A mapping study on documentation in Continuous Software Development

Theo Theunissen a,*, Uwe van Heesch a,b, Paris Avgeriou b

a HAN University of Applied Sciences, The Netherlands
b University of Groningen, The Netherlands

ARTICLE INFO

Keywords:
Systematic mapping studies
Systematic reviews
Continuous Software Development
Lean
Agile
DevOps
Documentation

ABSTRACT

Context: With an increase in Agile, Lean, and DevOps software methodologies over the last years (collectively referred to as Continuous Software Development (CSD)), we have observed that documentation is often poor.

Objective: This work aims at collecting studies on documentation challenges, documentation practices, and tools that can support documentation in CSD.

Method: A systematic mapping study was conducted to identify and analyze research on documentation in CSD, covering publications between 2001 and 2019.

Results: A total of 63 studies were selected. We found 40 studies related to documentation practices and challenges, and 23 studies related to tools used in CSD. The challenges include: informal documentation is hard to understand, documentation is considered as waste, productivity is measured by working software only, documentation is out-of-sync with the software and there is a short-term focus. The practices include: non-written and informal communication, the usage of development artifacts for documentation, and the use of architecture frameworks. We also made an inventory of numerous tools that can be used for documentation purposes in CSD. Overall, we recommend the usage of executable documentation, modern tools and technologies to retrieve information and transform it into documentation, and the practice of minimal documentation upfront combined with detailed design for knowledge transfer afterwards.

Conclusion: It is of paramount importance to increase the quantity and quality of documentation in CSD. While this remains challenging, practitioners will benefit from applying the identified practices and tools in order to mitigate the stated challenges.

1. Introduction

In recent years, we have seen an increase in the adoption of Lean and Agile software development, as well as DevOps. In our previous work [387, 389, 3402], we have introduced the term Continuous Software Development (CSD) as an umbrella term to collectively refer to such development processes and other processes that share the following characteristics:

(a) it covers the values, principles and practices from Agile [48], Lean [317] and DevOps.

(b) it embraces activities from the whole life cycle of a software product, from concept to end-of-life. In addition to Agile and Lean software development, it includes maintenance activities. In addition to DevOps, it includes continuous architecting activities [41].

(c) it considers the continuously changing state of the software product and progress, such as progressive insights (e.g. regarding process, design, implementation), changes in contextual factors, new features, bug fixes, or other unforeseen factors.

(d) it distributes information about software development across multiple tools, because of demands for fast time-to-market, as well as the need for a software development ecosystem for automated tests, deployment, and monitoring. Thus, no central repository for all information is available.

One of the main challenges in CSD is that documentation is poor [48, 317, 194]. This challenge hinders knowledge transfer [558], has a bad impact on maintenance [284] and introduces a steep learning curve [377] for new team members. We elaborate further on these consequences. First, knowledge transfer is hindered when knowledge about the software product, such as decisions, bugs, context, and practices, remains implicit in the minds of developers and is only informally written in whiteboard sketches. As a result, knowledge walks literally out the door at the end of a daily stand-up or even leaves the company (many companies have high staff turnover) [205, 280, 59]. Second, it is hard to act when bugs show up, new features or non-functional requirements arise. Developers are forced to make
assumptions about decisions, interfaces, or priorities; such assumptions are often wrong [251,43,166]. Third, the system is hard to understand as what is still lacking. The rationale typically concerns trade-offs between qualities, business factors, in-house expertise etc. [394,540]. (6) Design and/or architecture specifications. Even if design specifications are created, they are typically not updated according to changes in requirements and context, and thus become out-of-sync with the actual code [101].

As a first step in addressing the problem of poor documentation in CSD, we decided to look into the current state of practice as reported in scientific literature. Specifically, we conducted a systematic mapping study on identifying the challenges of documentation in CSD as well as the practices and tools that can potentially support documentation. We selected to study these aspects in order to shed light into both the problem (documentation challenges) and the solution (practices and tools); we note that practices and tools are the primary means for architecture documentation [384]. Our aim is to shed light on what is currently on offer for documentation purposes in a CSD context, as well as what is still lacking. Our results indicate that documentation is considered waste in Lean development when it does not contribute to the end product. Consequently, developers tend to minimize documentation or leave it out. Furthermore, documentation is often out-of-sync with the software, irrespective of whether documentation is within the source code or documented in wiki-like systems. Moreover, the focus is only short-term: knowledge about design decisions, practices, and lessons learned are within a team, primarily when the team is gathered in a single geographical location. The practices we discovered are that written documentation is left out, and communication is informal, while development artifacts are used as a specification. Finally, the use of architecture frameworks can also support sound documentation.

We decided to conduct a systematic mapping study (SMS) instead of a systematic literature review (SLR). Systematic Mapping Study (SMS) are typically used for newer research topics where there are few or no secondary studies and the main objective is to classify and conduct a thematic analysis of literature [232,312]. Further motivation for the use of SMS versus SLR is provided in the beginning of Section 2.

1.1. Research questions

We formulate the goal of the study using the format of the Goal-Question-Metric (GQM) approach [39]: Analyze literature for the purpose of exploration, characterization and analysis with respect to documentation challenges, practices, and tools from the point of view of researchers and industry practitioners in the context of Continuous Software Development. Based on the aforementioned goal, we have set the following research questions:

RQ1 What are the documentation challenges and specific practices in CSD?

We already know of several challenges in CSD. We have established that poor documentation hinders knowledge transfer [558], which, in turn, has a bad impact on maintenance [5284] and introduces a steep learning [3377] curve for new team members. Furthermore, documentation seems to have a lower value in CSD, than in traditional software development processes such as RUP [245]. For example, the Agile Manifesto explicitly values working code over written documentation; face-to-face communication is considered the most effective way of conveying information [48]. In Lean software development, documentation is often considered waste, as it does not directly contribute to customer satisfaction [317]. In DevOps, infrastructure is key to fast deployment and information is represented as code [194], rather than written documentation. With this research question, we aim at understanding in more depth such challenges that work against documentation in CSD. We also strive to uncover potential practices that result in successful documentation in the context of CSD.

RQ2 Which tools from the Continuous Software Development ecosys tem can be used for documentation purposes?

CSD relies heavily on tooling in order to achieve faster deployment, continuous testing, and monitoring quality [5224]. These tools contain much information about source code and configuration (e.g. git), test cases (e.g. Cucumber), deployment (e.g. Docker or Jenkins) and quality (e.g. SonarQube). This information would typically also be documented in software design description documents. With this research question, we want to understand which tools are used in CSD and how they could additionally be exploited for documentation purposes.

1.2. Related secondary studies

There are several secondary studies on the topics of Lean, Agile and DevOps. In the following, we describe those studies that discuss documentation in Lean, Agile and DevOps. We also present the reason why they are related and which gap our study attempts to address. These five studies address issues, describe industry practices, and propose and explore processes, tools and methods.

Rodríguez et al. analyze the body of knowledge in Continuous Deployment [528]. They give an overview of concepts and typical characteristics of continuous deployment, such as fast and frequent releases, and continuous automated testing. The authors emphasize the importance of tools for supporting continuous integration and continuous delivery, but they do not address the relation between tooling and documentation.

Diebold et al. looked into agile practices in the industry under different circumstances, such as different project types, domains, or processes [116]. They found that agile practices appear in most projects across several industry domains. Such agile practices are used in methods like Scrum, Kanban, and eXtreme Programming (XP). The study focuses specifically on the development activities that lead to the first major release, whereas maintenance concerns are not particularly taken
into consideration. The study does not cover documentation in agile projects.

In two different secondary studies on requirements engineering in agile software development, Heikilä et al. [178] and Curcio et al. [104] independently found that there is no clear line on how requirements engineering activities should be performed in agile processes; the overall understanding of requirements engineering in agile software development is still rather immature. Both studies report that an agile development team usually comprises highly skilled and experienced developers who act on their knowledge and skills; this knowledge and the thought process of developers is usually not written down in documents, i.e. agile teams rely mainly on tacit knowledge. Furthermore, these highly skilled professionals are often required for other jobs and frequently switch teams. New team members, who might be less qualified and experienced, do not know the decisions and actions taken. Heikilä et al. suggest that knowledge should be written down for the new team members [178]. Neither the study of Heikilä et al. [178], nor the one of Curcio et al. [104] discussed documentation practices, and tools.

Shafiq et al. found that agile development teams often make use of predefined document templates as a means for efficient standardization [355]. For instance, Feature-Driven Development (FDD) uses templates for use cases and functional requirements. Scrum uses user stories (as <role>), I want <objective> because of <rationale>). Generally, agile teams avoid recording long, complex, strictly-defined or rigid pieces of information in textual documents.

In summary, documentation concerns in CSD are gaining attention within the research community. However, there is currently no consensus on concrete documentation practices. There is no practice or documented tooling that can be used for documentation purposes; instead, information is distributed across software development tools.

1.3. Paper structure and reference styles

The remainder of this document is structured as follows: in Section 2, we present the study design. Section 3 provides demographic information about the selected primary studies. In Section 4, we discuss the results and provide our own interpretation, as well as implications for researchers and practitioners. Finally, we discuss potential threats to the validity of this work in Section 5.

We use two styles of references in this study. One style refers to the primary studies that are analyzed to answer the research questions. These references are denoted with an S (for study) and a number within square brackets, e.g. [S123] refers to study 123 that is shown in Appendix A. The other style of reference is without the ‘S’; it refers to papers that do not belong to our set of primary studies but are used for other purposes (for instance in the Related Secondary Studies section) and can be found in the References.

2. Study design

As a method to conduct this literature study, we considered a systematic literature review (SLR), as well as a systematic mapping study (SMS). Table 1 is adapted from Kitchenham et al. [232] and compares typical characteristics of the two methods.

Using these seven characteristics, we justify why we used the systematic mapping study as follows:

(a) Goals. We want to present a broad overview of literature and to categorize this literature in dimensions.

(b) Research Questions. We address broader research questions regarding the trends in documentation challenges, practices, and tools in CSD.

(c) Search Process. We are looking into a specific topic area: documentation in CSD.

(d) Scope. We focus on both empirical and non-empirical studies. The topics of Agile, Lean and DevOps are very practitioner-oriented, thus we expect that, at least part of literature is not empirical.

(e) Search strategy requirements. We are looking at trends, so we can afford to be less stringent.

(f) Quality Evaluation. The combination of non-empirical and empirical studies makes it complicated to evaluate the quality of primary studies.

(g) Results. We aim at classifying papers into dimensions.

Based on these reasons, we chose a systematic mapping study over a systematic literature review. We follow the guidelines of Petersen et al. for systematic mapping studies [313]. Fig. 1 depicts the steps of the study as well as the steps of the study protocol. Arrows pointing in both directions indicate that steps were performed iteratively. In the following sections, we briefly describe each of the steps of the study protocol (right part); the steps of “Phase 2: Execute study” (left part) are elaborated in Section 2.4.

Fig. 1 shows the contributions for all team members for each process step. The team comprises four researchers (labeled A, B, C, and D), varying in seniority. Two researchers selected studies, read the title, keywords, abstract and full paper. This resulted in a raw result set with candidate studies. Two other researchers read only the title, keywords and abstracts of the studies in the raw result. Studies that did not contribute to answering the research questions were rejected from the final result set. Finally, all team members read papers from the final result set. In Fig. 4, we made a distinction between raw results with candidate studies and final result sets with studies that contribute to answering research questions. Thus, it shows results per step and studies that were read concerning titles, keywords, abstracts, and full papers. By making a distinction between raw results and final results, we established a process for reaching consensus.

2.1. Search strategy

The search process combines a manual process with automated search. A manual search process typically has a higher accuracy than automated search, because it focuses on targeted venues, but it also has a risk of bias, because of the researcher’s personal preferences. Additionally, it is more time-consuming. Other criteria such as transparency and reproducibility are hard to achieve with a manual search, even if all quality and evaluation criteria are explicitly defined [228, 235]. Furthermore, automated search is typically more comprehensive than manual searches [228, 235]. We therefore decided to apply a combination of both methods. The manual search process, as well as the automated search will be further elaborated in Section 2.4.

2.2. Scope of search and sources searched

The scope for this study is limited by the following criteria:

1. The study is published between January 2001 (i.e. the publication of the Agile Manifesto [48]) and February 2019 when the writing of this report started;
2. The study can be found in scientific databases in the field of software engineering, that include journals, conference papers, and workshop papers; the following sources were used: ACM, IEEE, ScienceDirect, SpringerLink and WebOfScience. These databases are quite commonly used in secondary studies in Software Engineering [67].

2.3. Inclusion and exclusion criteria

Inclusion and exclusion criteria help to make a transparent and reproducible selection of papers in the mapping study. Papers are included if they meet all of the inclusion criteria and excluded if they meet any of the exclusion criteria. The criteria are exhibited in Tables 2 and 3.
Table 1
Comparison of typical characteristics in literature research methods.

| Characteristic | Systematic Literature Review (SLR) | SMS |
|----------------|-----------------------------------|-----|
| Goals          | Identification of best practices  | Classification and thematic analysis of literature |
| Research Questions | Specific — related to outcomes of empirical studies | Generic — related to research trends |
| Scope          | Based on research questions       | Defined by topic area |
| Search strategy requirements | Focused — outcomes of empirical studies | Broad — includes non-empirical studies |
| Quality evaluation | Important. Results must be based on best-quality evidence | Not essential. Non-empirical studies may make quality evaluation hard |
| Results        | Using outcomes of primary studies to answer specific research questions | Categorization of papers into dimensions |

Table 2
Inclusion criteria.

| ID  | Inclusion criteria                          |
|-----|---------------------------------------------|
| I1  | PDF or full text must be available          |
| I2  | Domain or discipline must be software engineering |
| I3  | Study must be written in English            |
| I4  | Study must be peer reviewed                |
| I5  | The search terms must appear in title, keywords or abstract |

Table 3
Exclusion criteria.

| ID  | Exclusion criteria                          |
|-----|---------------------------------------------|
| E1  | Study is a duplicate of another study in scope |

2.4. Search process

The search process follows the steps in the execution phase (see Fig. 1, left part). The study was conducted by four researchers. The data collection was done by the corresponding author with the assistance of a master student; the analysis and interpretation was performed by all authors. We began the process with the snowballing technique by following Wohlin’s guidelines (see Fig. 2) [427]. Specifically, we formed an initial set of papers on subjects, topics, and authors we found relevant for this SMS. Additionally, we asked subject-matter experts from academia and industry to come up with papers they deem primarily relevant for this study (see Appendix C). With the resulting set of papers, we conducted the snowballing technique until no more relevant publications could be found.

The resulting set of papers was used to define a quasi-gold-standard (QGS), which is a “well-known” set of papers that are relevant to evaluate the results and to establish a search string for an automated
search [436]. The search string was based on the QGS. The performance of the search string was performed by comparing the results of the automated search with the QGS: all papers from the QGS were returned from the automated search.

Subsequently, we defined the search string, based on the words from the title, keywords and abstract from the papers in the QGS. We used the n-gram procedure to assist in establishing the search string [81]. Specifically, we first removed 1000 common English stop words. Next, from the remaining words we took sequences of words to cover the domain (CSD) and the research questions. A manual step was required to adjust the search string to make it more efficient by removing unnecessary terms. Especially for RQ2, we added a wildcard to leave the single “document” out, as searching for “document” resulted in too many irrelevant hits. The resulting search string we used for the automated search is:

```
-- domain
  (lean
  OR agile
  OR DevOps
  OR "continuous software"
  OR scrum
  OR "extreme programming"
  )
-- RQ1
  AND
  (documenti* OR documenta*)
  )
-- RQ2
  AND (tool*)
```

With the search string defined, the execution of the database search was performed. For this, we scraped the meta-data from the online libraries to store it locally in our database. The reason for local storage is to compare studies equally. In the first place, the online search engines all do have a different query language which look similar when it comes to syntax, but the one online library is more precise in targeting than the other online library, especially the discipline (e.g. SpringerLink) or domain (e.g. ACM). Second, the online libraries do not use the same data model for the bibliographical data, for instance, the meta-data has a different format (BibTeX, RIS). Another difference is the type of the fields, such as the fields for "authors", "titles" and "keywords" might either be a string or a list. The third reason is to be able to add tags, labels and comments for answering research questions.

2.5. Data extraction

For each study, the information shown in Table 4 was collected. The attributes for the title (F1), keywords (F2) and abstract (F3) where used for snowballing and calibration of the search string. We used Mendeley⁵ for storing and tagging the papers during the initial phases. With the tags it was easy to select the papers. We commented the papers with keywords and comments for relevance, as well as terms that can be used for the search string. The suggestions for studies from external experts were also stored in Mendeley and tagged accordingly. During snowballing and establishing the Quasi-Gold Standard, Mendeley was used to store, comment the papers and add keywords. With the automated search, the results were too many to store in Mendeley, thus we used a database to store them. The database was structured according to the basic BibTeX bibliographic references, supplemented with extra fields for additional keywords, categories, and concepts.

The attribute full text (F4) was used in exploring the research area in the pilot search. The attribute values for publishers database (F5) and year (F6) were required by the inclusion and exclusion criteria. Attributes F7 and F8 were used for answering the two research questions.

2.6. Data analysis

For the analysis of quantitative data, we used descriptive statistics. For the analysis of qualitative data, we adapted the approach of Miles et al. [277], as depicted in Fig. 3. As supporting tooling, we used Atlas.ti.⁶ We started with the studies from the final result set. Next, we read the studies and marked text when it answered or contributed to a research question. This resulted in marked text. In the second step, we coded the marked text with keywords that characterize the fragment. The keywords could be individual words from the marked text but also other words that are typical for the marked text. We also used an online thesaurus and used Google to come up with additional or alternative keywords. The result of the coding step is a list of keywords. The third activity was grouping keywords into categories. Categories are a higher-order abstraction of the keywords. The result of the grouping of keywords is a list of categories. The fourth activity concerns identifying relations between categories. The relations denote connections among categories. The types of relations are derived from Unified Modeling Language (UML): activity edges, associations, dependencies, generalizations, realizations, and transitions. We kept the number of relations to a minimum to have clear distinctions between the resulting concepts. The activities for keywords, categories, relations, and concepts were iterated until no more refinement was possible. The last activity was the mapping of the concepts on the systematic map (see Fig. 7).

3. Results

This section describes the results of the mapping study, according to the guidelines of Petersen et al. [313]. First, we show the demographic data of the identified studies (Section 3.1). Then we classify the studies according to our research questions using a facet map (the systematic map) in Section 3.2. Finally, we discuss the results in the context of each individual research question: RQ1 in Section 3.3 and RQ2 in Section 3.4.

3.1. Demographic data

As described in 2.4, we used a two-fold search strategy comprising (1) a purely manual search based on input from subject-matter experts and snowballing and (2) an automated search. The results from both search types were merged, resulting in 58 unique papers that were used for answering our research questions. Fig. 4 illustrates this process again together with the numbers of papers resulting from each individual step. The initial set of four papers was relevant content-wise, but did not pass our inclusion criteria, because they are not scientific studies. Nevertheless, they served as a basis for the snowballing procedure, together with the input from the external experts, who suggested 39 articles in total (see Appendix C), out of which three papers matched our criteria. The snowballing procedure delivered 92 studies. We studied these articles to select a set of nine studies (shown in Table 5) that we consider a Quasi-Gold Standard.

The QGS was used to validate the search string for the automated search, i.e. we tweaked the search string iteratively until all studies in the QGS ended up in the results of the search. Table 6 shows the search

---

3 https://gist.github.com/deekayen/4148741.
4 https://www.mendeley.com.
5 https://en.wikibooks.org/wiki/LaTeX/Bibliography_Management.
6 https://atlasti.com/.
Table 4
Data extraction form.

| ID | Attribute                  | Usage                                      |
|----|---------------------------|--------------------------------------------|
| F1 | Title                     | snowballing, search string calibration, synthesis |
| F2 | Keywords                  | snowballing, search string calibration      |
| F3 | Abstract                  | snowballing, search string calibration      |
| F4 | Full text                 | exploration                                |
| F5 | Publishers database       | inclusion/exclusion                        |
| F6 | Year                      | inclusion/exclusion                        |
| F7 | Documentation, documenting, architecture framework | RQ1 |
| F8 | Tools, tooling            | RQ2                                        |

Table 5
The Quasi-Gold Standard.

| ID | Publication                                                                 |
|----|-----------------------------------------------------------------------------|
| [S407] | A Study of Documentation in Agile Software Projects - Voigt, Stefan; von Garrel, Jorg; Muller, Julia; Wirth, Dominic |
| [S372] | A study of the documentation essential to software maintenance - de Souza, Sergio Cozzetti B; Anquetil, Nicolas; de Oliveira, Kathia M |
| [S373] | Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects - A. Rüping |
| [S402] | Software Specification and Documentation in Continuous Software Development: A Focus Group Report - Van Heesch, U.; Theunissen, T.; Zimmermann, O.; Zdon, U. |
| [S389] | Specification in Continuous Software Development - Theunissen, T., Van Heesch, U. |
| [S406] | SprintDoc: Concept for an agile documentation tool - Voigt, Stefan; Huttemann, Derle; Gohr, Andreas |
| [S334] | Supporting agile software development through active documentation - Rubin, Eran; Rubin, Hillel |
| [S159] | Towards the essentials of architecture documentation for avoiding architecture erosion - Gerdes, Sebastian; Jasser, Stefanie; Riebach, Matthias; Schröder, Sandra; Soliman, Mohamed; Stehle, Tilman |
| [S346] | Using design rationale for agile documentation - Sauer, T |

Fig. 3. Qualitative classification process on the final set (Miles et al. [277]).

Fig. 4. Overview of search results per step.

string, its relation to the research questions, and the number of hits in abstracts, keywords, or title, respectively. The column Intersection shows the number of papers in which the search term was found in all three parts of the studies. In total, we ended up with 58 unique papers that relate to our research questions (i.e. 40 for RQ1 plus 23 for RQ2 makes a total of 63 non-unique papers).
Table 7 shows the papers identified during the manual search, Table 8 shows the automated search results. The union resulted in 58 unique studies (see Table 9) that were analyzed in the next step.

The distribution of studies according to publication types is displayed in Fig. 5. About 80% studies are from conferences. Fig. 6 plots the publication years of all identified studies. Details are printed in Table 10. Clearly, the topic is increasingly gaining attention in the research community. decided to look into the current state of practice.

3.2. Classification scheme using a systematic map

As a next step in the analysis, we classified the studies using a systematic map, as described by Petersen et al. [313,312]. Each study was categorized using three facets: (1) a contribution facet covering the type of contribution to the software engineering domain, (2) a research type facet describing the type of study and (3) a context facet that maps the content of the studies to our research questions. Fig. 7 shows the resulting map using two bubble charts. The size of the bubble represents the number of studies falling into the corresponding categories. The absolute number of studies is shown in the centers of the bubbles followed by a letter that refers to the list of studies that can be found in Appendix B. For example, in the coordinate plane between the Context Facet “Research” and Research Facet “Solution Proposal”, the bubble represented by the letter “k” shows that 17 studies have been found. Studies can appear in multiple facets. The total number of 58 unique studies has been mapped 200 times (see Table 10).

For the contribution and research facets, we used existing classification schemes by Petersen et al. [313], and Wieringa [423], respectively. The classification scheme for the Contribution facets from Peterson [313] describes the potential categories of a paper’s contribution: Metric, Tool, Model, Method, and Process. For our Systematic Mapping Study, we moved the Tool category to the Context facet, because RQ2 concerns tools. The classification scheme for the Research facets from Wieringa [423] includes six types, four of which were found in our primary studies:

1. **Validation Research.** The investigation of a problem or implementation of a technique in practice.

2. **Evaluation Research.** The validation of a solution proposal that has not yet been investigated.

3. **Solution Proposal.** This type of papers contains solution proposals without validation.

4. **Experience Papers.** This type of papers describe what has been experienced by the author as matters of fact, and do not describe the reasons why.

The categories of the context facet evolved while doing the data extraction. We merged categories where appropriate to keep the number of categories small so we could plot them against the other two facets. In the following, those categories are briefly described. Additionally, we assign each category to one of the research questions:

1. **Documentation life cycle:** aspects of creation, maintenance and management of documentation artifacts (RQ1).

2. **Documentation subjects:** architecture: architecture related documentation such as design, solutions and architecture description (RQ1).

3. **Documentation subjects:** source-code: the documentation of source-code and source-code related aspects such as version control (RQ1).

4. **Documentation subjects:** autogenerated documentation: the storing and retrieval of documentation that is scattered throughout a software ecosystem and is stored in tools such as git commits, Jira tasks or wiki-like documents (RQ1). Please note that we omitted the term “documentation” in the bubble chart to ease readability.

5. **Documentation subjects:** decisions: software architecture decisions and their rationale (RQ1).

6. **Tool:** how tools are used in supporting documentation in continuous software development (RQ2).

Fig. 7 shows that architecture documentation is a popular topic within the studies (42 papers in total), predominantly as solution proposals (21 papers) and evaluation research (18 papers). The documentation life cycle is also found in many studies (28 papers), as well as source-code documentation (25 papers), again primarily in the shape of solution proposals or evaluation research.

The most frequent contribution types of the identified studies are method (44 papers), metrics (35 papers), and models (27 papers); all three are mostly found on architecture documentation. It is notable that the least number of studies map to the tool category, which we consider counter-intuitive as continuous software development is a discipline that makes vast use of tool-ecosystems and automation.

3.3. Results for RQ1: Documentation challenges and practices

This section describes the results of our analysis on studies assigned to RQ1 (What are documentation practices and resulting challenges in CSD?). Table 9 lists all studies considered in this analysis. Continuous software development, as mentioned in the Introduction section, is not a development process model on its own; it is rather an umbrella term for existing methods that share certain characteristics. The papers found in this mapping study cover the following specific process models and methods:

- Lean Software Development [317],
Table 7
Studies contributing to answering the research questions from the manual search.

| RQ | Found | Studies |
|----|-------|---------|
| RQ1 | 38    | [S23], [S55], [S56], [S57], [S58], [S68], [S89], [S90], [S372], [S114], [S156], [S159], [S167], [S181], [S206], [S221], [S255], [S261], [S274], [S284], [S316], [S322], [S334], [S340], [S346], [S364], [S375], [S376], [S377], [S407], [S413], [S415], [S115], [S175], [S184], [S199], [S355], [S389], [S410] |
| RQ2 | 18    | [S8], [9], [S17], [S23], [S57], [S109], [S112], [S149], [S217], [S250], [S265], [S316], [S340], [S406], [S413], [S414], [S422], [S224] |

Table 8
Studies contributing to answering the research questions from the database search.

| RQ | Found | Studies |
|----|-------|---------|
| RQ1 | 40    | [S23], [S55], [S56], [S57], [S58], [S68], [S89], [S90], [S372], [S114], [S156], [S159], [S167], [S181], [S206], [S221], [S255], [S261], [S274], [S284], [S316], [S322], [S334], [S340], [S346], [S364], [S375], [S376], [S377], [S389], [S407], [S413], [S415], [S115], [S132], [S175], [S184], [S199], [S355], [S389], [S410] |
| RQ2 | 23    | [S8], [9], [S17], [S23], [S24], [S57], [S109], [S112], [S149], [S217], [S225], [S250], [S265], [S316], [S330], [S340], [S309], [S406], [S413], [S414], [S422], [S214], [S224] |

Table 9
Final set of studies that contribute to answering the research questions.

| RQ | Found | Studies |
|----|-------|---------|
| RQ1 | 40    | [S23], [S55], [S56], [S57], [S58], [S68], [S89], [S90], [S372], [S114], [S156], [S159], [S167], [S181], [S206], [S221], [S255], [S261], [S274], [S284], [S316], [S322], [S334], [S340], [S346], [S364], [S375], [S376], [S377], [S389], [S407], [S413], [S415], [S132], [S175], [S184], [S199], [S355], [S389], [S410] |
| RQ2 | 23    | [S8], [9], [S17], [S23], [S24], [S57], [S109], [S112], [S149], [S217], [S225], [S250], [S265], [S316], [S330], [S340], [S309], [S406], [S413], [S414], [S422], [S214], [S224] |

![Fig. 6. Distribution per year.](image_url)

• Scrum [48],
• Extreme Programming (XP) [S139],
• Feature-Driven Development (FDD) [S305],
• Crystal Clear [S95],
• Adaptive Software Development (ASD) [S182],
• Dynamic systems development method (DSDM) [S220,355],
• Microsoft Solution Framework (MSF) [S13,355],
• Agile Unified Process (AUP) [254,355],
• and Test Driven Development [S46].

In the context of these process models and methods, we identified the documentation challenges listed below. By documentation challenges, we refer to obstacles that developers face towards documenting knowledge about the system or keeping it up to date.

1. **Informal documentation is hard to understand.** As stated above, the sparse written documentation in CSD is often informal and volatile (e.g. white board sketches and drawings [S375, S156]). Hadar et al. refer to the different backgrounds from architects, reviewers and other stakeholders and note that it is rather cumbersome for one stakeholder to understand and improve the informal documentation of another [S167]. Such types of informal documentation artifacts require a kind of “voice-over” or additional explanation to be effective for knowledge transfer [S376].
### Table 10
Publications per year for RQ1, RQ2 and totals per year and RQ.

| Year | RQ1 | RQ2 | Total |
|------|-----|-----|-------|
| 2001 | 1   | 0   | 1     |
| 2002 | 1   | 0   | 1     |
| 2003 | 3   | 0   | 3     |
| 2004 | 0   | 1   | 1     |
| 2005 | 3   | 0   | 3     |
| 2006 | 0   | 0   | 0     |
| 2007 | 0   | 2   | 2     |
| 2008 | 4   | 5   | 9     |
| 2009 | 0   | 4   | 4     |
| 2010 | 5   | 1   | 6     |
| 2011 | 4   | 2   | 6     |
| 2012 | 4   | 2   | 6     |
| 2013 | 5   | 2   | 7     |
| 2014 | 9   | 3   | 12    |
| 2015 | 1   | 2   | 3     |
| 2016 | 6   | 8   | 14    |
| 2017 | 12  | 5   | 17    |
| 2018 | 2   | 7   | 9     |
| **Total** | **60** | **44** | **104** |

---

2. **Documentation is considered waste.** Generally, documentation is considered waste when it does not contribute to the end product [317, S322]. Documentation is only created if it is required to create the end product, or to raise the quality of the end product. Prause et al. differentiate between documentation for developers and for end-users. Documentation for developers does not contribute to the end product and is therefore neglected. The source code itself is considered the “ultimate documentation”. An example of documentation that does contribute to the end product is a user-manual, for instance [S322]. As a result, design knowledge, reasoning knowledge, as well as knowledge about the problem space are typically not preserved in CSD in any written form.

3. **Productivity is measured by the amount of working software only.** In CSD, productivity is measured by the amount of delivered working software over development time. Beck and other founders of the agile manifesto state that working software is valued over comprehensive documentation [48, 226, S269, S410]. Thus, they emphasize that working code is the ultimate measure of productivity; documentation has value, but its comprehensiveness is less important. Stettina et al. note that documentation is rather seen as a burden, than a (co-)created artifact [S375]. This attitude causes developers to generally consider documentation as counter-productive, which in turns causes knowledge loss.

4. **Documentation is out-of-sync with the software.** In CSD, developers do not keep documentation in sync with the actual software [S290]. This applies to both documentation outside the code such as in Microsoft Word documents and wiki-like tools, but also to source code documentation, e.g. regarding the objectives of methods or their parameters. Especially source code documentation is often outdated because CSD emphasizes the continuous update of code, but not its documentation. This is an issue, as stakeholders lose confidence and trust in the documentation [S167], which makes the sparse documentation even less useful. A lack of up-to-date documentation is particularly problematic in the context of architecture design decisions, as it leads to a loss of rationale behind design choices and considered
alternatives; it thus become increasingly difficult to understand and judge solutions during software evolution [S177].

5. Short-term focus. Producing comprehensive documentation is a resource-intensive task that interferes with other short-term tasks like sketching diagrams or programming. Primarily, the short-term goals of design, programming or maintenance tasks can be achieved without documenting important decisions, documenting rationale, consequences or alternatives [218,S377]. However, this focus on achieving mostly short-term goals has an adverse effect: all the knowledge that is required for those goals disappears over the following iterations with changing context, new objectives and new team members. Nawrocki et al. state that in XP there are three sources of knowledge about the software that are required but are hard to maintain in the long run [S284]: the code, test cases and the memory of the developers.

Despite, the aforementioned challenges, we were also able to observe documentation practices. In this study, a “practice” is defined as an activity that is usually or regularly conducted, e.g. as a habit, tradition, rule, or organizational culture.

1. Non-written and informal communication. In CSD, verbal communication is often used to achieve a mutual understanding between team members (e.g. in [S139]) rather than written documentation. Verbal communication is also one of the twelve principles in the agile manifesto [48], which states that face-to-face communication is both most effective and most efficient within a development team. For agile development in general and XP in particular, Praise et al. state that knowledge is the result of collaboration and is spread by different means [S322], other than written documentation. Often only sketches and informal drawings are used to support the verbal communication. One exception to this rule is requirements, which are typically documented in the format of user stories [98]. The sparse documentation that is deliberately created for documentation purposes is usually created afterwards and describes the state of the software “as is”, rather than the software “to be” [S184]; this has the advantage of being up-to-date.

2. Usage of development artifacts for documentation purposes. Apart from artifacts created solely for the purpose of documentation, some artifacts created as part of the development process can also serve as a type of documentation. Test Driven Development (TDD) and Behavior Driven Development(BDD) [S389], for instance, lead to executable specifications of the software to be built [S28]. Another form of executable documentation is “infrastructure as code”, as mentioned by Callanan et al. [S75]. Infrastructure-as-code refers to any executable description of the infrastructure that is not part of the application itself [194]. This can be achieved with tools like Ansible, or Puppet, for instance.

3. Architecture frameworks
Although architecture knowledge often evaporates in CSD projects, we have seen one particular documentation format, namely architecture frameworks, being used in practice. We briefly describe two examples of frameworks that qualify as architecture frameworks according to the definition in ISO/IEC/IEEE 42010 [200], while specifically addressing concerns of continuous software development. Di Nitto et al. proposed a framework called SQUID (Specification Quality In Devops), an extension of Kruchten’s 4+1 view model [S246] with additional viewpoints for dealing with DevOps-related concerns [S114]: an Operations Viewpoint, a Monitoring Viewpoint, a Deployment Viewpoint, and a Quality Verification Viewpoint. Similarly, the continuous integration and delivery architecture framework (Cinders), described by Stähle and Bosch, is an architecture framework specifically designed to deal with architectural concerns in continuous integration and delivery [S379].

3.3.1. Interpretations of the results
We think that challenges lead to practices, and vice versa, as illustrated in Fig. 8. The first relation is between the challenge of short-term focus (1) that leads to non-written and informal communication. In turn, the non-written and informal communication leads to documentation that is hard to understand (2), and documentation being considered waste when it does not contribute to the end product (3). Using development artifacts for documentation purposes leads to the challenges that productivity is measured by the amount of software only (4) or that documentation is out-of-sync with the software (5). Furthermore, only the practice of architecture frameworks might be considered a practice to contribute to better documentation (see the green box in Fig. 8).

3.3.2. Implications for practitioners
The demand for fast time-to-market leads to fewer artifacts with lower quality. In small teams that are geographically located in one building or one room with long-term employees, informal knowledge is built up in the team. For larger teams, geographically distributed or with changing team members, building up knowledge about the software product and processes might be challenging. A typical practice for documentation is that a whiteboard sketch, and formal API documentation are considered sufficient instead of big upfront documentation with many UML diagrams. Apparently, these informal documentation practices are just enough to start an iteration. At the same time, however, these practices are not sufficient for operations, maintenance, or knowledge transfer. Another candidate approach to overcome these challenges is executable documentation; for TDD, this is a common practice.

3.3.3. Areas for future research
Future research is required to investigate if just enough upfront documentation can be limited to shaping thoughts by using informal whiteboard sketches, together with the codified API documentation. The upfront documentation should be accompanied by design-when-done after an iteration is completed. Anything that can be generated or reverse-engineered is not required to document because it is available anytime. Relevant for operations, maintenance, and knowledge transfer are decisions, considerations about the software product and process, and team organization.

A second area for future research is executable documentation. The practice of TDD, which is one example of executable specification, is a non-intrusive way of documenting requirements as part of the development process. This, however may not be the case for other types of executable specifications. In general, the question how executable documentation can be best produced and consumed by developers is subject to further research.

3.4. Results for RQ2: Tools used in CSD
In RQ2, we investigated, which tools are used in CSD with respect to documentation. The studies we considered for answering this research question are listed in Table 9. The studies from the final result are presented in Fig. 7 and Appendix B. As already discussed, the sparse documentation in CSD is scattered over the entire tool ecosystem, mainly in the form of executable specifications. This concerns up-front documentation (mainly requirements), as well as design, code, and deployment information. Table 11 lists tools typically used in CSD practices for documentation purposes, along with the type of documentation (mark-down, binary, drawings), and what is documented (decisions, annotations, commit messages).

An ideal tool for documentation, according to Cannizzo et al. would support four main practices: status visibility (build tools), extensive automation (i.e. automating as much as possible), effective communication (collaboration and communication tools, metrics tools), and tools for immediate feedback (test tools) [S78]. Other researchers
1. **Development tools**

   (a) **Requirements management tools** (e.g. Blueprint, RequirementONE)
   
   The documentation information includes stakeholder concerns, risks, constraints and context formulated as specifications: these are typically codified instructions, sufficient for developers to start an iteration. Such specifications range from very informal and abstract (e.g. user stories and high-level use-case descriptions) to formal and concrete (e.g. detailed use-case descriptions with pre- and post-conditions, or Cucumber in combination with Gerkin).

   (b) **IDEs** (e.g. IntelliJ, Eclipse, Cloud9)
   
   The documentation information include the source code of the software, often annotated with comments, meant for developers to understand the code. The annotations in the code are also used for the automated generation of documentation for APIs.

   (c) **Agile Management tools** (e.g. Active.collab, Agile bench, JIRA)
   
   Agile management tools are used to support developers in applying agile methods like Scrum, Lean, or Kanban. The tools typically capture information about requirements (e.g. user stories or use-cases), tasks, progress (e.g. burn-down charts), planned and achieved goals for iterations, development speed (e.g. team velocity), and the provide traceability between requirements, tasks, and code (e.g. a JIRA board on which tasks fall under use-cases and are linked to source-code using Git branches and pull requests).

   (d) **Development Analytics tools** (e.g. SonarQube, Metrixware)
   
   The documentation information includes metrics, as well as actionable data that can be used by all roles in the software development including managers, developers, and maintainers. Typical examples of actionable data are quality measurements in SonarQube (a static source code analysis tool) along with concrete suggestions for fixing the quality rule violations.

   (e) **Repositories/Development Communities** (e.g. GitLab, StackShare, StackOverflow)
   
   The documentation information that repositories hold are the source code ready for review, or deployment and commit messages where every push to the repository (ideally) includes a meaningful comment on the changes. In development communities (e.g. Stack Overflow of GitHub), the information contains questions and answers for software development topics. As such, it can be considered as a knowledge base.

2. **Test tools**

   (a) **Performance/Load / Stress test tools** (e.g. JMeter and SmartStorm)
   
   These types of test codify specific QoS requirements (typically quality attribute requirements in the categories of performance efficiency, reliability, and security (see ISO/IEC/IEEE 25010 [198]).

   (b) **Functional Automation, Virtualization tools** (e.g. Cucumber and Appium)
   
   The documentation information includes: the test object (method, component, API, UI); the test strategy with guiding values, principles, and objectives for stakeholders; and test tactics, and types with techniques (e.g. destructive/UI tests), processes (e.g. CI/CD) and approaches (e.g. manual/automated testing) applied to specific test tasks.

   (c) **Continuous testing tools** (e.g. test.ai, buildbot)
   
   Continuous testing is a software testing type that involves a process of testing early, often, everywhere, and automatically (to the best possible extent). The objective of continuous testing is to find defects and support immediate response to the defects during the whole development cycle, including requirement specification, development, and maintenance. The documentation information refers
Documentation tools in CSD retrieved from the studies for RQ2.

| Tool, Excel, Powerpoint etc. | Used for reports and intended for long term usage | Binary files | Decisions, drawing, sketches, pictures of whiteboard drawings | [S78], [S364], [S419], [S56], [S57], [S58], [S88], [S156], [S279], [S316], [S322], [S330], [S375], [S377], [S402], [S406], [S411], [S413], [S12], [S38], [S49], [S78], [S99], [S115], [S126], [S164], [S175], [S184], [S198], [S197], [S199], [S320], [S322], [S332], [S349], [S359], [S361], [S389], [S424] |
| Wiki | Wiki-likes, Confluence | Short descriptions in the format of the tool, e.g. xml, html | Tasks, how-to’s, SRS, SAD, SDD, info on PoCs, Prototypes, Releases | [S56], [S57], [S58], [S86], [S156], [S279], [S316], [S322], [S330], [S377], [S402], [S406], [S411], [S413], [S12], [S38], [S78], [S99], [S115], [S126], [S154], [S198], [S197], [S199], [S320], [S322], [S332], [S349], [S359], [S361], [S389], [S424], [S425], [S175], [S184], [S220], [S322] |
| Source control | Tools for Git, Mercurial, SVN | Source-code, annotations, commit messages | Source-code, commit messages | [S375], [S413], [S175], [S340], [S58], [S410] |
| Scripts | Executable lines of code to run tasks | Human readable text | Infrastructure-as-code: Tests, integration, and deployment including installation, configuration, data import, and security | [S169], [S90], [S377], [S115], [S175], [S402], [S389], [S255], [S410] |
| Markdown editors | Light weight text editor, often used without editor but edited directly | Human readable text | Anything that can be documented in Word, Excel, Powerpoint, or Wikis including configuration files | [S413], [S237] |
| Verbal communication | The proverbial water-cooler conversation. | Face-to-face | No document, knowledge remains tacit | [S184], [S375], [S175], [S389], [S181], [S355], [S416] |

3. Deployment tools
   (a) App Automation tools (e.g. Ansible, Puppet, Chef)
   The documentation information includes instructions for installation, updates and configuration of software, the import of data, and hardening of systems. This information is typically defined in CI/CD scripts. This type of scripts assists in the automation of (complex) IT tasks into repeatable playbooks. Although the scripts contain information for a range of tasks, they are typically configured for a single task such as installation or phase such as test.
   (b) CI/CD (e.g. CircleCI, Jenkins)
   The documentation information includes the relation between single tasks and the results of executing these tasks. The automation considers the execution of single tasks from App Automation into a set of scripts for multiple tasks (e.g. installation, configuration, import, hardening) and for multiple stages (e.g. development, test, integration, deployment) whether on premise or in the cloud. The test results show developers or release managers the sanity of the builds in a comprehensive visual overview.

4. Service execution tools
   (a) Cloud/Container Orchestration/Management (e.g. Docker, Mesos)
   The documentation information includes metrics about non-functional requirements such as, but not limited to availability and reliability. It includes also installation and configuration scripts for the software as well as scripts for automated up- or down scaling. This can refer to a single container as well as the orchestration of containers. Infrastructure monitoring tools provide visibility of the complete infrastructure and allow for troubleshooting and resource optimization.

5. Monitoring tools
   (a) Monitoring & Management (e.g. DataDog, RunDeck)
   In CSD, a lot of processes, and (supporting) applications run at the same time which can lead to a chaotic software development ecosystem as well as a hard to manage deployment pipeline and production environment. The monitoring and management tools support the team to have control of processes and software products. The documentation information captured in the tools contains desired and actual quality-of-service parameters, as well as standard operating procedures for incidents.

6. Security tools
   (a) Container Security (e.g. AppArmor, Cloud Insights)
   The documentation information for container security involves the build scripts for the container and the additional security policies (such as non-root user, application isolation, and authentication/authorization).
   (b) Application Security (e.g. Threat Stack, HyTrust)
Containerized applications typically make use of a microservice architecture. The information comprises infrastructural security, information on the security aspects (confidentiality, integrity, non-repudiation, accountability and authenticity), information on the distribution of the multiple applications in multiple containers including functionality, data, subsystems, and APIs.

(c) DevSecOps (e.g. Cigital, CheckMarx)
DevSecOps refers to the processes and practices to merge the security that is used in development process into processes in operations and vice versa with the purpose of faster deployment with security measures in place. The documentation information includes authentication, and authorization with roles for users (where applications are also defined as a user that needs to be authenticated to obtain authorization). It also includes information about technical (software and hardware) security, as well as test procedures to force and validate security measures.

7. API management tools and directories
(a) API management tools (e.g. Smartbear, Mashape, RapidAPI, OpenAPI)
These tools document the definition of the resource description, endpoints with methods, parameters, often a request and response example, and sometimes a playground for testing the API.
(b) API directories (e.g. ProgrammableWeb)
These provide a directory of external APIs that include a description, documentation for developers, SDK, “How to” instructions, (optional) libraries, and information from the community and thus also capture general documentation information about APIs and underlying technologies.

As shown in the list above, documentation information is scattered throughout an entire eco-system of tools rather than being provided in a single self-contained document. As a consequence of the scattering, stakeholders who need to understand the vision and long term goals, architecture decisions, risks, constraints, interface definitions, deployment instructions etc., need to dig into the entire tool-stack [S372, S167]. There is no single source of truth, but there are many sources of truth, each holding information on a relevant part of the software product [328].

3.4.1. Interpretations of the results
There are many tools used in CSD, for every phase (e.g. design, implementation, and testing), as well as every activity (e.g. drawing, collaborating, writing, and constructing). The amount of structure of the information is strongly related to the tool. Some information is easy to capture and easy for human communication such as whiteboard sketches or conversations in chats. At the same time, these types of information are hard for automatic processing. Source code on the other hand, can be automatically processed.

3.4.2. Implications for practitioners
For the construction of the software product, the tools support the developers. As such, the tooling has a positive effect on the productivity. However, with the increase of the number of tools, information about the software product, processes and organization is distributed across these tools.

3.4.3. Areas for future research
A candidate area for future research could be to organize the documentation into yellow pages (wiki, git, markdown) that contains references to relevant documentation for designated stakeholders. For instance PowerPoint slides for conveying ideas about the software product, design documentation for developers and infrastructure-as-code for operations.

4. Discussion
In this section, we interpret our results and provide implications for practitioners and researchers. To begin with the interpretation of the results, the software engineering discipline has always been struggling with documentation. Parnas, for instance, reported back in 1994, that documentation – if written at all – is usually poorly organized, incomplete and imprecise [306]. The human factors that caused this problem back then, are – to the same extent – responsible for the issues reported over documentation in Continuous Software Development today. The difference is that missing or poor documentation was traditionally seen as the result of negligence or even misconduct of developers; in continuous software development it is deliberately promoted to a desired behavior. In other words, CSD puts many obstacles in the way of properly documenting the different aspects of created knowledge (e.g. considering documentation as waste, having a short-term focus, measuring productivity through working software only).

In CSD, with a few mentioned exceptions like documenting requirements, dedicated documentation (i.e. documentation that does not serve as development artifact also) is informal (e.g. white board sketches) and needs to be supported by face-to-face communication. This may not be ideal, but we argue that it can be an effective and efficient approach to support the design reasoning process. However, it cannot effectively preserve knowledge and thus not serve as documentation; this is not a surprise as informal artifacts are not created for the purpose of documenting, but for the purpose of supporting a design discussion.

Knowledge-preserving documentation that stands on its own requires a certain level of formalism and needs to be created for the purpose of describing something unambiguously. Such documentation is very rarely created in CSD projects. Thus, in our opinion, the documentation practices in CSD – or lack thereof – do not contribute to solving the traditional problems related to knowledge loss and missing information during maintenance activities. Unfortunately, we have not seen evidence of new or emerging practices that can alleviate this problem.

Although the results we found regarding dedicated documentation practices in CSD are sobering, there is also good news. With the rise of Lean, Agile and DevOps projects, we observe a drastic boost in tool-ecosystems, which mainly stems from the goal to automate software-related processes as much as possible. This also enables new ways of thinking about documentation. The specifications required for automating processes (we refer to them as executable specifications), at the same time serve as documentation of the process. This essentially urges us to refine Robert Martin’s statement that the truth can only be found in the code: now the truth can also be found in test scripts, provisioning scripts, build pipeline configurations, and cloud platform configurations, to name just a few.

4.1. Characteristics of executable documentation
Executable specifications have a lot of potential for serving as documentation in CSD. Their characteristics are in line with the principles of CSD, and at the same time address the previously mentioned traditional problems that come with missing documentation. We highlight the following characteristics of executable documentation that require further research.

4.1.1. Types of executable documentation
Types of executable documentation include: requirements, such as “Specification by Example” [152], test-cases resulting from Domain-Driven-Design [404] or TDD [S292,S334,S175,S28], database scripts with Data Definition Language (DDL) and Data Manipulation Language (DML), deployment scripts with Ansible [S389] or infrastructure-as-code [72].
4.1.2. Executable documentation is never out-of-sync

Executable documentation is never out-of-sync, it’s evolution is naturally connected to the evolution of the other parts of the software. Executable documentation does not just describe the software, but it is part of the software.

4.1.3. Executable documentation can be tested

Executable documentation can be tested. If it does not lead to the desired results, then something must be wrong. In that respect, executable documentation is just like source-code.

4.1.4. Executable documentation is non-intrusive

The process of creating executable documentation is not intrusive, i.e. developers do not stop their work to take care of documentation; coding and documenting are part of the same task.

In future research, these items will be investigated. Questions remain, for example, how can software development teams use such executable specifications? This could include a considerable amount of unstructured (and unrelated) data.

4.2. Implications for practitioners

In the following, we present some implications for practitioners who want to benefit from the potential of CSD to document the created knowledge.

4.2.1. Tools, tool-stacks, and software development ecosystems

Support your entire development process by a tool-chain that seamlessly supports all activities in the process. Eliminate manual or interactive steps in the development process to the greatest possible extent. Manuals for developers describing process steps to follow (or the need for such manuals) should be considered as bad smells [193] that should be transferred to executable specifications interpreted by tools. Executable specifications are always up-to-date and at the same time document processes in an unambiguous way that can be interpreted by both machines and humans.

4.2.2. Informal sketches

Use informal sketches (that are minimally intrusive) to support your design reasoning process and discussions with team members. The reasoning process and discussions ultimately lead to decisions that are implemented (e.g. in source-code or executable specifications). Consider briefly documenting the rationale behind those decisions that may not be obvious to other stakeholders (including future developers). Examples of obvious decisions are choices of tools or combinations of tools that are very popular for certain purposes, e.g. the combination of Elasticsearch, Logstash, and Kibana for distributed logging and analytics.

4.2.3. Use of version control

Keep everything under version control. Use project management tools or wikis as a central entry point for all information related to the project; otherwise, stakeholders may easily get lost in the great amount of project locations, tools and URLs. Also consider providing high-level overviews of the designed sub-systems, and link the respective executable specifications to the sub-systems to facilitate access for stakeholders.

4.3. Future research

In terms of research, the results of this mapping study have shown that documentation in CSD, has not yet gained the required attention by the research community. In the following, we describe three areas for future research:

1. The individual tools in a CSD ecosystem are mostly created separately, thus having limited interoperability. However, the combination of information from different tools can be “more than the sum of its parts”, i.e. it can provide insights that capture a greater part of the system and life cycle. Thus research is required to establish traceability links between the different types of tools and intelligently combine information from different kinds of executable documentation in dashboards.

2. Traditional architecture documentation approaches seem to come in direct conflict with the identified documentation challenges in CSD. Research is required to develop architecture documentation and specification approaches that integrate seamlessly in CSD-practices. For example, architecture frameworks could be developed that tap the potential of executable specifications, while preserving design rationale, explaining architecture to stakeholders, linking design decisions to concerns and architectural requirements and providing an informational basis for architectural evaluation.

3. The high degree of automation offers rich sources of information that can be mined using Mining Software Repository techniques, or in general Data Science. Examples of questions that could be addressed using such approaches in CSD are:
   - What is the current technical debt in source code, testing, requirements or other types?
   - What is the optimal point in time for refactoring a specific sub-system?

5. Threats to validity

We use the framework of Amapzoglou et al. [21] that describes potentials threats to validity for secondary studies. Specifically this framework classifies threats to validity in three categories, as illustrated in Table 12.

5.1. Study selection validity

Regarding the selection of digital libraries, we have to a large extent addressed this by including the most used digital libraries in this area (which are also commonly used in secondary studies in software engineering). The construction of the search string may lead to yielding too many primary studies or missing relevant studies. We mitigated this threat by calibrating the search string through the quasi-gold standard. Specifically the QGS was used to assess the performance of the search string and refine it until all primary studies of the QGS were returned from the search string. The QGS itself was built using the snowballing technique guidelines as proposed by Wolhin [427].

Furthermore, we have mitigated the risk of an arbitrary starting year, because it was related to the year of the publication of the Agile Manifesto. With this decision we excluded a historic overview of consecutive concepts that lead to the Agile Manifesto; however, we did not aim at such a historic overview but a systematic classification and thematic analysis of literature.

7 In August 2019, “executable documentation” had the following results: ACM: 9; Google Scholar: 207; IEEE: 2; ScienceDirect: 15; SpringerLink: 35; Web of Science: 3
Table 12

| Category                | Description                                                                 |
|-------------------------|-----------------------------------------------------------------------------|
| Study Selection Validity| These threats can be identified in the initial search process where the set of candidate papers for primary studies is selected and the study filtering where the final set of primary studies is determined. Typical examples are the selection of digital libraries, search string construction and study selection bias. |
| Data Validity           | These threats can be identified in the data extraction phase (a data set is populated) and data analysis phase (the data set is qualitatively or quantitatively analyzed). Typical examples include data collection bias and publication bias. |
| Research Validity       | These threats can be identified over the whole mapping study and concern the design of the research. Typical examples are generalizability, and coverage of research questions. |

The threat for non-English papers was not mitigated; these papers were excluded. We did however address the threat of studies not being accessible: we made sure we could access all studies. The threat of duplicate articles was mitigated by filtering on the Document Object Identifier. If a study appeared in multiple digital libraries, then the publishers' digital library was used and the duplicate was ignored. We excluded gray literature and included only studies from peer reviewed journals, conferences or workshops to have more rigor. Finally, the potential bias of study inclusion/exclusion was mitigated by discussion among the authors and accordingly revising the inclusion/exclusion criteria.

5.2. Data validity

The risk of retrieving a small sample was mitigated by constructing a search string that could zoom in from a domain with over approximately 35,000 studies to finally about 200 relevant papers to answer the research questions. The threat of choosing the correct variables to be extracted was addressed through extensive discussions between the authors. The threat of publication bias (the majority of identified primary studies coming from specific venues) was mitigated by using snowballing. Furthermore, we addressed the threat of inadequate validity of primary studies through the inclusion criteria by only looking at peer reviewed venues. The threat of biasing the classification schema is mitigated by going through several iterations to refine the RQs, and redefining the search string and the analysis process. The threat of researchers' bias was partially mitigated by doing the analysis with multiple researchers where research and review were different roles, and by using a combination of manual and automated search.

5.3. Research validity

The threat of repeatability is mitigated by meticulously documenting the study protocol. In addition, the retrieved studies, search strings and data are all available on https://theotheunissen.nl/SMS. The threat of the chosen research method bias is mitigated by extensive discussions among the authors and the rationale of our decision is clearly described in the study design section. Furthermore, the authors have also discussed in multiple iterations the choice and coverage of the research questions. Regarding the generalizability of our results, they are only applicable within the scope of documentation in continuous software development.

6. Conclusions

We conducted a systematic mapping study to investigate the documentation practices and challenges, as well as the tooling used in continuous software development (CSD). The study has provided an overview of the relevant primary studies and has listed a number of challenges, practices, and tools that pertain to documentation in CSD.

Section 3.3 elaborates on our findings regarding documentation challenges and practices (RQ1). The challenges include: informal documentation is hard to understand, documentation is considered waste when it does not contribute to the end product, productivity is limited to the measured amount of working software, documentation is easily out-of-sync with the actual code, and there is short term focus. The practices include: a significant amount of communications happens verbally and informally; there is a positive usage of development artifacts for documentation purposes, such as TDD; and the use of architecture frameworks might positively influence documentation quality.

Section 3.4 discusses an increasing number of tool categories and tools that can be used to support development, operations, and maintenance in CSD (RQ2). CSD is a high-paced evolving and dynamic environment; without tools, development and deployment would not be possible. An interesting side-effect of the tooling that has not been adequately researched yet, is that with every tool that is being used, knowledge about the piece of software is stored, maintained and transferred as well. For example, commits in a repository describe the changes of the source code and test scripts in a test tool describe the required outcomes of software. Knowledge about the software is scattered throughout all the tools in a software ecosystem. There is not a single source of truth, but there are a lot of sources of truth, each holding a small piece of knowledge. The discovery of these pieces of knowledge has not been investigated and it could be interesting to do further research on how to locate and combine these information sources.

Finally, we identified several implications for practitioners regarding the use of executable specifications in combination with a high degree of automation. Additionally, we found that architecture frameworks streamlined for use in CSD and dashboards combining information from the entire development tool chain are important areas for future research.

CRediT authorship contribution statement

Theo Theunissen: Writing – original draft, Writing – review & editing. Uwe van Heesch: Writing – review & editing, Supervision. Paris Avgeriou: Writing – review & editing, Supervision.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2021.106733.

Appendix A. List of studies

Please see Further reading section.

Appendix B. Studies in the bubble chart

This section lists all studies visualized in the bubble chart from Fig. 7. For each table, we present the number of studies along with the respective references to these studies. Appendix A contains the complete list of papers with full bibliographical details.
| Category | A: Method, Documentation life cycle |
|----------|-----------------------------------|
| Papers   | [S340], [S284], [S23], [S17], [S206], [S316], [S346], [S376] |
| Found    | 8 |

| Category | B: Method, Documentation subjects: decisions |
|----------|---------------------------------------------|
| Papers   | [S340], [S55], [S17], [S206], [S376] |
| Found    | 5 |

| Category | C: Method, Documentation subjects: architecture |
|----------|-----------------------------------------------|
| Papers   | [S340], [S355], [S23], [S55], [S265], [S89], [S56], [S389], [S17], [S181], [S316], [S90], [S377], [S346], [S376] |
| Found    | 15 |

| Category | D: Method, Tool |
|----------|-----------------|
| Papers   | [S340], [S55], [S56], [S181] |
| Found    | 4 |

| Category | E: Method, Documentation subjects: source-code |
|----------|-----------------------------------------------|
| Papers   | [S355], [S55], [S17], [S181], [S377], [S376] |
| Found    | 6 |

| Category | F: Metric, Documentation subjects: source-code |
|----------|-----------------------------------------------|
| Papers   | [S372], [S413], [S156], [S406], [S274], [S112], [S364] |
| Found    | 7 |

| Category | G: Metric, Documentation subjects: decisions |
|----------|---------------------------------------------|
| Papers   | [S372], [S156], [S406], [S274], [S112] |
| Found    | 5 |

| Category | H: Metric, Documentation subjects: architecture |
|----------|-----------------------------------------------|
| Papers   | [S372], [S413], [S156], [S406], [S274], [S217], [S407], [S58], [S112], [S316], [S364], [S224] |
| Found    | 12 |

| Category | I: Metric, Tool |
|----------|-----------------|
| Papers   | [S372] |
| Found    | 1 |

| Category | J: Metric, Documentation life cycle |
|----------|-----------------------------------|
| Papers   | [S413], [S156], [S406], [S217], [S407], [S58], [S316], [S68], [S364] |
| Found    | 9 |

| Category | K: Metric, Documentation subjects: autogenerated |
|----------|-----------------------------------------------|
| Papers   | [S112] |
| Found    | 1 |

| Category | L: Model, Documentation subjects: architecture |
|----------|---------------------------------------------|
| Papers   | [S522], [S274] |
| Found    | 2 |

| Category | M: Model, Documentation subjects: source-code |
|----------|-----------------------------------------------|
| Papers   | [S274] |
| Found    | 1 |

| Category | N: Model, Documentation subjects: decisions |
|----------|---------------------------------------------|
| Papers   | [S274] |
| Found    | 1 |

| Category | O: Other, Documentation subjects: source-code |
|----------|-----------------------------------------------|
| Papers   | [S389] |
| Found    | 1 |

| Category | P: Other, Documentation life cycle |
|----------|-----------------------------------|
| Papers   | [S389] |
| Found    | 1 |

| Category | Q: Other, Documentation subjects: decisions |
|----------|---------------------------------------------|
| Papers   | [S389] |
| Found    | 1 |

| Category | R: Other, Documentation subjects: architecture |
|----------|-----------------------------------------------|
| Papers   | [S389], [S159] |
| Found    | 2 |

| Category | S: Process, Documentation subjects: source-code |
|----------|-----------------------------------------------|
| Papers   | [S375], [S406] |
| Found    | 2 |

| Category | T: Process, Documentation life cycle |
|----------|-----------------------------------|
| Papers   | [S375], [S406], [S410] |
| Found    | 3 |

| Category | U: Process, Documentation subjects: architecture |
|----------|-----------------------------------------------|
| Papers   | [S375], [S406], [S57], [S410], [S90] |
| Found    | 5 |
| Category | Subcategory | Subject Areas | Papers Found |
|----------|-------------|---------------|--------------|
| V: Process, Documentation subjects: decisions |
| Papers Found | [S406] |
| 1 |
| W: Process, Documentation subjects: autogenerated |
| Papers Found | [S57] |
| 1 |
| X: Process, Tool |
| Papers Found | [S57], [S410] |
| 2 |
| Y: Evaluation Research, Documentation life cycle |
| Papers Found | [S340], [S413], [S156], [S284], [S406], [S217], [S407], [S58], [S206], [S316], [S68], [S364] |
| 12 |
| Z: Evaluation Research, Documentation subjects: decisions |
| Papers Found | [S340], [S372], [S156], [S406], [S274], [S112], [S206] |
| 7 |
| a: Evaluation Research, Documentation subjects: architecture |
| Papers Found | [S340], [S372], [S413], [S156], [S406], [S265], [S274], [S89], [S217], [S407], [S58], [S112], [S316], [S159], [S364], [S224] |
| 16 |
| b: Evaluation Research, Tool |
| Papers Found | [S340], [S372] |
| 2 |
| c: Evaluation Research, Documentation subjects: source-code |
| Papers Found | [S372], [S413], [S156], [S406], [S274], [S112], [S364] |
| 7 |
| d: Evaluation Research, Documentation subjects: autogenerated |
| Papers Found | [S112] |
| 1 |
| e: Experience Report, Documentation subjects: source-code |
| Papers Found | [S355], [S389], [S17] |
| 3 |
| f: Experience Report, Documentation subjects: architecture |
| Papers Found | [S355], [S389], [S17] |
| 3 |
| g: Experience Report, Documentation life cycle |
| Papers Found | [S389], [S17] |
| 2 |
| h: Experience Report, Documentation subjects: decisions |
| Papers Found | [S389], [S17] |
| 2 |
| i: Solution Proposal, Documentation life cycle |
| Papers Found | [S340], [S375], [S23], [S410], [S255], [S346], [S376] |
| 7 |
| j: Solution Proposal, Documentation subjects: decisions |
| Papers Found | [S340], [S372], [S55], [S274], [S376] |
| 5 |
| k: Solution Proposal, Documentation subjects: architecture |
| Papers Found | [S340], [S372], [S375], [S23], [S322], [S55], [S57], [S274], [S89], [S56], [S389], [S181], [S410], [S90], [S377], [S346], [S376] |
| 17 |
| l: Solution Proposal, Tool |
| Papers Found | [S340], [S372], [S55], [S376] |
| 7 |
| m: Solution Proposal, Documentation subjects: source-code |
| Papers Found | [S372], [S375], [S55], [S274], [S181], [S377], [S376] |
| 7 |
| n: Solution Proposal, Documentation subjects: autogenerated |
| Papers Found | [S57], [S255] |
| 2 |
I am conducting a systematic mapping study to research the literature on documentation, tooling and existing frameworks in continuous software development (or: agile, lean, DevOps, CI/CD).

Your input as an academic researcher or industry practitioner in this area is appreciated.

**BACKGROUND**

Documentation of software architecture, design, development and operations have a long tradition of both storing knowledge and communicating decisions. At the same time, documentation is a tedious, time-consuming task that is usually reduced to a minimum in continuous software development processes such as lean, agile and DevOps. Continuous software development has been discussed in. The focus of this mapping study is on documentation practices in continuous software development processes such as lean, agile and DevOps. These development processes are the de facto standards in many small startups as well as in large enterprises.

A mapping study for documentation in continuous software development processes does not exist. Because documentation in these processes deviates from textbook standards that are taught during education, and there is no prescribed standard but just a practice of documentation, this study is relevant for both researchers, practitioners, and educators. This mapping study is an assessment of existing literature on development processes, documentation methods, and frameworks --- including tools. It aims to find and classify the primary studies in this specific topic area.

**RESEARCH QUESTIONS**

RQ1: What studies exist on documentation practices in continuous software development (CSD)?

Rationale: Documentation plays a major role in preserving knowledge and communicating decisions in software architecture and technical implementation. At the same time, documentation practices in CSD are lacking. With this research question, an overview of documentation methods will be presented.

RQ2: What studies exist on tools used in CSD?

Rationale: In the community of practice for continuous software development, tools are used to speed up development, monitor quality, and automatic deployment. This documentation is not exported to a central repository but kept with the tool, e.g. Jira, Github. The focus is primarily on tools that are described in the literature but will be extended to tools that are actually used for architects and developers.

1. *A Study of Enabling Factors for Rapid Fielding: Combined Practices to Balance Speed and Stability*, Stephany Bellomo and Robert L. Nord and Ipek Ozkaya, 2013
2. *A Study of the Documentation Essential to Software Maintenance*, Sergio Cozzetti B. de Souza and Nicolas Anquetil and Kathia M. de Oliveira, 2005
3. *Agile Architecture Interactions*, J. Madison, 2010
4. *Agile Architecture IS Possible You First Have to Believe!*, M. Isham, 2008
5. *Agile Documentation, Anyone?*, B. Selic, 2009
6. *Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects*, Rüping, Andreas, 2005
7. *Agile in Distress: Architecture to the Rescue*, Robert L. Nord and Ipek Ozkaya, 2014
8. *Agile software development: the business of innovation*, J. Highsmith; A. Cockburn, 2001
9. *Agility and Architecture: Can They Coexist?*, P. Abrahamsson; M. A. Babar; P. Kruchten, 2010
10. *Analysis and Management of Architectural Dependencies in Iterative Release Planning*, Brown, Nanette and Nord, Robert L. and Ozkaya, Ipek and Pais, Manuel, 2011
11. *Architecting for Large-Scale Agile Development: A Risk-Driven Approach*, Ipek Ozkaya, Michael J. Gagliardi, Robert Nord, 2013
12. *Architecting for sustainable software delivery*, Koonitz, Ronald J and Nord, Robert L, 2012
13. *Architecting in a Complex World: Eliciting and Specifying Quality Attribute Requirements*, Wojcik, Rob, 2013
14. *Architects as Service Providers*, R. Faber, 2010
15. *Beyond Scrum + XP: Agile Architecture Practice*, Ozkaya. Ipek, Robert L. Nord, Stephany Bellomo, and Heidi Brayer, 2013
16. *Climbing the “Stairway to Heaven” – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development Towards Continuous Deployment of Software*, Olsson, Helena Holmstrom and Alahyari, Hiva and Bosch, Jan, 2012
17. *Combining architecture-centric engineering with the team software process*, Nord, Robert I. and McHale, James and Bachmann, Felix, 2010
18. *DevOps: A Software Architect's Perspective*, Bass, Len and Weber, Ingo and Zhu, Liming, 2015
19. *Elaboration on an integrated architecture and requirement practice: Prototyping with quality attribute focus*, S. Bellomo; R. L. Nord; I. Ozkaya, 2013
Wen Lung Tsai, Chung Yang Chen, Chun Shuo Chen, Snowman: Agile development on software traceability: A mapping study, J. Softw. Evol. Process 33 (2) (2020) http://dx.doi.org/10.1002/smr.2294.

Edgewall Software, Trac, 2012, 1https://trac.edgewall.org/.

Rini van Solingen, Vic Basili, Gianluigi Caldiera, Dieter Rombach, Goal question metric (GQM) approach, Encycl. Softw. Eng. (2002) http://dx.doi.org/10.1002/9780470829599.a4202.8.

Shevta Sounndarajan, James D. Arthur, A soft-structured agile framework for larger scale systems development, in: Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems, 2009, pp. 187–195, http://dx.doi.org/10.1109/ECBS.2009.21.

Matt Stephens, The Case Against Extreme Programming, APress, 2002, pp. 1–20, http://dx.doi.org/10.1007/978-1-4302-0610-5.

Karsten Stocker, Hironori Washizaki, Yoshiaki Fukazawa, Closing the gap between test and code documentation, in: 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2017, pp. 304–308, http://dx.doi.org/10.1109/ICSTW.2017.80775.6.

Antony Tang, Jun Han, Pin Chen, MENDELEY_ID Software Engineering Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings 10, Vol. 9839 LNCS, Springer, Cham, 2016, pp. 265–273, http://dx.doi.org/10.1007/978-3-319-48992-6_20.

Antony Tang, Junji Zhi, Vahid Garousi-Yusifolo, Bo Sun, Golara Garousi, Shawn Shahnewaz, He Zhang, Muhammad Ali Babar, Paolo Tell, Identifying relevant studies in complex software engineering domain: A systematic search process, J. Syst. Softw. (ISSN: 01641212) 89 (7) (2014) 148–59, http://dx.doi.org/10.1016/j.jss.2013.08.032.

Antony Tang, Jun Han, Pin Chen, MENDELEY_ID Software Engineering Conference, 2004. 11th Asia-Pacific, Technical Report: SUTTI-TR2004-01: A Comparative Analysis of Architecture Frameworks, Swinburne University of Technology, 2004, pp. 640–647.

Antony Tang, Peng Liang, Hans Van Vliet, Software architecture documentation: The road ahead, in: 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, IEEE, 2011, pp. 252–255.

T Teena, A H Norta, H Rootsi, A H Norta, H Rootsi, K Taveter, I Vorontsova, Enhancing requirements engineering in agile methodologies by agent-oriented goal models: Two empirical case studies, in: 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), 2017, pp. 268–275, http://dx.doi.org/10.1109/REW.2017.24.

Theo Thränissen, Uwe van Heesch, The disappearance of technical specifications in web and mobile applications, in: Software Architecture: 10th European Conference, ECSA 2016, Copenhagen, Denmark, November 28–December 2, 2016, Proceedings 10, Vol. 9839 LNCS, Springer, Cham, 2016, pp. 265–273, http://dx.doi.org/10.1007/978-3-319-48992-6_20.

Theo Thränissen, Stijn Hoppenbrouwers, Sietse Overbeek, Theo Tenso, A H Norta, H Rootsi, K Taveter, I Vorontsova, Enhancing requirements engineering in agile methodologies by agent-oriented goal models: Two empirical case studies, in: 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), 2017, pp. 268–275, http://dx.doi.org/10.1109/REW.2017.24.

Guenther Ruhe, Cost, benefits and quality of software development documentation, in: Proceedings of the 23rd International Conference on Enterprise Information Systems, SCITEPRESS - Science and Technology Publications, 2021, http://dx.doi.org/10.5220/001036901530164.

Roel Wieringa, Neil Maiden, Nancy Mead, Colette Rolland, Requirements engineering: A classification of development approaches, J. Syst. Softw. (ISSN: 01641212) 111 (2016) 157–184, http://dx.doi.org/10.1016/j.jss.2015.09.028.

Robert Yin, Case Study Research: Design and Methods, 3rd Edition (Applied Social Research Methods, Vol. 5), Third ed., Sage Publications, Inc., 2002, http://dx.doi.org/10.1007/978-1-4612-6029-8.

He Zhang, Muhammad Ali Babar, Paolo Tell, Identifying relevant studies in complex software engineering domain: A systematic search process, J. Syst. Softw. (ISSN: 01641212) 89 (7) (2014) 148–59, http://dx.doi.org/10.1016/j.jss.2013.08.032.

Jungji Zhi, Vahid Garousi-Yusifolo, Yu Su, Somayajulu Goutham, Shauver Shahanwaz, Guenther Ruhe, Cost, benefits and quality of software development documentation: A systematic mapping, J. Syst. Softw. (ISSN: 01641212) 99 (2015) 175–198, http://dx.doi.org/10.1016/j.jss.2014.09.042.

Doraj Si, Mei Zhou, Yuan Jin, Qiang Liu, Hua Zhang, A map of threats to validity of systematic literature reviews in software engineering, in: Proceedings Asia-Pacific Software Engineering Conference, ASEPC, 2017, pp. 153–160, http://dx.doi.org/10.1109/ASEPC.2016.031.

Michał mia, Jacek Bojarski, Wiktor Nowakowski, Tomasz Straszak, Scenario construction tool based on extended UML metamodel, in: Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems, in: MoDELS'05, Springer-Verlag, 2005, pp. 414–429, http://dx.doi.org/10.1007/1175432_3.

Darja Šmitte, Pär J. Ågerfeldt, Nils Brede Moe, Agility Across Time and Space: Implementing Agile Methods in Global Software Projects, 2010, pp. 1–341, http://dx.doi.org/10.1007/978-3-540-62442-6.

Further reading

[88] Ademar Aguiar, Tutorial on agile documentation with wikis, in: Proceedings of the 5th International Symposium on Wikis and Open Collaboration - Wikisym '09, in: Wikisymp '09, ACM, New York, NY, USA, 2009, p. 1, http://dx.doi.org/10.1145/1641309.1641365.

[211] Farzaneh Akbari, Sayed Mehran Sharafi, A review to the usage of concepts of software architecture in agile methods, in: 2012 International Symposium on Instrumentation & Measurement, Sensor Network and Automation (IM- MEASNET), Vol. 2, IEEE, 2012, pp. 2601268.

[212] Suha Akma, Elif Berru Aksyuek, Onur Kaynak, Alm tool infrastructure with a focus on DevOps culture, in: European Conference on Software Process Improvement, Springer, 2018, pp. 291–303, http://dx.doi.org/10.1007/978-3-319-97925-0_24.
