Evaluation of Approaches for Documentation in Continuous Software Development

Theo Theunissen[1][a], Stijn Hoppenbrouwers[1,2][b] and Sietse Overbeek[3][c]
[1]Department of ICT, HAN University of Applied Sciences, Arnhem, The Netherlands
[2]Radboud University, Institute for Computing and Information Sciences, Nijmegen, The Netherlands
[3]Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands

Keywords: Artifacts, Continuous Software Development, Documentation, Executable Documentation, Just Enough Upfront.

Abstract: With the adoption of values, principles, practices, tools and processes from Agile, Lean, and DevOps, knowledge preservation has become a serious issue because documentation is largely left out. We identify two questions that are relevant for knowledge acquisition and distribution concerning design decisions, rationales, or reasons for code change. The first concerns which knowledge is required upfront to start a project. The second question concerns continuation after initial development and addresses which knowledge is required by those who deploy, use or maintain a software product. We evaluate two relevant approaches for alleviating the issues, which are 'Just enough Upfront' and 'Executable Documentation' with a total of 25 related artifacts. For the evaluation, we conducted a case study supported by a literature review, organizational and project metrics, and a survey. We looked into closed source-code and closed classified source-code. We found two conclusive remarks. First, git commit messages typically contain what has been changed but not why source-code has been changed. Design decisions, rationale, or reasons for code change should be saved as close as possible to the source-code with Git Pull Requests. Second, knowledge about a software product is not only written down in artifacts but is also a social construction between team members.

1 INTRODUCTION

This study concerns the evaluation of novel approaches to documentation in Continuous Software Development (CSD). CSD is an umbrella term that covers values, principles, practices, tools and processes from Agile, Lean, and DevOps (Theunissen, van Heesch, & Avgeriou, 2022). Characteristics of CSD are that information about a software product is distributed across all tools which hold code and other (non)executable artifacts that stakeholders require to start an iteration or keep continuing an iteration. Another characteristic is that knowledge is loose, informal, and communicated in meetings such as daily stand-ups, leading to a risk of knowledge evaporation. In a previous study (Theunissen, Hoppenbrouwers, et al., 2022), we looked into information about software products, primarily to Git commit messages from open source repositories. In this study, we evaluate two approaches: 'Just enough Upfront' and 'Executable Documentation' concerning the characteristics of CSD. The first approach concerns 'just enough knowledge about stakeholder concerns, requirements, and specifications to start development'. The distribution of knowledge takes place through the delivery of a design afterward, including design decisions. This approach is typically used for fast Time-to-Market (TTM) situations. The second approach covers more mature projects where requirements and specifications are used at the start of Test Driven Development (TDD) and Behavior Driven Development (BDD).

To evaluate the approaches with artifacts, we studied practical usages in the industry of artifacts with closed source code. 'Closed source code' refers to software that is not publicly available, including information about the software. Moreover, some code bases are also classified and are not even available to everyone within the organization studied. Special clearances were required, or developers had to be disconnected from the internet. Reasons were (national)

[a] https://orcid.org/0000-0003-0681-8666
[b] https://orcid.org/0000-0002-1137-2999
[c] https://orcid.org/0000-0003-3975-200X

Theunissen, T., Hoppenbrouwers, S. and Overbeek, S.
Evaluation of Approaches for Documentation in Continuous Software Development.
DOI: 10.5220/0011846200003464
In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2023)*, pages 404-411
ISBN: 978-989-758-647-7; ISSN: 2184-4895

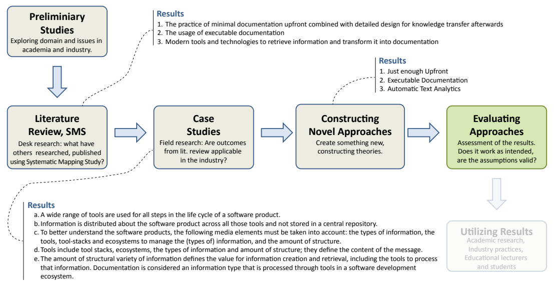

Preliminiary Studies
Exploring domain and issues in academia and industry.
Results
1. The practice of minimal documentation upfront combined with detailed design for knowledge transfer afterwards
2. The usage of executable documentation
3. Modern tools and technologies to retrieve information and transform it into documentation
Results
1. Just enough Upfront
2. Executable Documentation
3. Automatic Text Analytics
Literature Review, SMS
Desk research: what have others researched, published using Systematic Mapping Study?
Case Studies
Field research: Are outcomes from lit. review applicable in the industry?
Constructing Novel Approaches
Create something new, constructing theories.
Evaluating Approaches
Assessment of the results. Does it work as intended, are the assumptions valid?
Results
a. A wide range of tools are used for all steps in the life cycle of a software product.
b. Information is distributed about the software product across all those tools and not stored in a central repository.
c. To better understand the software products, the following media elements must be taken into account: the types of information, the tools, tool-stacks and ecosystems to manage the (types of) information, and the amount of structure.
d. Tools include tool stacks, ecosystems, the types of information and amount of structure; they define the content of the message.
e. The amount of structural variety of information defines the value for information creation and retrieval, including the tools to process that information. Documentation is considered an information type that is processed through tools in a software development ecosystem.
Utilizing Results
Academic research, Industry practices, Educational lecturers and students

Figure 1: Phases in Research Project. This study concerns the evaluation, depicted in green.

security, privacy, or protecting (national) infrastructure against threats.

1.1 Research Project and Related Studies

This study concerns the evaluation of approaches to knowledge preservation in a research project that started with preliminary studies to explore the IT-engineering domain and knowledge preservation problems (Theunissen & van Heesch, 2016). Following the preliminary study, a systematic mapping study was performed to find out what others have found already (Theunissen, van Heesch, & Avgeriou, 2022). Following the literature review, the case studies in the industry set out to validate the mapping studies and discover new issues, including the distribution of information about software products across a software development ecosystem (Theunissen et al., 2021). In the construction phase, novel approaches such as 'Just enough Upfront' and 'Executable Documentation', including artifacts concerned, were described to cover the issues (Theunissen, Hoppenbrouwers, et al., 2022). See Figure 1 for an overview.

In previous research, three novel approaches were constructed in response to the findings (Theunissen, Hoppenbrouwers, et al., 2022), of which two are evaluated and discussed in the current paper. These approaches are:

Just Enough Upfront. This approach leaves out a big upfront design and encourages starting development as soon as possible. Agility in adapting to change is motivated by the progressive insight that leads to modified requirements and specifications, as inherent to CSD. There are no obstacles to applying this approach. Characteristics of the approach are that it is best used for exploratory projects. Exploration is typical for Technology Readiness Levels (TRL) ≤ 3 with Proof of Concept (PoC), but its exploration is also applicable to more mature phases such as prototyping, doing pilots, and taking software into production. The approach fits Agile and Lean practices. There are sixteen relevant artifacts, of which whiteboard sketches and a Interface Definition Language (IDL) plan are most relevant upfront. After delivery, design decisions and accountability are most applicable.

Executable Documentation. This concerns any artifact related to a software product that is executable and relatively easy to read by non-technical persons. Conditions are requirements (what), and specifications (how) that must be well-defined upfront. Characteristic of this approach is the use of pipelines in Continuous Integration/Continuous Deployment (CI/CD) (Theunissen, Hoppenbrouwers, et al., 2022). The maturity level in TRL is $4 \leq 9$. It is suitable for DevOps and accommodates fast TTM. Of the nine relevant artifacts, frameworks, templates, libraries, and Application Programming Interface (API)s are the most relevant. Typical processes are TDD and BDD. Developers and operators meet over infrastructure-as-code where concerns match, for instance, for fast TTM.

2 RESEARCH DESIGN

2.1 Research Questions

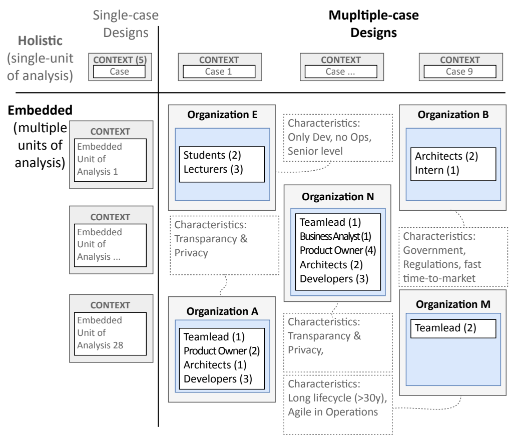
RQ1. What are the necessary and sufficient conditions for evaluating our novel approaches of documentation in CSD?
This question concerns the research method to achieve rigor by demonstrating transparency and repeatability.

RQ2. For both approaches, which of the artifacts were used, are missing, or need adjustments of conditions and characteristics?
This question examines in detail the claims for merits.

RQ3. What are defining characteristics of obstacles that need to be resolved for implementing the novel approaches in the industrial and educational contexts?
This question is a preparation for utilizing the approaches in actual usage.

2.2 Research Methods

In previous research, we used Design Science (DS) to construct and validate the novel approaches (Wieringa, 2014). We will continue with DS for evaluating the approaches using case studies as proposed by Wieringa (2014). Case studies are used for data collection, following Yin (2016); see Figure 2. According to Wieringa (2014, p. 31), validation con-



Single-case Designs
Mupltiple-case Designs
Holistic (single-unit of analysis)
CONTEXT (5) Case
CONTEXT Case 1
CONTEXT Case ...
CONTEXT Case 9
Embedded (multiple units of analysis)
CONTEXT Embedded Unit of Analysis 1
CONTEXT Embedded Unit of Analysis ...
CONTEXT Embedded Unit of Analysis 28
Organization E
Students (2)
Lecturers (3)
Characteristics: Only Dev, no Ops, Senior level
Organization B
Architects (2)
Intern (1)
Characteristics: Transparancy & Privacy
Organization N
Teamlead (1)
Business Analyst (1)
Product Owner (4)
Architects (2)
Developers (3)
Characteristics: Government, Regulations, fast time-to-market
Organization A
Teamlead (1)
Product Owner (2)
Architects (1)
Developers (3)
Characteristics: Transparancy & Privacy,
Organization M
Teamlead (2)
Characteristics: Long lifecycle (>30y), Agile in Operations

Figure 2: Units of Analysis in Case Studies.

cerns the assessment of potential usage of (in our case) novel approaches and their construction. Evaluation is the assessment of novel approaches, justified by assessing the actual outcomes compared with the intended outcomes. Assessment methods for validation are formative and summative judgments (Gonzalez & Sol, 2012; Lee & Hubona, 2009). Below, definitions for the judgments are given.

1. Theory evaluation includes formative validity and summative validity (Gonzalez & Sol, 2012).
2. Formative validity refers to the process of building a new theory or approach. A key characteristic is transparency (Lee & Hubona, 2009).
3. Summative validity refers to the sum of the results of the theories or approaches. It is achieved through artifact evaluation (Lee & Hubona, 2009).

3 DATA COLLECTION

We conducted nine case studies in five organizations with 28 units of analysis. One organization is commercial, one organization is educational, and three others are governmental. From five organizations, we consulted the participants with semi-structured interviews, studied non-executable artifacts such as documentation in tools (including Git commit messages), and reviewed source code. In Figure 2, the case studies are presented. In selecting the organizations, we targeted two specific approaches, i.e., exclusively 'Just enough Upfront' or 'Executable Documentation'.

3.1 A Myriad of Tools That Contain Information

The motivation for collecting this data is that it contributes to answering RQ2 and RQ3. A second motivation for collecting this data is that in previous research (Theunissen et al., 2021), we found that modern software development ecosystems include many tools that hold information about software products. This ranges from tools for capturing loose and informal communication, such as whiteboard sketches and natural language, to constructing source code and configuration data. For knowledge management, all organizations use Confluence, Jira, and Git. Confluence is the designated tool for all kinds of documentation, including design decisions. Jira is typically used for task and process management, and Git is used for source management control. See Figure 3 for the collected data.

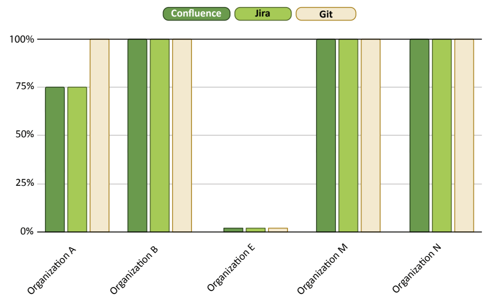


Figure 3: Usage of knowledge tools including design decisions. Git includes GitHub, GitLab, and Bitbucket.

3.2 Tenure of Team and Age of Repositories

The motivation for collecting this data is that it supports answering RQ2 and RQ3. A second motivation is that knowledge preservation becomes more relevant with applications aging because of the risk of knowledge vaporization. A team with senior developers working for years on an application or in the same organization shares embodied and tacit knowledge about values, principles, practices, and tools and processes, including changes over the years. See Figure 4 for the collected data.

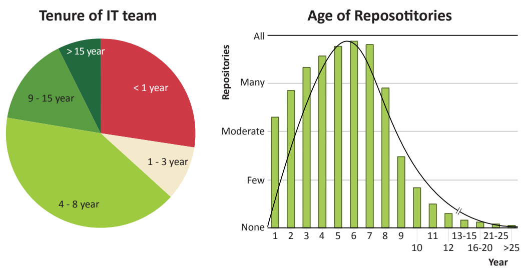


Figure 4: Tenure of team members for all organizations. Note that the diagram is skewed for < 1 year for the educational organization because students are only involved for one term (8-16 weeks). The scale of the right diagram is not proportional.

3.3 Just Enough Upfront

The reason for collecting this data is that it contributes to answering RQ2 and RQ3. Furthermore, this data supports and suggests modifying conditions, characteristics, practices, and use cases for the previously (Theunissen, Hoppenbrouwers, et al., 2022) defined artifacts. See Figure 5 for the usage of the artifacts. Interviewees mentioned knowledge elicitation, which compensates for missing artifacts, as an upfront activity in an educational context.

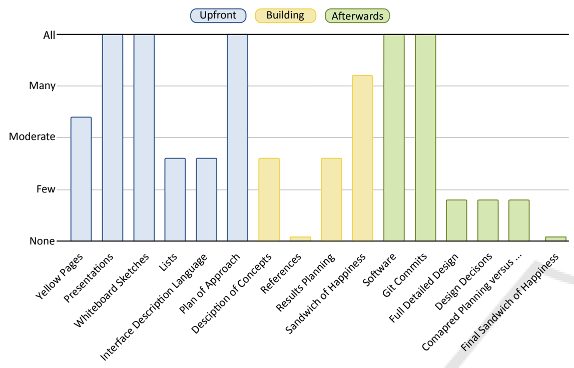


Figure 5: Usage of Artifacts for ’Just enough Upfront’ across all organizations.

3.4 Executable Documentation

The reason for collecting this data is that it contributes to answering RQ2 and RQ3. Furthermore, this data supports and suggests modifying conditions, characteristics, practices, and use cases for the previously (Theunissen, Hoppenbrouwers, et al., 2022) defined artifacts. See Figure 6 for the data. This data is skewed because it is not used in educational organizations.

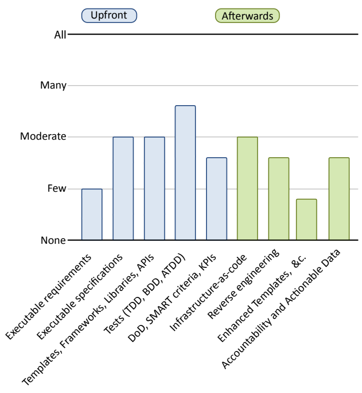


Figure 6: Usage of Artifacts for ’Executable Documentation’ across all organizations.

4 DATA ANALYSIS

Data analysis concerns the process of bringing order, structure, and meaning to the pile of collected data (Marshall & Rossman, 2015, p. 399). Furthermore, remarkable results and omissions are mentioned. For this study, we assessed if the approaches and artifacts did have the intended outcomes. With this, we follow the summative assessment from DS.

4.1 A Myriad of Tools That Contain Information

Using a software development ecosystem entails that information about the software product is distributed across all tools in the ecosystem. Designated tools are used for specific types of information. For this research project, we are interested in design decisions. Only three tools were used for this type of information in all organizations studied. These tools are Confluence, as a repository for knowledge about the software product, Jira, for task management; and Git, primarily for source code. For the range of types of information, varying from loosely communicated natural language and sketches on one side of the spectrum, to constructed source code that can compile to running applications on the other, design decisions are not stored in other tools. It may very well be possible that descriptions, interpretations, and explanations as answers to questions might be stored in chats, emails, or spoken language. However, we excluded these types of communication since they are not realistically retrievable. A typical workflow mentioned by

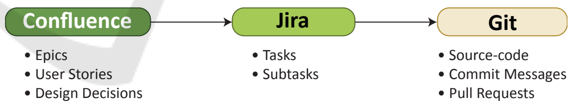


Figure 7: Typical workflow across all studied organizations.

the participants outlines the process of writing epics and user stories in Confluence. Tasks from the user stories are managed with Jira, source code is stored in Git, and the comments can be limited to just the task number from Jira. Within this flow, code traceability from task to user story to epic should be guaranteed for knowledge preservation, such as design decisions. See Figure 7. The flow matches with literature (Theunissen, van Heesch, & Avgeriou, 2022) and suggestions from Atlassian[1].

[1]https://www.atlassian.com/agile/project-management/user-stories

Observations from the case studies are as follows:

1. **No Confluence or Jira for Running Software Products.** One of the organizations shuts down instances of Confluence and Jira at the end of a project. The end of a project does not entail the retirement of the software product, only that information is lost about the software product. For that organization, the Git commit messages only contain a task number, and the rationale for the code changes could not be retrieved. This leads to knowledge evaporation.
2. **Tooling Without Design Decisions.** Another organization did not store design decisions and had to rely on a rather fuzzy vision without requirements or specifications for implementation. This organization became aware of the lack of recorded design decisions and started to store these decisions in hindsight.
3. **Short Lived Projects.** For educational organizations, knowledge preservation is not relevant. The lifetime from start to retirement is only one term (8, 16 weeks). Students are required to learn to use standards like a Software Requirements Specification (SRS) (Standards Committee, 1998), Software Architecture Description (SAD) (Technical Committee, 2011) or Software Design Description (SDD) (Standards Committee, 2009). Knowledge vaporization is immanent in this context.
4. **Migrations of Tools over the Years.** One organization has used software for dozens of years. Information about the software product must be available for over 40 to 50 years. Migration of tools did happen from paper to photocopies organized in a hierarchical file structure on disks to optical character recognition and eventually to Confluence.
5. **No Design Decisions in Git.** No organization mentioned keeping design decisions in Git. Discussing this made sense for participants, but it is no actual practice.

4.2 Tenure of Team and Age of Repositories

Compared with data from the U.S. (Bureau of Labor Statistics, U.S. Department of Labor, 2022), team members in the studied organizations keep working for the same organization much longer, except for the educational organization. This is a big difference compared to the tenures for developers in big tech companies, which is less than three years (Anonymous, 2017).

Observations from the case studies are as follows:

1. **Educational Contexts Require no History nor Seniority.** Students are involved in a project for a short time, and maintenance or support is not part of the program. The teams investigated are larger than in the industry. The team in the educational organization consists of 20 to 25 developers and is divided into sub-teams working on sub-systems. The primary objective for students is to pass the course and project, not to maintain or support a software product. Students remember tools, techniques, and processes rather than design decisions.
2. **Team with History in an Organization - Sharing Knowledge.** One of the participants explained that employees in non-profit organizations earn less money than in commercial companies but are more loyal to the organization. Some organizations have a network of family members.
   Employees stay much longer in the three governmental organizations than in commercial organizations. Team members have a history of past decisions, which is relevant because not all decisions are documented. This positively affects knowledge preservation, even if it is not documented. The memory of undocumented historic decisions is not identified as an artifact in the literature.
3. **Life Span of Software.** The moderate age of software repositories is six years, with a Gaussian distribution ranging from 1 year to 12 years (Hasselbring et al., 2020). The distribution is skewed with mode and median left from the mean because newer software is in use more than older software. The software can age for dozens of years for one of the studied organizations. This software is part of cyber-physical systems such as frigates and submarines.

4.3 Just Enough Upfront

Observations from the case studies are as follows:

1. **Artifacts Always Used.** Some artifacts were always used, for example, presentations, source code, and commit messages. Presentations are helpful in transferring knowledge that can serve as input for a project or iteration because the mix of high-level diagrams supports an understanding of causal and logical relations between concepts. Depth can be obtained by supporting text accompanying the diagrams (Ainsworth, 2006). Source code serves as a single source of truth and reveals what the software product does and how it works through close reading by (experienced) developers. It does not explain *why* the software prod-

uct works. Tools like Confluence or Jira are often used to record the reasons for modifications. For Git, individual commit messages were not valuable, as mentioned by several respondents. However, Pull Requests are in use that “should contain design decisions” instead of ordinary commit messages.

2. **Artifacts Never Used.** Some artifacts, such as references and a final Sandwich of Happiness (SoH), were never used. The artifacts SoH and Result Planning (RP) are not familiar in most organizations, but assessments and retrospection are common across all organizations.
3. **Knowledge Distributed Across Software Development Ecosystem.** In previous studies, we found that nowadays, many tools are in use that store pieces of information about software products (Theunissen et al., 2021), which remains true. However, design decisions, rationales, and reasons for change are typically stored in tools like Confluence or Jira.
4. **Design Decisions Keeping Close to Source Code.** Source code is the Single Source of Truth (SSOT). Developers can read the source code and understand *what* it does and *how* it works. However, the reason *why* the source code is as it cannot be retrieved from the source code itself. Typical tools in use are Confluence and Jira. We incorrectly expected that design decisions were stored as close as possible to source code, and not in other tools. To emphasize this point: one organization decided to turn off Confluence and Jira because the project was operational. As a result, the knowledge related to the project was gone.

4.4 Executable Documentation

Observations from the case studies are as follows:

1. **Artifacts Always Used.**There are no artifacts concerning Executable Documentation (ED) that are always used across all organizations.
2. **Artifacts Never Used.** ED is not used in many organizations, see Figure 6. Only a few organizations practice it. An exception concerns TDD. The highest score is because participants prefer other tests, especially unit tests. This type of testing often concerns the ‘happy flow’. TDD aims at writing tests that fail. In the case studies, one organization used ED for all projects in their division. This organization did not use any other documentation for knowledge acquisition or knowledge transfer. This implies a lack of testing of the software product. Developers mentioned reasons such as test cases that made it possible to reverse-engineer software that should be kept secret. Another organization with long-term software products (> 30 years) relies solely on its developers. Developers are not connected to the internet to ‘google’ an answer or use publicly available libraries and frameworks for security reasons.

5 DATA INTERPRETATION

In this section, the most remarkable results will be presented, being either a confirmation or a rejection of proposed artifacts. Artifacts such as presentations, whiteboard sketches, software, and git commits are used by all participants. Furthermore, following RQ3, defining characteristics for obstacles to implementation will be mentioned.

1. **Relax on Design Upfront.** Developers can start a project or iteration as soon as requirements and specifications are sufficient. Longer contemplation does not prevent progressive insights, especially in an educational context where progressive insights are implicit in the process and objectives. No obstacle hinders the implementation of relaxing on upfront design. Relaxing does not imply becoming sloppy. The effort starts with remembering values, principles, practices, and tools and processes to make proper judgments on requirements and priorities.
2. **Strict on Codified Interface Description.** Integration of (sub)systems is always hard, it also happens at the end (when deadlines are approaching), and the blame falls on others. Starting with, and holding on to, a codified interface prevents integration issues.
   Barriers to integration are related to being accountable for results one has no control over the other (sub)systems, teams, or external parties. This leads to short-sighted vision, losing the big picture, and shared responsibilities.
3. **Knowledge is Social.** Not all knowledge is stored in artifacts. Participants across all organizations mentioned that design decisions, rationales, or reasons for change are not documented. For some organizations where security, reliability, or confidentially are critical, knowledge about the software product is in the hearts and minds of senior developers. Knowledge is shared in meetings, conversations, or presentations. People are loyal to their organization, and knowledge stays within the organization. This supports knowledge preservation.
   A risk for accepting that knowledge could be found in social interaction concerns teams with

a high change rate of team members, including changes caused by internal reorganizations. The organizations we researched do not pay the highest salaries, but team members have been loyal to the organization over the years.

4. **Knowledge is Primarily *NOT* Distributed Across a Software Development Ecosystem.** Information about the software product is distributed across all tools in a software development ecosystem in an organization. However, some types of information, such as design decisions, rationales, or reasons for change, are kept in a single place.
   There is no barrier to this result. A communicated *modus operandi* supports knowledge preservation.
5. **Saving Design Decisions in Git.** It is common practice across all organizations to use Git (GitHub, GitLab, Bitbucket) for keeping source code. Typically, commit messages do not add much information about the source code changes as they mention *what* has been changed or *how* it works. As design decisions cannot be retrieved from source code, the best option for keeping this type of information is git commit messages. Some participants mention that Pull Requests are the designated commit types.
   We could not identify a hurdle for keeping design decisions as close to the source code as possible. Participants could not give a clear reason why not to do so. Speculation from the participants includes the way of working and tooling. The way of working is a behavioral change, and tooling is a management decision.

6 THREATS TO VALIDITY

Threats to validity in DS is not a mature discipline (Larsen et al., 2020). We follow Gonzalez and Sol (2012) with formative and summative assessments to evaluate the treatments. Formative assessment concerns the process of how a result is achieved, and summative assessment refers to the result ('does it work?') of the treatment. Wieringa (2014, pp. 128, 138) mentions the following threats for treatment design:

1. *Inference Support.* What are reasoning or statistical schemes to draw valid conclusions based on assumptions and observations?
2. *Repeatability.* Can data sampling and reasoning be reproduced several times, leading to the same conclusions by other researchers?
3. *Ethics.* Does the research harm participants?
4. *Interpretation.* Does the reader accept the interpretation as a fact?

Inference Support. This risk is mitigated by the verification of assumptions and hypotheses with the participants. An *assumption* is a statement that is considered to be valid and proven in an inference of taking to be invalid, which leads to an invalid conclusion by applying the principle of excluded middle. A *hypothesis* is considered to be true when it is not possible to falsify it. Some artifacts, as can be seen in Figures 5 and 6, were valid (value 'All') and some were not valid (value 'None'). Some were inconclusive (other values than 'None' or 'All'). For this threat, both formative and summative assessments are relevant. The formative assessment considers the validity of the reasoning process. The summative assessment concerns the validity of assumptions and hypotheses.
Repeatability. This risk is partially mitigated. For the data sampling, it is mitigated in principle. However, because of the classified source code of some organizations, clearance level is required [2] or legal requirements[3] are required. For this threat, summative assessments are relevant. With this assessment, the repeatability is evaluated as to whether results can be reproduced, partially when a situation is changed or can not be reproduced.
Ethics. Internal processes mitigate this risk in organizations by excluding privacy-sensitive data from the research. Some organizations have a legal task to use violence (military) or detect acts of crime. This could harm people (enemies, criminals) involved but not society as such. We take a utilitarian ethical viewpoint that values the happiness of society above the happiness of the individual. We consider the formative assessment applicable where the process is evaluated.
Interpretation. This risk is mitigated by method and data triangulation. We used a literature review, case studies, and a survey to have several methods. Furthermore, we used semi-structured interviews, executable and non-executable artifacts, and data from a survey. For this threat, both formative and summative assessments are applicable. The formative assessment concerns the process of collecting and interpreting the data. For the summative assessment, results might differ because of progressing insight, whereas a different result might emerge with identical assumptions and hypotheses.

[2]'Verklaring omtrent Gedrag' (VOG, Certificate of Conduct) or 'Verklaring van geen bezwaar' (VGB, Certificate of no Objection).

[3]'Wet op de Openbaarheid van Bestuur' (WOB, 'Dutch Freedom of Information Act').

7 CONCLUSIONS

We have three research questions to answer. The first research question concerns the necessary and sufficient conditions to use approaches and artifacts. Shared values, principles, practices, tools, and processes are critical for necessary and sufficient conditions. The second research question assesses the actual usage of approaches and artifacts. 'Just enough Upfront' is an approach that is used across all organizations or is considered appealing. Some artifacts are constantly in use: presentations, whiteboard diagrams, plans of approach, software, and commit messages. Some are never used, such as references and final SoH. For the approach 'Executable Documentation,' no conclusive artifacts are included or excluded. The artifacts that were most in use were tests. The third research question concerns the defining characteristics of barriers to implementing the approaches and artifacts. An obstacle mentioned across all organizations concerns unconfirmed, loose deviations from prescribed processes or interfering objectives. Examples of loose deviations of prescribed processes are left-outs of ceremonies of textbook definitions of Scrum or Scaled Agile Framework (SAFe). Conclusive remarks concern one observation and one consideration. The observation is about the social construction of knowledge, where knowledge is not a mere act of intellect or rational or intelligible epistemic contemplation. The consideration concerns design decisions, rationales, or reasons for change that should be saved as close as possible to the source code in Git. What and how of the source code can be read in the code. A rationale cannot be retrieved from the source code. Separation of source code and design decisions does not contribute to knowledge preservation.

REFERENCES

Ainsworth, S. (2006). DeFT: A Conceptual Framework for Considering Learning With Multiple Representations. *Learning and instruction, 16*(3), 183–198. https://doi.org/10.1016/j.learninstruc.2006.03.001

Anonymous. (2017). *HackerLife*. hackerlife. Retrieved September 11, 2022, from https://hackerlife.co

Bureau of Labor Statistics, U.S. Department of Labor. (2022). *Employee Tenure in 2012: The Economics Daily: U.S. Bureau of Labor Statistics.* Retrieved September 11, 2022, from https://www.bls.gov/opub/ted/2012/ted20120920.htm

Gonzalez, R. A., & Sol, H. G. (2012). Validation and Design Science Research in Information Systems. In M. Mora (Ed.), *Research methodologies, innovations and philosophies in software systems engineering and information systems* (pp. 403–426). IGI Global.

Hasselbring, W., Carr, L., Hettrick, S., Packer, H., & Tiropanis, T. (2020). Open source research software. *Computer*, 53(8), 84–88.

Larsen, K. R., Lukyanenko, R., Mueller, R. M., Storey, V. C., VanderMeer, D., Parsons, J., & Hovorka, D. S. (2020). Validity in Design Science Research. In S. Hoffman, O. Müller, & M. Rossi (Eds.), *International Conference on Design Science Research in Information Systems and Technology* (pp. 272–282). Springer. https://doi.org/10.1007/978-3-030-64823-725

Lee, A. S., & Hubona, G. S. (2009). A Scientific Basis for Rigor in Information Systems Research. *MIS quarterly*, 237–262.

Marshall, C., & Rossman, G. B. (2015, January 7). *Designing Qualitative Research*. SAGE Publications, Inc.

Standards Committee. (1998). *IEEE Recommended Practice for Software Requirements Specifications* (Vol. 1998). http://www.math.uaa.alaska.edu/\%7B\%\%7D7B\%7B~\%7D\%7B\%\%7D7Dafkjm/cs401/IEEE830.pdf

Standards Committee. (2009). IEEE Std 1016-2009 (Revision of IEEE Std 1016-1998), IEEE Standard for Information Technology–Systems Design–Software Design Descriptions. Joint Technical Committee ISO/IEC JTC 1. https://doi.org/10.1109/IEEESTD.2009.5167255

Technical Committee. (2011). *ISO/IEC/IEEE 42010:2011 - Systems and Software Engineering – Architecture Description* (ISO/IEC/IEEE). Joint Technical Committee ISO/IEC JTC 1. Geneva, Switzerland. https://www.iso.org/standard/50508.html

Theunissen, T., Hoppenbrouwers, S., & Overbeek, S. (2021). In Continuous Software Development, Tools Are the Message for Documentation. In J. Filipe, M. Smialek, A. Brodsky, & S. Hammoudi (Eds.), *Proceedings of the 23th International Conference on Enterprise Information Systems.* SCITEPRESS - Science; Technology Publications. https://doi.org/10.5220/0010367901530164

Theunissen, T., Hoppenbrouwers, S., & Overbeek, S. (2022). Approaches for Documentation in Continuous Software Development. *Complex Systems Informatics and Modeling Quarterly (CSIMQ)*, 32, 1–27. https://doi.org/10.7250/csimq.2022-32.01

Theunissen, T., & van Heesch, U. (2016). The Disappearance of Technical Specifications in Web and Mobile Applications. In B. Tekinerdogan & U. Zdun (Eds.), *Software Architecture* (pp. 265–273). Springer International Publishing. https://doi.org/10.1007/978-3-319-48992-620

Theunissen, T., van Heesch, U., & Avgeriou, P. (2022). A Mapping Study on Documentation in Continuous Software Development. *Information and Software Technology*, 142, 106733. https://doi.org/10.1016/j.infsof.2021.106733

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering.* Springer Berlin Heidelberg. Retrieved January 30, 2021, from https://doi.org/10.1007/978-3-662-43839-8

Yin, R. K. (2016). *Qualitative Research from Start to Finish* (Second edition). The Guilford Press OCLC: 935783468.