2.75

The main conclusion during viv-1 was that the created model was structurally sound. Some final kinks in naming and modeling conventions were ironed out, but nothing substantial needed to be changed. The interviews with domain experts yielded more significant conclusions. As could be deduced from the answer "neither agree nor disagree" most of the criteria that were scored close to two had the driving reasoning that it was hard to judge. Starting with operational feasibility, the interviewees believed that while the potential upside and the projected adoption of the method are great, the actual upside is currently not that significant. Because of this reliance on future work, it is hard to state exactly how much the method would be incorporated into a routine, and the interviewees were hesitant to answer. What they did agree on was that the suggested execution of the method might take a lot of time which could cause resistance [viv-2, viv-4]. The same thing that applies to operational feasibility also applies to the utility of the treatment. Since it is hard to assign a value or cost to the treatment, it is also hard to answer this question [viv-3]. Finally, the modifiability of the method was deemed as hard to judge as well by some candidates [viv-2, viv-4] since they found themselves unfamiliar with the modeling environment. While all other criteria rated by the experts were scored as a three or higher, some changes were still suggested to make the model more applicable. Firstly, an extension needed to be made to the model in which the algorithm [viv-2] was elaborated. Aside from this, the suggestion was made to update documentation standards during the end of the project instead of at the start of the method [viv-4]. Finally, as suggested previously, the algorithm that is used to score tools was changed from AHP to MAUT. While the original purpose of using AHP for both parts of the tool selection was to make the method easier to understand, the interviewees suggested that saving time in the process was more important. One final thing to mention is that some of the participants [viv-4], as well as some literature [17, 49], suggested using a database for saving information about tools. While this could potentially save a great amount of time during tool research, creating such a database, if it is not readily available, could prove too costly and was, therefore, not considered for this treatment. To highlight the changes that were made as a result of the validation, a method evolution PDD was created. The notation for this construct, as well as the PDDs, can be found in Appendix D.1. The updated and final version of the PDD, including the accompanying concept and activity tables, can be found in Appendix D.2. For the sake of traceability, the activity table for the final method contains the codes of the applicable requirements that were defined in table 5. Aside from this, the table also includes references to the method fragments that were used as defined in table 4.

7 Discussion

7.1 Triangulation

To increase academic validity throughout a qualitative research project, triangulation can be

included. Triangulation can increase the confidence in findings by using two or more independent measures to confirm a proposition [30]. Thurmond [67] describes several categories of triangulation, two of which apply to this project.

Methodological triangulation reduces the deficiencies and biases that are present when conducting research using a single method. This type of triangulation was primarily performed during the problem investigation phase. Two types of research were performed, including the literature review and interviews, which primarily resulted in the same conclusions. While describing the exact overlap between the two methods is excessive, some of the more important findings were that both methods identified a lack of interest as the main contributing factor for documentation quality suffering. Also, both identified that automation efforts are available but are often in early stages and therefore hard to introduce. No significant contradictions were found between the two methods.

Data source triangulation is the use of data gathered from different times, spaces, or people to come to a conclusion. This was achieved several times during this project. First off, different research was gathered during the literature reviews giving access to multiple different viewpoints when drawing conclusions. This was extended to the interviews where several people with different positions within Incentro were interviewed to obtain a broad view of the topic. Finally, to construct the method, several method fragments were used from differing research projects. These three things provide ample data source triangulation for this project.

7.2 Threats to validity

As is the case in all research, some threats to the validity of this research can be identified. While it is impossible to eliminate all threats in qualitative research completely [40], measures were still taken to reduce the impact of these threats. Maxwell [40] identifies five categories of threats for qualitative research:

1. **Descriptive validity:** *Are the findings of the research accurately portrayed?* Potential threats to this validity could be mishearing or mistranscribing an interview. Some steps were taken to minimize the risk of this happening. Interviews were recorded and transcribed at a later date to avoid too much distraction. To maintain focus during the interviews, simple notes were taken instead. Additionally, interviews took place in neutral and peaceful settings as much as possible to avoid external factors interfering. While this was hard to maintain due to interviews mostly taking place in an online environment, no significant distracting events were noted. These three measures allowed for a straightforward transaction of information during the interviews.
2. **Interpretive validity:** *Are the observations made within the research interpreted correctly?* To attempt to avoid misunderstandings during interviews, all feedback, and information gathered was be discussed extensively. Alongside this, any scales or rating systems that were used to validate or evaluate aspects of the method were discussed and explained beforehand.
3. **Theoretical validity:** *Is the right theoretical structure being used to explain observations?* Maxwell [40] describes theoretical validity as some combination of the more well-known aspects of construct validity and internal validity. While they do not mention any specific ways to combat theoretical invalidity Yin [77] does provide some safeguards for construct and internal validity. One thing they recommended is using multiple sources of evidence to support work. During this research, this was achieved by combining the information obtained within literature research with the information from conducted interviews as described in Sect. 7.1. On top of this, an expert review of work is recommended. The validation interviews that will be held during this research should work towards this goal.
4. **Generalizability:** *How extendable are the accounts of particular situations to other potential subjects?* This research aimed for the results to be at least somewhat generalizable across the software engineering domain. The method that was built includes steps of eliciting stakeholders and their needs which should help in accounting for different scenarios that can occur in different companies. Aside from this, participants of interviews were picked to be diverse, which should benefit the generalizability of this research. Finally, one thing that

could be seen as a significant threat is that the interviews are mostly conducted within one company. While this threat is unfortunately unavoidable at this time, Incentro does provide somewhat of a special case as addressed in Sect. 2.4.

5. **Evaluative validity:** *Are all drawn conclusions based on gathered data?* Maxwell [40] identifies evaluative validity as perhaps the least relevant aspect of validity to qualitative research since it lacks any concrete methods that could help avoid it. As there are no specific steps to take, this validity was kept in mind throughout the research but not explicitly addressed.

8 Conclusions and future work

This paper presents a method for tool selection that aims to facilitate automating project documentation. We conducted two literature reviews as well as several interviews to create this method. The research aimed to answer five research questions that were created to finally answer what methodological support can be provided for facilitating automatic software documentation via tool selection. We aimed to learn more about software documentation and tool selection in the first two questions. While looking for characteristics of software documentation, we found that its application can vary depending on the project. Aside from this, we discovered that the main problem with software documentation was a lack of interest by developers, resulting in sub-par documentation up-to-dateness, structure, and quality. Tool selection is a topic that is taken more seriously overall, and several decision factors were identified for tool selection to answer the second research question. Some of the more relevant factors include pricing, ease of use, and security. The third and fourth questions pertained to constructing a method. Within these questions, two parts of the final method were researched: eliciting documentation needs and using these needs as an element of tool selection. A method was created as a result of these questions, which was validated to answer the final research question as well as complete the research project. The resulting method is called the Software Project Tool Selection Method (SoPro-TSM). It should be able to help in eliciting documentation needs and selecting tools that fit these needs. Future work could include further validation, implementing this methodology, and evaluating the results. Doing this in a way that is beneficial to the participants is likely some time away since actually automating documentation is not yet as advanced as it should be. Other exciting research that could be done to supplement this paper is expanding on the process of creating documentation standards in a generalizable way or creating a database that includes the relevant information about tools for tool selection.

Appendix A Research method PDD

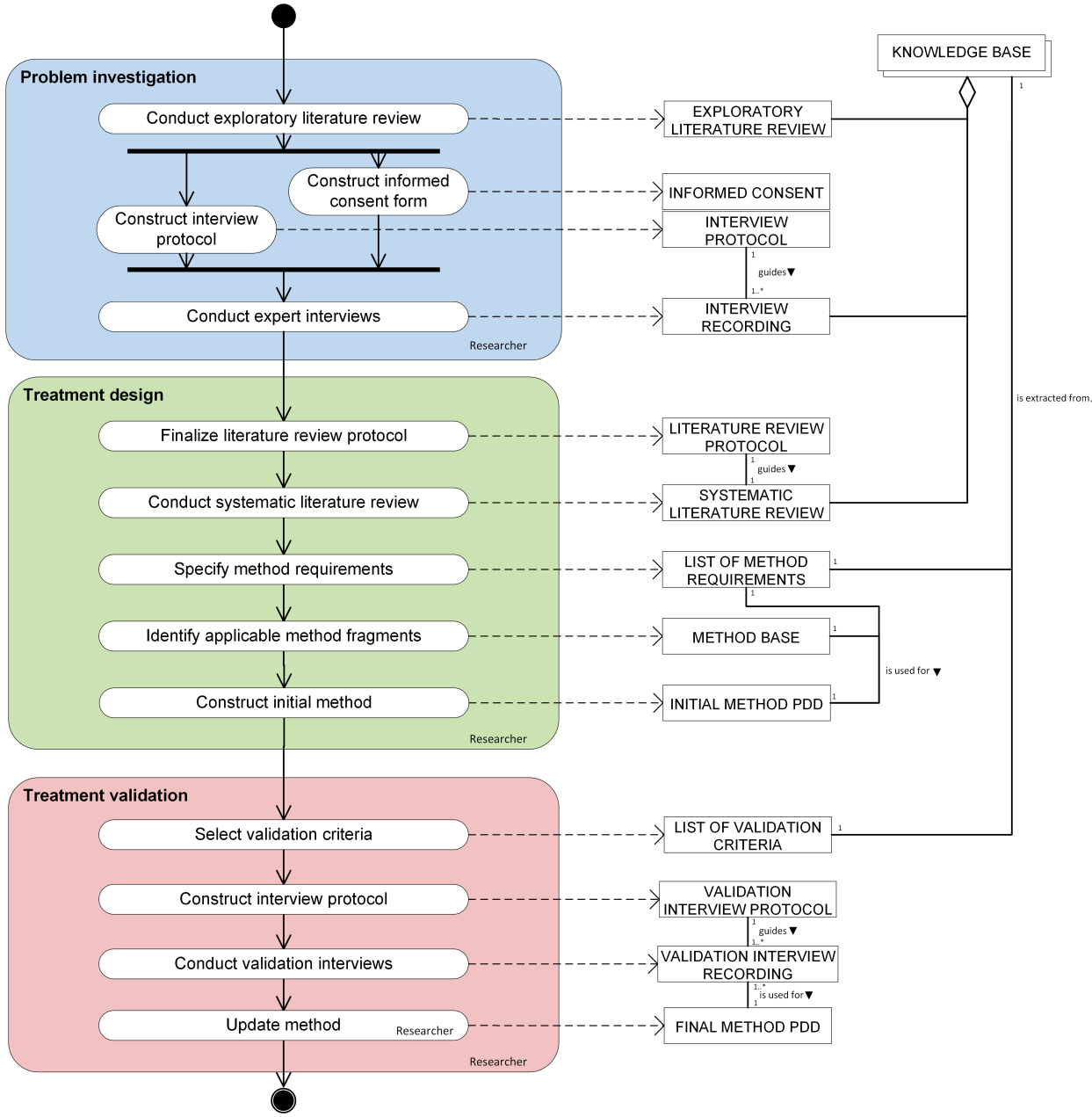

Fig. A1 PDD of the research process

Appendix B Compliance with ethical standards

B.1 Primary interview protocol

My name is Floris Wijbrands and I am currently following the Master's program in Business Informatics at Utrecht University. As part of this program, I am performing a research project at Incentro about creating a method for tool selection that can serve as a foundation for automated software project documentation.

This interview will take approximately 60 minutes and aims to serve two main purposes. At the point of conducting this interview, an exploratory literature review has already been performed to gain a basic understanding of the topic at hand. The first part of this interview will serve to expand on this existing literary knowledge and fill in the gaps. On top of this, during this interview, we will aim to identify stakeholder goals concerning software documentation which will be used to construct requirements for the method that will be created.

For the purpose of alignment, it is important to create a basic understanding of what the term documentation refers to throughout this interview. Sommerville [61] split up the concept of project documentation into the categories product documentation and process documentation. Further details regarding these two categories can be found in Fig. B2. For the duration of this interview, any mention of documentation will include both these categories unless specifically mentioned otherwise.

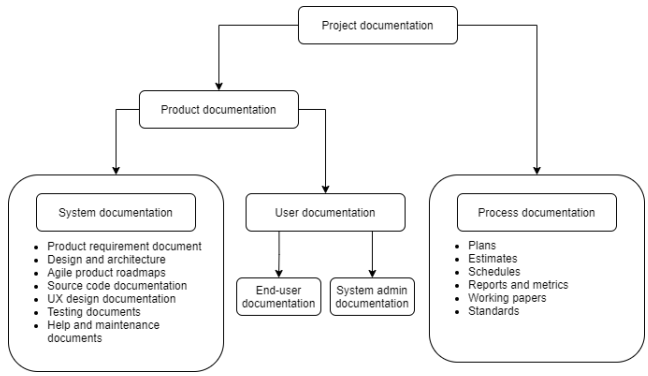

Fig. B2 Summary of the software documentation types [5].

General questions

- What is the name of the department you are in?
- Can you shortly describe what the position you hold in your organization entails?
- How long have you worked in your current position?
- How long have you been working at this organization?
- Can you elaborate on some of the projects that you have worked on throughout the last years?
  - What is the nature of these projects?
  - What is the scale and duration of these projects?

About software documentation

Since the main topic of research for this research is project documentation, I would like to ask you some questions regarding this concept in the context of Incentro.

1. Can you describe to me the process of documentation for projects that you have worked on?
   (a) How do you decide what to document and who is involved in this process?
   (b) Is this a standardized process or is it customized according to each project?

2. How do you feel regarding the quality of documentation as a result of the current process?
   (a) Is it of sufficient detail?
   (b) Is it always up-to-date?
   (c) Can you always find everything you need?
3. Have there been cases where you, in hindsight, found out something was not documented that would have been useful?
   (a) How did you deal with this at the time?
   (b) How have you tried to prevent this in the future?

About automated documentation

The next few questions will be about the concept of automated project documentation. Important to note is that these questions do not address the method that this thesis will produce but the hypothetical continuation of it, which is automated documentation.

1. Although fully automated documentation is currently not available, some efforts have already been made towards automating small parts of the documentation process. Do you have any experience with any of these automated efforts?
   (a) Where have you encountered these efforts?
   (b) If possible, would you like for more of these pieces of automation to be included in your documentation?
2. How do you think fully automated software project documentation would change the way you work?
   (a) How do you think it could benefit your work?
   (b) How do you think it could harm your work?
3. Who do you think would benefit the most from automated documentation?
4. Do you think automated project documentation is something worth pursuing?

About the method

While the exact workings of the final method are not yet defined, two main components have been identified so far. The first part of this method concerns formalizing the information that should be documented during a project. The second part of this method concerns selecting a set of tools to use that can contain this information, thus allowing for potential automated documentation. The following questions concern these two components and some of their requirements.

1. How easy should the method be to execute? Should there be a person specialized in setting up this process or should anyone be able to do it?
2. How often do you alternate between using different tools that essentially do the same thing for different projects?
   (a) What reasons could you think of for using a specific tool during a project?
   (b) Would you be willing to alternate between different tools across projects for the purpose of automated software project documentation?
3. What do you consider being the main goals of the tool selection method as described to you previously?
4. Do you anticipate any risks in implementing such a method that should be taken into account?
5. When would you consider this method to be implemented successfully?

Final words

Thank you for your time participating in this interview. If you would like it I will send you a copy of my thesis upon completion.

B.2 Validation interview protocol

My name is Floris Wijbrands, and I am currently following the Master's program in Business Informatics at Utrecht University. As part of this program, I am performing a research project at Incentro about creating a method for tool selection that can serve as a foundation for automated software project documentation.

This interview will take approximately 60 minutes and aims to serve the purpose of validating a created Process-Deliverable Diagram (PDD). This diagram has been formed using previous interviews and extensive literature review. At the start of this interview, I will explain to you the workings of the proposed PDD, and we will walk through the method together. During the second part of this interview, we will go through several validation criteria as shown in table B1. While doing this, I will ask your opinion on each of the posed statements in the form of a five-point Likert scale. This gives you the options to answer as strongly disagreeing, disagreeing, neither agreeing nor disagreeing, agreeing, or strongly agreeing. If you have questions at any point, please do not hesitate to ask.

Table B1 Final selection from the criteria by Prat et al. [51]

Evaluation criteria	Description
Efficiency	The method could achieve its goal in a real-life situation.
Utility	There is a significant positive difference between the worth of achieving the projected goal and the price paid for executing the method.
Operational feasibility	Management, employees, and other stakeholders will be likely to support the proposed method and incorporate it into their routines.
Absence of side effects (people)	The artifact is free of undesirable impacts on individuals in the long run.
Absence of side effects (organization)	The artifact is free of undesirable impacts on the organization in the long run.
Ease of use	Executing the steps in the artifact is mostly free of effort.
Completeness	The structure of this artifact contains all necessary elements and relationships between elements.
Adaptability	The artifact can easily work in contexts other than those it was originally designed for.
Modifiability	The artifact can easily be changed without introducing defects.

Thank you for your time participating in this interview. If you would like it I will send you a copy of my thesis upon completion.

B.3 Informed consent form

Please read the following consent document carefully before you decide to participate in this study. The researcher will answer any questions before you sign this form.

Title of study:
Communication based on the Principle of Tools are the Message: Methodological Support for Software Project Tool Selection.

Purpose of study:
The purpose of this study is to develop a method for tool selection that can act as a foundation for facilitating automated software project documentation. Interviews will be conducted to gain knowledge on relevant topics as well as to elicit potential method requirements.

Potential risks of participating:
The potential risks of participating in this research are no more than those in everyday life.

Potential benefits of participating:
Contributing to this research in any way could help in providing the first steps toward automated software project documentation. Should this eventual goal be achieved, your company can use it to improve efficiency.

Confidentiality:
All data that is gathered during this research will be recorded. These recordings will only be used for scientific purposes and access to it will be limited to the researcher. Upon completion of the research, the interview recording will be deleted.

Voluntary participation:
Your participation in this study is entirely voluntary. There is no penalty for not participating. You may also refuse to answer any question that is ask

Right to withdraw from the study:
You have the right to withdraw from the study at any time without consequence.

Whom to contact about the study:
If you have any further questions or comments about this study, please contact the researcher via email at f.h.f.wijbrands@students.uu.nl.
If you have an official complaint about the study, you can send an email to the complaints officer at klachtenfunctionaris-fetcsocwet@uu.nl.

Agreement:
I have read the contents described above. I voluntarily agree to participate in this research and I have received a copy of the description.

Participant	*Researcher*
Name:	Name:
Signature:	Signature:
Date:	Date:

Appendix C Coding schemes

Table C2 Coding scheme for the preliminary interviews

Code	Reference
iv-1	(IntervieweeA, personal communication, November 23, 2021)
iv-2	(IntervieweeB, personal communication, November 24, 2021)
iv-3	(IntervieweeC, personal communication, November 24, 2021)
iv-4	(IntervieweeD, personal communication, November 25, 2021)
iv-5	(IntervieweeE, personal communication, November 26, 2021)
iv-6	(IntervieweeF, personal communication, November 29, 2021)
iv-7	(IntervieweeG, personal communication, December 3, 2021)
iv-8	(IntervieweeH, personal communication, December 6, 2021)

Table C3 Coding scheme for the validation interviews

Code	Reference
viv-1	(ValidationExpertA, personal communication, February 12, 2022)
viv-2	(ValidationExpertB, personal communication, February 15, 2022)
viv-3	(ValidationExpertC, personal communication, February 17, 2022)
viv-4	(ValidationExpertD, personal communication, February 20, 2022)

Appendix D Treatment design

D.1 Method evolution PDDs

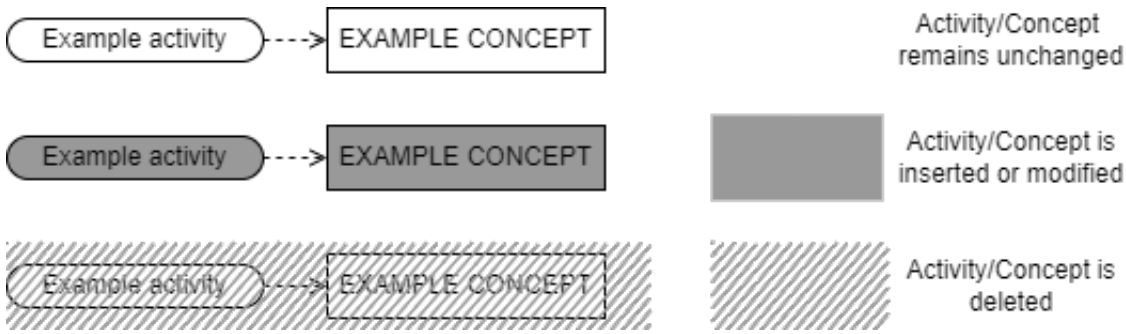

Fig. D3 Method evolution notation

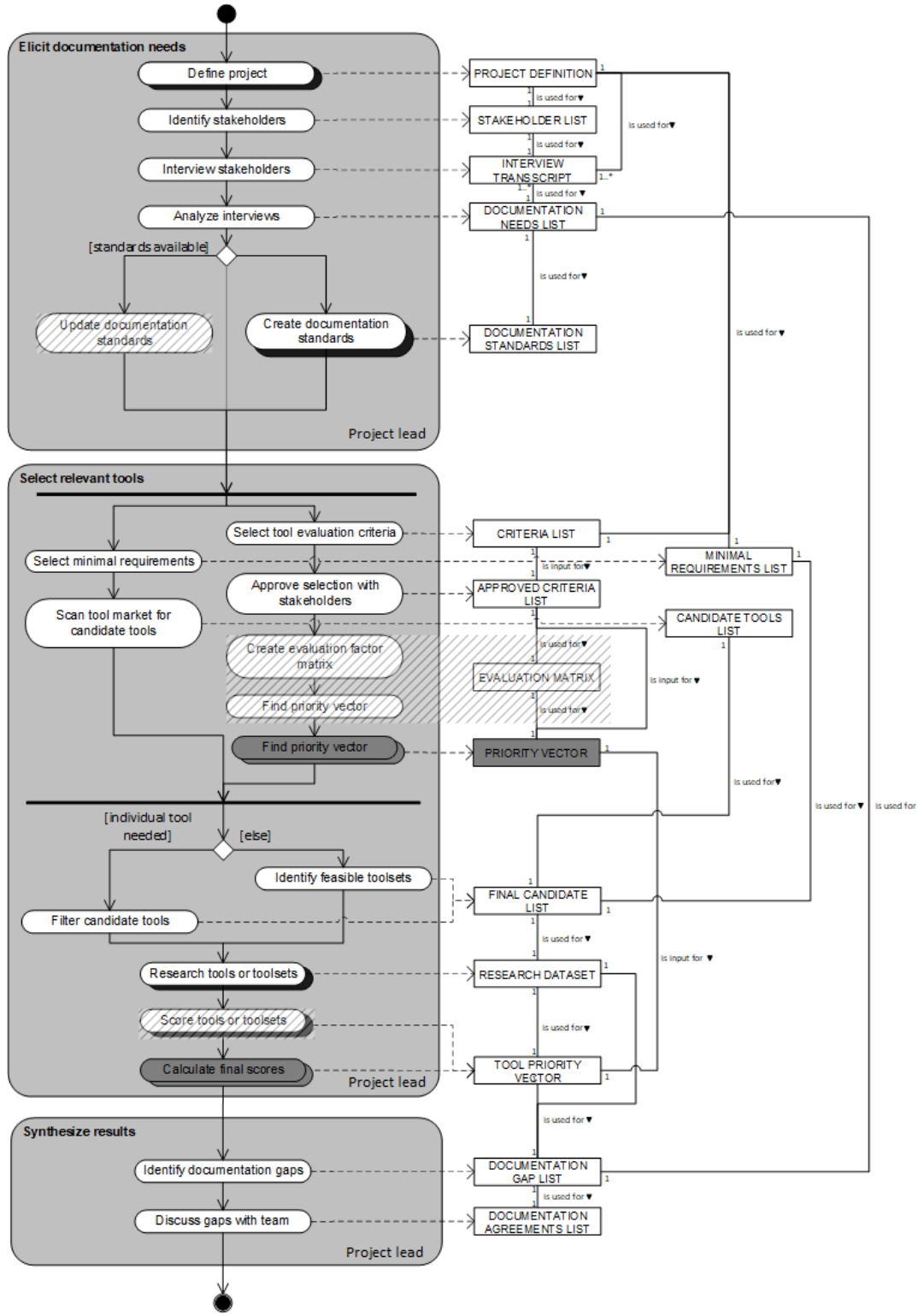

Fig. D4 Method evolution PDD for the main method

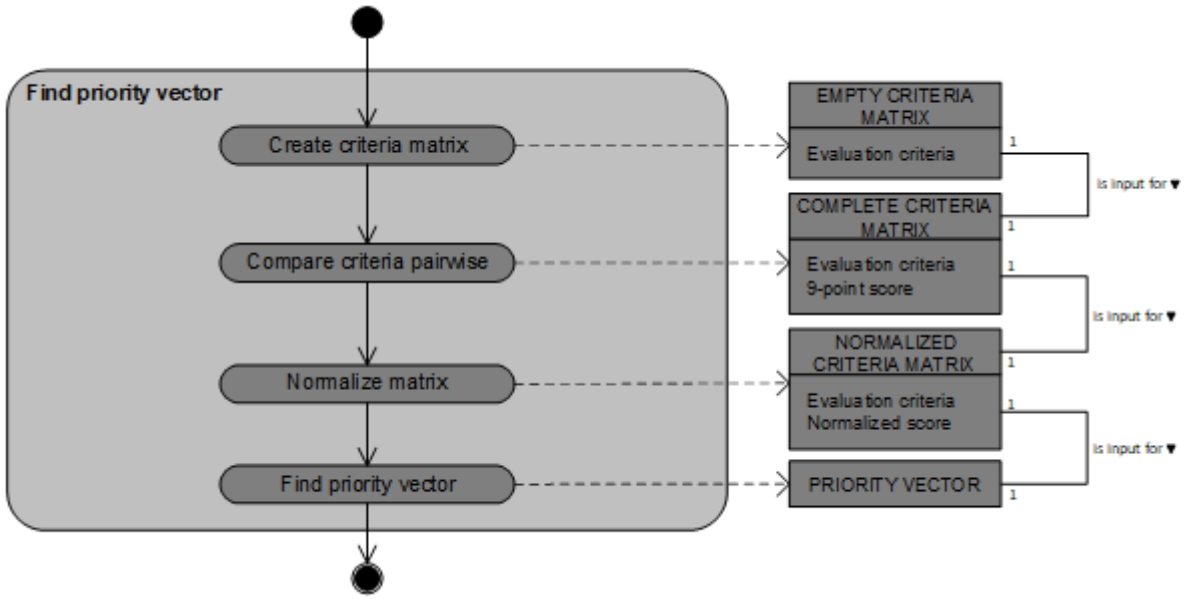

Fig. D5 Method evolution PDD for finding the AHP priority vector

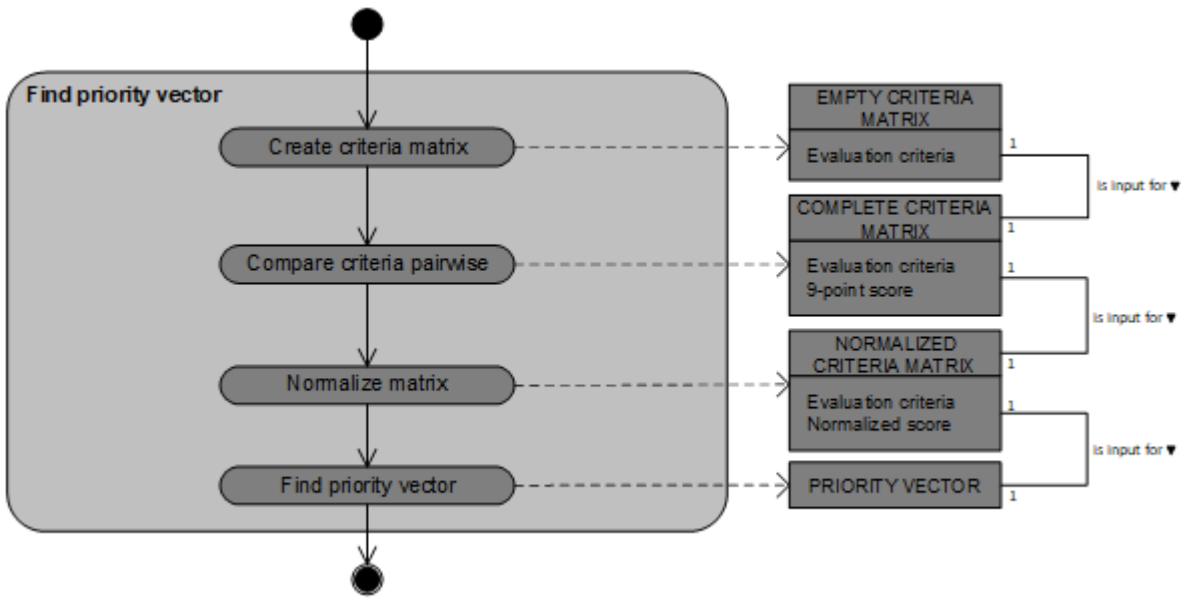

Fig. D6 Method evolution PDD for finding the MAUT tool priority vector

D.2 SoPro-TSM

D.2.1 PDDs

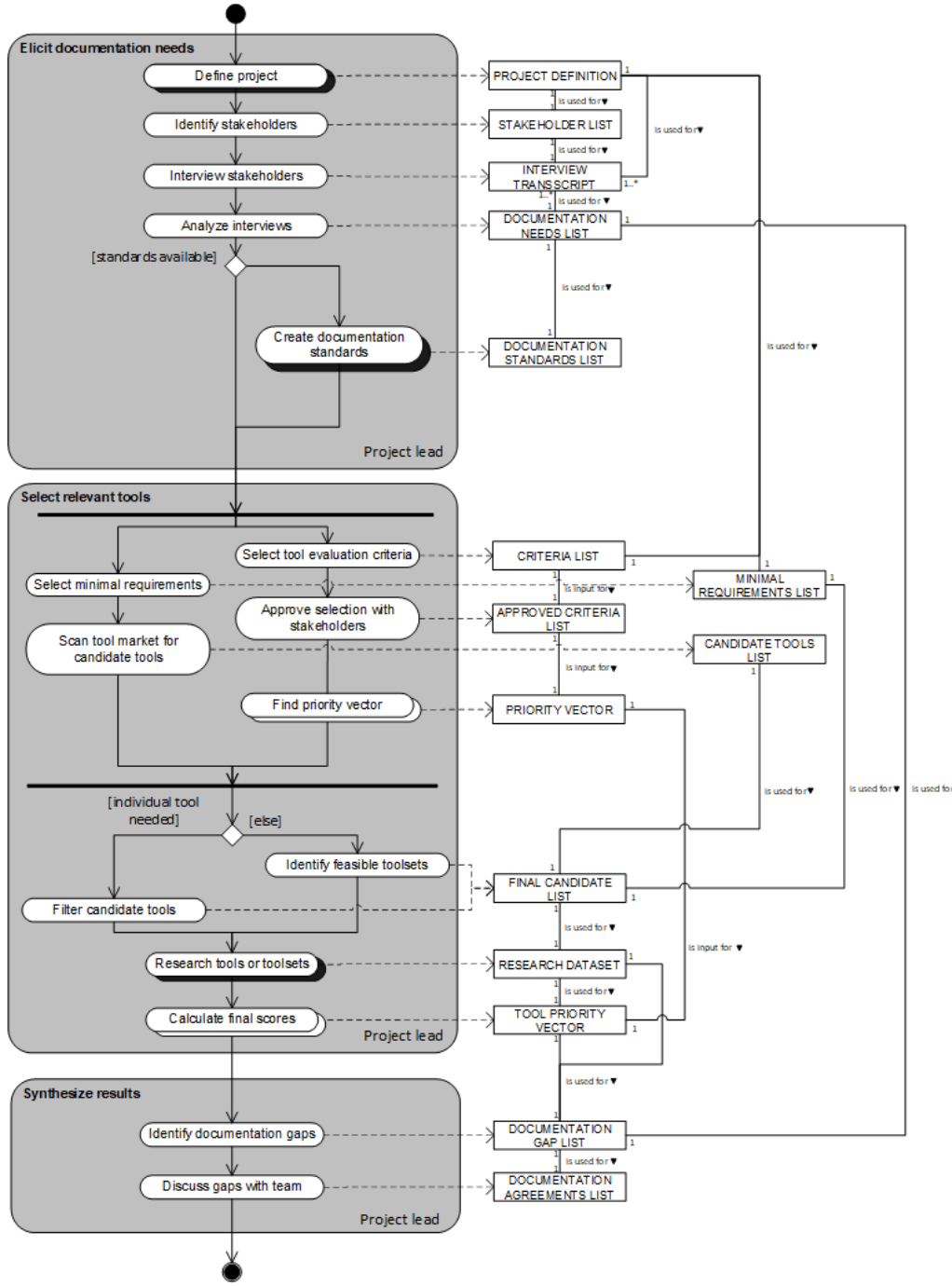

Fig. D7 PDD for the main phases of SoPro-TSM

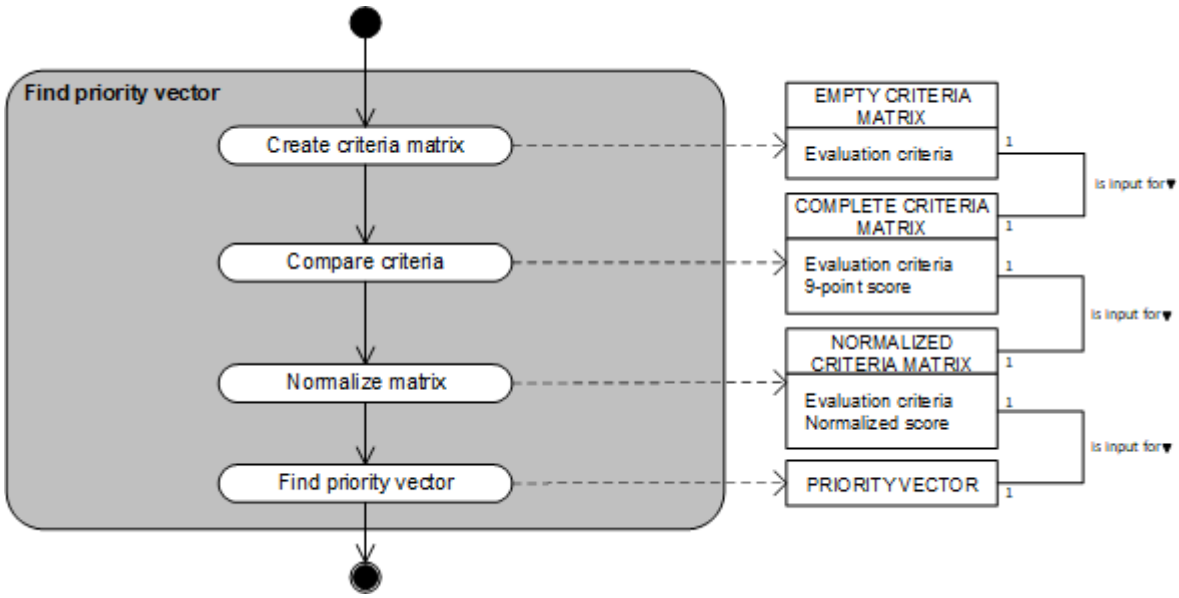

Fig. D8 PDD for finding the AHP priority vector

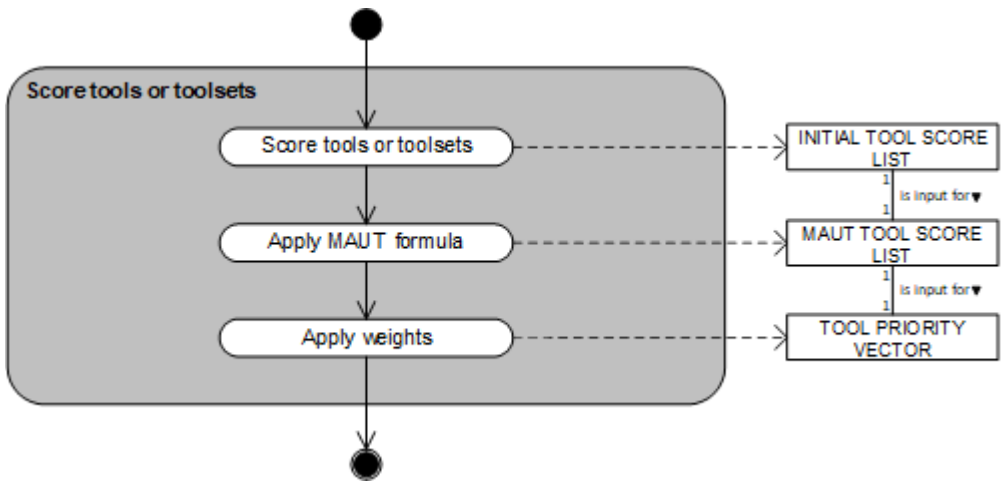

Fig. D9 PDD for finding the MAUT tool priority vector

D.2.2 Activity table

Table D4: Activity table of the SoPro-TSM PDD

Activity	Sub-activity	Description
Define project		At the start of any project the project should be defined. How exactly this is done will depend on the way of working within the organization. The activity results in a PROJECT DEFINITION.
Identify stakeholders		After defining the project, relevant stakeholders need to be identified in order to select participants for any future meetings or interviews. The activity results in a STAKEHOLDER LIST.
Interview stakeholders [**AR2, AR8**]		To start identifying documentation needs the stakeholders should be interviewed. During the views of each stakeholder on project documentation should be elicited so that needs can be extracted. The activity results in an INTERVIEW TRANSCRIPT.
Analyze interviews		Analyzing the views of each stakeholder on documentation allows for the creation of an extensive list of documentation needs. The activity results in a DOCUMENTATION NEEDS LIST.
Create documentation standards [**AR1**]		If no previous documentation standards were available for use within the project, new documentation standards should be created to use in the future. The activity results in a DOCUMENTATION STANDARDS LIST.
Select tool evaluation criteria [**M1**]		Based on the PROJECT DEFINITION, a list of criteria should be created on which the tools that will be selected can be judged. The activity results in a CRITERIA LIST .
Approve selection with stakeholders [**AR3**]		The CRITERIA LIST should be verified by the stakeholders to make sure they relevant and gain an idea of how important they are to the project. The activity results in an APPROVED CRITERIA LIST.
Find priority vector [**M1**]		Finding the priority vector is done through application of the AHP. After following all the steps this activity results in a PRIORITY VECTOR.
	Create criteria matrix	The first activity for execution of the AHP is to create a matrix of the selected criteria to score. The activity results in an EMPTY CRITERIA MATRIX.
	Compare criteria	To identify the importance of criteria, the EMPTY CRITERIA MATRIX has to be filled in. This is done by performing pairwise comparison on each of the criteria and scoring them using a 9-point scale. The activity results in a COMPLETE CRITERIA MATRIX.

Table D4: (continued)

Activity	Sub-activity	Description
	Normalize matrix	The results within the COMPLETE CRITERIA MATRIX need to be normalized in order to be useable for evaluating the criteria. The activity results in a NORMALIZED CRITERIA MATRIX.
	Find priority vector	To complete the AHP, the average score of each evaluation criterion is taken and documented in a PRIORITY VECTOR.
Select minimal requirements **[M2]**		Using the PROJECT DEFINITION, a selection of minimal (or must-have) requirements is made for the tools used within the project. This activity results in a MINIMAL REQUIREMENTS LIST.
Scan tool market for candidate tools **[M5]** **[AR6]**		A selection of available candidate tools is made based on the expertise of the project lead. This activity results in a CANDIDATE TOOLS LIST.
Identify feasible toolsets **[M5]**		If there are no individual tools that meet the selected minimal requirements or there is simply a need for multiple tools, a set of tools that can be considered sufficient will be selected. This activity results in a FINAL CANDIDATE LIST.
Filter candidate tools **[M5]**		Using the MINIMAL REQUIREMENTS LIST as a guide, any tools that are not suitable for the project are filtered out of the candidates. This activity results in a FINAL CANDIDATE LIST.
Research tools or toolsets **[M5]**		To be able to identify the optimal tools for a project the candidate tools should be researched. Depending on the criteria to evaluate, this research might look differently yet it will often come down to searching through product websites, user forums, and actually testing the products. This activity results in a RESEARCH DATASET.
Calculate final scores **[M3]** **[AR6]**		Based on the previously calculated PRIORITY VECTOR and the RESEARCH DATASET the final scores for each candidate tool are calculated using MAUT. This activity results in a TOOL PRIORITY VECTOR.
	Score tools or toolsets	The tools or toolsets that are available for selection are given an initial score on each criterion. The activity results in a INITIAL TOOL SCORE LIST.
	Apply MAUT formula	The MAUT formula is applied to the INITIAL TOOL SCORE LIST in order to find the MAUT TOOL SCORE LIST. Given a score x on criterion y, The formula that is used for this method is $\frac{x-worsty}{besty-worsty}$ where wosty and besty signify the worst and best scores that were given on y out of all the rated tools.
	Apply weights	The weights of the criteria are applied to the MAUT TOOL SCORE LIST to obtain the WEIGHTED SCORE LIST.

Table D4: (continued)

Activity	Sub-activity	Description
Identify documentation gaps		After selecting a set of tools based on the best combination of attributes, the documentation capabilities need to be analyzed. This is done based on the RESEARCH DATASET and results in a DOCUMENTATION GAP LIST.
Discuss gaps [**AR4, AR5**]		Using the DOCUMENTATION GAP LIST as a foundation, the development team should meet and discuss what will need to be documented by hand during the project, what can be automated and how this will be approached. This activity results in a DOCUMENTATION AGREEMENT LIST.
Update documentation standards [**AR1, AR7**]		After the project has run its course the documentation should be reflected on and documentation standards should be updated if necessary. These updates are represented in the DOCUMENTATION STANDARDS LIST.

D.2.3 Concept table

Table D5 Concept table of the SoPro-TSM PDD

Concept	Sub concept	Description
PROJECT DEFINITION		A document containing the parameters and details of a project.
STAKEHOLDER LIST		A list of stakeholders that either participate in creating documentation or use it.
INTERVIEW TRANSCRIPT		A transcript of the interviews conducted with relevant stakeholders.
DOCUMENTATION NEEDS LIST		A list of documentation needs that should provide complete oversight of what should be present within the documentation.
DOCUMENTATION STANDARDS LIST		A list of documentation standards that should be abided by in the current or in future projects.
CRITERIA LIST		A list of the selected tool criteria that could be used to evaluate the candidate tools.
APPROVED CRITERIA LIST		A complete list of evaluation criteria that was approved by the relevant stakeholder and will be used for the final evaluation.
PRIORITY VECTOR		A vector containing a list of evaluation criteria and their respective weights.
	EMPTY CRITERIA MATRIX	A matrix containing a matchup of all the criteria that were selected previously.
	COMPLETE CRITERIA MATRIX	A matrix containing a matchup of all the criteria that were selected previously. This version includes the scores of each matchup on a 9-point scale.
	NORMALIZED CRITERIA MATRIX	A matrix containing a matchup of all the criteria that were selected previously. This version includes a normalized version of the scores of each matchup on a 9-point scale.
MINIMAL REQUIREMENTS LIST		A list of minimal, or must-have, requirements that the candidates have to fulfill in order to be considered for the final selection.
CANDIDATE TOOLS LIST		A list of candidate tools for the final selection. This list has not yet been subjected to the list of minimal requirements.
FINAL CANDIDATE LIST		A final list of candidate tools or toolsets that all fulfill the minimal requirements and are ready to be evaluated.
RESEARCH DATASET		A complete dataset resulting from the research performed on the candidate tools. This data includes information on each of the evaluation criteria including their suitability for automated documentation.
TOOL PRIORITY VECTOR		A final vector of the candidate tools or toolsets including their final weighted score. This vector is what is used to make a final decision on the selection.
	INITIAL TOOL SCORE LIST	A list that includes the scores of each tool for all individual evaluation criteria.
	MAUT TOOL SCORE LIST	A list of scores that have been subjected to the MAUT formula.
DOCUMENTATION GAP LIST		A list of all the gaps that will be present in the automated documentation because of the tool selection that was chosen.
DOCUMENTATION AGREEMENT LIST		A list of all the agreements made on how to deal with automated documentation. This includes a plan for reviews, dealing with gaps, and responsibilities.

Appendix E Validation scores

Table E6 Score for each validation criterion per interviewee

Criterion	viv-1	viv-2	viv3	viv4
Efficiency	3	3	4	4
Utility	3	3	2	3
Operational feasibility	2	2	2	2
Absence of side effects (people)	3	3	3	3
Absence of side effects (organization)	3	3	3	3
Ease of use	3	3	3	3
Completeness	3	3	3	3
Adaptability	4	3	4	4
Modifiability	3	2	3	3

References

[1] Aghajani, E., Nagy, C., Vega-Márquez, O.L., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M.: Software documentation issues unveiled. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 1199–1210 (2019). IEEE

[2] Aghajani, E., Nagy, C., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., Shepherd, D.C.: Software documentation: the practitioners' perspective. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 590–601 (2020). IEEE

[3] Ahmad, N., Laplante, P.A.: Software project management tools: making a practical decision using ahp. In: 2006 30th Annual IEEE/-NASA Software Engineering Workshop, pp. 76–84 (2006). IEEE

[4] Alomar, N., Almobarak, N., Alkoblan, S., Alhozaimy, S., Alharbi, S.: Usability engineering of agile software project management tools. In: International Conference of Design, User Experience, and Usability, pp. 197–208 (2016). Springer

[5] Altexoft: Technical Documentation in Software Development: Types, Best Practices, and Tools. https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools

[6] Ambler, S.: Effective practices for extreme programming and the unified process. Ist. Ed. John Wiley & Sons, Inc (2002)

[7] Azizyan, G., Magarian, M.K., Kajko-Matsson, M.: Survey of agile tool usage and needs. In: 2011 Agile Conference, pp. 29–38 (2011). IEEE

[8] Bosilj-Vukšić, V.: The method of business process oriented tool selection in information systems development projects. Zagreb International Review of Economics & Business **9**(2), 135–153 (2006)

[9] Chomal, V.S., Saini, J.R.: Software project documentation-an essence of software development. International Journal of Advanced Networking and Applications **6**(6), 2563 (2015)

[10] Cockburn, A.: Agile Software Development. Addison Wesley, Boston, MA (2001)

[11] Cortés-Coy, L.F., Linares-Vásquez, M., Aponte, J., Poshyvanyk, D.: On automatically generating commit messages via summarization of source code changes. In: 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation, pp. 275–284 (2014). IEEE

[12] Curtis, B., Krasner, H., Iscoe, N.: A field study of the software design process for large systems. Communications of the ACM **31**(11), 1268–1287 (1988)

[13] Dagenais, B., Robillard, M.P.: Creating and evolving developer documentation: understanding the decisions of open source contributors. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 127–136 (2010)

[14] Das, S., Lutters, W.G., Seaman, C.B.: Understanding documentation value in software maintenance. In: Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology, p. 2 (2007)

[15] de Souza, S.C.B., Anquetil, N., de Oliveira, K.M.: A study of the documentation essential to software maintenance. In: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information, pp. 68–75 (2005)

[16] Díaz-Pace, J.A., Villavicencio, C., Schiaffino, S., Nicoletti, M., Vázquez, H.: Producing just enough documentation: An optimization approach applied to the software architecture domain. Journal on Data Semantics **5**(1), 37–53 (2016)

[17] Farshidi, S., Jansen, S., Fortuin, S.: Model-driven development platform selection: four industry case studies. Software and Systems Modeling **20**(5), 1525–1551 (2021)

[18] 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). IEEE

[19] Fontanet Losquiño, D., Urdell, T.: Why do developers struggle with documentation while excelling at programming. B.S. thesis, Universitat Politècnica de Catalunya (2014)

[20] Forward, A., Lethbridge, T.C.: The relevance of software documentation, tools and technologies: a survey. In: Proceedings of the 2002 ACM Symposium on Document Engineering, pp. 26–33 (2002)

[21] Fraser, G., Staats, M., McMinn, P., Arcuri, A., Padberg, F.: Does automated unit test generation really help software testers? a controlled empirical study. ACM Transactions on Software Engineering and Methodology (TOSEM) **24**(4), 1–49 (2015)

[22] Garousi, G., Garousi, V., Moussavi, M., Ruhe, G., Smith, B.: Evaluating usage and quality of technical software documentation: an empirical study. In: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, pp. 24–35 (2013)

[23] Garousi, G., Garousi-Yusifoğlu, V., Ruhe, G., Zhi, J., Moussavi, M., Smith, B.: Usage and usefulness of technical software documentation: An industrial case study. Information and Software Technology **57**, 664–682 (2015)

[24] Gehanno, J.-F., Rollin, L., Darmoni, S.: Is the coverage of google scholar enough to be used alone for systematic reviews. BMC medical informatics and decision making **13**(1), 1–5 (2013)

[25] Gotel, O., Mäder, P.: Acquiring tool support for traceability. In: Software and Systems Traceability, pp. 43–68 (2012). Springer

[26] Guest, G., Namey, E., Taylor, J., Eley, N., McKenna, K.: Comparing focus groups and individual interviews: findings from a randomized study. International Journal of Social Research Methodology **20**(6), 693–708 (2017)

[27] Hadar, I., Sherman, S., Hadar, E., Harrison, J.J.: Less is more: Architecture documentation for agile development. In: 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 121–124 (2013). IEEE

[28] Hager, J.A.: Software cost reduction methods in practice: A post-mortem analysis. Journal of systems and software **14**(2), 67–77 (1991)

[29] Haiduc, S., Aponte, J., Moreno, L., Marcus, A.: On the use of automated text summarization techniques for summarizing source code. In: 2010 17th Working Conference on Reverse Engineering, pp. 35–44 (2010). IEEE

[30] Heale, R., Forbes, D.: Understanding triangulation in research. Evidence-based nursing **16**(4), 98–98 (2013)

[31] Hu, X., Li, G., Xia, X., Lo, D., Jin, Z.: Deep code comment generation. In: 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), pp. 200–210 (2018). IEEE

[32] Janicki, R., Parnas, D.L., Zucker, J.: Tabular representations in relational documents. In: Relational Methods in Computer Science, pp. 184–196 (1997). Springer

[33] Jayasuriya, D.B., Perera, I.: Ontology based software design documentation for design reasoning. In: 2019 Moratuwa Engineering Research Conference (MERCon), pp. 710–715 (2019). IEEE

[34] Jones, C.: Software project management practices: Failure versus success. CrossTalk: The Journal of Defense Software Engineering

17(10), 5–9 (2004)

[35] Li, B., Vendome, C., Linares-Vásquez, M., Poshyvanyk, D., Kraft, N.A.: Automatically documenting unit test cases. In: 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), pp. 341–352 (2016). IEEE

[36] Lotufo, R., Malik, Z., Czarnecki, K.: Modelling the 'hurried'bug report reading process to summarize bug reports. Empirical Software Engineering **20**(2), 516–548 (2015)

[37] Madsen, J., Munck, A.: A systematic and practical method for selecting systems engineering tools. In: 2017 Annual IEEE International Systems Conference (SysCon), pp. 1–8 (2017). IEEE

[38] Mahmood, S., Khan, A.: An industrial study on the importance of software component documentation: A system integrators perspective. Information Processing Letters **111**(12), 583–590 (2011)

[39] Mani, S., Catherine, R., Sinha, V.S., Dubey, A.: Ausum: approach for unsupervised bug report summarization. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pp. 1–11 (2012)

[40] Maxwell, J.: Understanding and validity in qualitative research. Harvard educational review **62**(3), 279–301 (1992)

[41] McBurney, P.W., McMillan, C.: Automatic source code summarization of context for java methods. IEEE Transactions on Software Engineering **42**(2), 103–119 (2015)

[42] McBurney, P.W., Liu, C., McMillan, C., Weninger, T.: Improving topic model source code summarization. In: Proceedings of the 22nd International Conference on Program Comprehension, pp. 291–294 (2014)

[43] McLuhan, M., Fiore, Q.: The medium is the message. New York **123**, 126–128 (1967)

[44] Moreno, L., Aponte, J., Sridhara, G., Marcus, A., Pollock, L., Vijay-Shanker, K.: Automatic generation of natural language summaries for java classes. In: 2013 21st International Conference on Program Comprehension (ICPC), pp. 23–32 (2013). IEEE

[45] Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., Marcus, A., Canfora, G.: Automatic generation of release notes. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 484–495 (2014)

[46] Niazi, M., Mahmood, S., Alshayeb, M., Hroub, A.: Empirical investigation of the challenges of the existing tools used in global software development projects. IET Software **9**(5), 135–143 (2015)

[47] Okoli, C.: A guide to conducting a standalone systematic literature review. Communications of the Association for Information Systems **37**(1), 43 (2015)

[48] Panichella, S., Panichella, A., Beller, M., Zaidman, A., Gall, H.C.: The impact of test case summaries on bug fixing performance: An empirical investigation. In: Proceedings of the 38th International Conference on Software Engineering, pp. 547–558 (2016)

[49] Pilar, M., Simmonds, J., Astudillo, H.: Semi-automated tool recommender for software development processes. Electronic Notes in Theoretical Computer Science **302**, 95–109 (2014)

[50] Poston, R.M.: When does more documentation mean less work. IEEE Software, 98–99 (1984)

[51] Prat, N., Comyn-Wattiau, I., Akoka, J.: A taxonomy of evaluation methods for information systems artifacts. Journal of Management Information Systems **32**(3), 229–267 (2015)

[52] Prause, C.R., Durdik, Z.: Architectural design and documentation: Waste in agile development? In: 2012 International Conference on Software and System Process (icssp), pp. 130–134 (2012). IEEE

[53] Priestley, M., Utt, M.H.: A unified process for software and documentation development. In: 18th Annual Conference on Computer Documentation. Ipcc Sigdoc 2000. Technology and Teamwork. Proceedings. IEEE Professional Communication Society International Professional Communication Conference An, pp. 221–238 (2000). IEEE

[54] Rastkar, S., Murphy, G.C., Murray, G.: Automatic summarization of bug reports. IEEE Transactions on Software Engineering **40**(4), 366–380 (2014)

[55] Raulamo-Jurvanen, P., Kakkonen, K., M¨antyl¨a, M.: Using surveys and web-scraping to select tools for software testing consultancy. In: International Conference on Product-Focused Software Process Improvement, pp. 285–300 (2016). Springer

[56] Rivas, L., Pérez, M., Mendoza, L.E., Grimán, A.C.: Tools selection criteria in software-developing small and medium enterprises. Journal of Computer Science & Technology **10** (2010)

[57] Rodeghero, P., Jiang, S., Armaly, A., McMillan, C.: Detecting user story information in developer-client conversations to generate extractive summaries. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 49–59 (2017). IEEE

[58] Rodic, B., Marinova, G., Chikov, O.: Algorithms and decision making methods for filter design tool selection for a given specification in online-cadcom platform. In: Proceedings of the Twenty-Sixth International Electrotechnical and Computer Science Conference ERK'2017, Slovenian IEEE Section, pp. 247–251 (2017)

[59] Sauer, T.: Using design rationales for agile documentation. In: WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003., pp. 326–331 (2003). IEEE

[60] Scherer, R.W., Saldanha, I.J.: How should systematic reviewers handle conference abstracts? a view from the trenches. Systematic reviews **8**(1), 1–6 (2019)

[61] Sommerville, I.: Software documentation. Software Engineering, Volume 2: The Supporting Processes, 143–154 (2001)

[62] Stettina, C.J., Heijstek, W.: Necessary and neglected? an empirical study of internal documentation in agile software development teams. In: Proceedings of the 29th ACM International Conference on Design of Communication, pp. 159–166 (2011)

[63] Stokes, D., Bergin, R.: Methodology or "methodolatry"? an evaluation of focus groups and depth interviews. Qualitative market research: An international Journal (2006)

[64] Taheri, M., Sadjadi, S.M.: A feature-based tool-selection classification for agile software development. In: SEKE, pp. 700–704 (2015)

[65] Theunissen, T., Van Heesch, U.: Specification in continuous software development. In: Proceedings of the 22nd European Conference on Pattern Languages of Programs, pp. 1–19 (2017)

[66] Theunissen, T., Hoppenbrouwers, S., Overbeek, S.: In continuous software development, tools are the message for documentation. In: Proceedings of the 23rd International Conference on Enterprise Information Systems, SCITEPRESS-Science and Technology Publications (2021)

[67] Thurmond, V.A.: The point of triangulation. Journal of nursing scholarship **33**(3), 253–258 (2001)

[68] Treude, C., Robillard, M.P., Dagenais, B.: Extracting development tasks to navigate software documentation. IEEE Transactions on Software Engineering **41**(6), 565–581 (2014)

[69] Uddin, G., Robillard, M.P.: How api documentation fails. Ieee software **32**(4), 68–75 (2015)

[70] van de Weerd, I., Brinkkemper, S.: Meta-modeling for situational analysis and design methods. In: Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, pp. 35–54 (2009). IGI Global

[71] van de Weerd, I., Brinkkemper, S., Souer, J., Versendaal, J.: A situational implementation method for web-based content management system-applications: method engineering and validation in practice. Software process: improvement and practice **11**(5), 521–538 (2006)

[72] Venable, J., Pries-Heje, J., Baskerville, R.: A comprehensive framework for evaluation in design science research. In: International Conference on Design Science Research in Information Systems, pp. 423–438 (2012). Springer

[73] Verma, V., Dhawan, S.: Methodology for selection of a data mining tool. International Journal of Software & Hardware Research in Engineering **2**(5), 189–192 (2014)

[74] Voropaev, V., Gelrud, Y., Klimenko, O.: Who manages what? project management for different stakeholders. Procedia-Social and Behavioral Sciences **226**, 478–485 (2016)

[75] Waits, T., Yankel, J.: Continuous system and user documentation integration. In: 2014 IEEE International Professional Communication Conference (IPCC), pp. 1–5 (2014). IEEE

[76] Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering, (2014). Springer

[77] Yin, R.K.: Case Study Research and Applications: Design and Methods, (2018). Sage

[78] Zhi, J., Garousi-Yusifoğlu, V., Sun, B., Garousi, G., Shahnewaz, S., Ruhe, G.: Cost, benefits and quality of software development documentation: A systematic mapping. Journal of Systems and Software **99**, 175–198 (2015)