Omission of quality software development practices : a systematic literature review.

Author(s): Ghanbari, Hadi; Vartiainen, Tero; Siponen, Mikko

Title: Omission of quality software development practices : a systematic literature review.

Year: 2018

Version: Accepted manuscript

Please cite the original version:

Ghanbari, H., Vartiainen, T., & Siponen, M., (2018). Omission of quality software development practices : a systematic literature review. *ACM computing surveys* 51(2), 1–27. https://doi.org/10.1145/3177746

Omission of Quality Software Development Practices: A Systematic Literature Review[1]

HADI GHANBARI , Faculty of Information Technology, University of Jyvaskyla, Jyväskylä, Finland
TERO VARTIAINEN , Department of Computer Science, University of Vaasa, Vaasa, Finland
MIKKO SIPONEN , Faculty of Information Technology, University of Jyvaskyla, Jyväskylä, Finland

Abstract

Software deficiencies are minimized by utilizing recommended software development and quality assurance practices. However, these recommended practices (i.e., quality practices) become ineffective if software professionals purposefully ignore them. Conducting a systematic literature review (n=4838), we discovered that only a small number of previous studies, within software engineering and information systems literature, have investigated the omission of quality practices. These studies explain the omission of quality practices mainly as a result of organizational decisions and trade-offs made under resource constraints or market pressure. However, our study indicates that different aspects of this phenomenon deserve further research. In particular, future research must investigate the conditions triggering the omission of quality practices and the processes through which this phenomenon occurs. Especially, since software development is a human-centric phenomenon, the psychological and behavioral aspects of this process deserve in-depth empirical investigation. In addition, futures research must clarify the social, organizational, and economical consequences of ignoring quality practices. Gaining in-depth theoretically sound and empirically grounded understandings about different aspects of this phenomenon, enables research and practice to suggest interventions to overcome this issue.

Categories and Subject Descriptors: D.2.10 [Software Engineering]: Design; D.2.9 [Software Engineering]: Management; K.6.3 [Software Management]: software development; K.7.4 [The Computing Profession] Professional Ethics

General Terms: Design, Human Factors, Management

Additional Key Words and Phrases: Behavioral Software Engineering, Technical Debt, Systematic Literature Review

ACM Reference format:

Hadi Ghanbari, 2, Tero Vartiainen, 3, Mikko Siponen. XXXX. Omission of Quality Software Development Practices: A Systematic Literature Review. *ACM Comput. Surv.* XXXX, XXXX. XXXX (XXXX XXXX), 28 pages.
DOI: XXXX

Introduction

[1] This work was funded by Tekes (the Finnish Funding Agency for Technology and Innovation) and also supported by the European Union ITEA 2 Programme (Call 6) under grant no. 11011 (MERgE).

DOI: XXXX

Despite all the financial and human resources that have been spent on information systems and software development projects, defects and bugs in software products are widely reported in the research and practice literature [Fraser and Mancl. 2008, Brooks. 1995, Mancl, et al. 2007]. Previous research has shown that such software deficiencies are amongst the most important causes of software failures and vulnerabilities [Fonseca and Vieira. 2008, Wijayasekara, et al. 2012]. These software deficiencies not only make information systems vulnerable but also cause extensive financial costs for software stakeholders and societies [Fonseca and Vieira. 2008, Linberg. 1999, Wijayasekara, et al. 2012, Linberg. 1999, Wijayasekara, et al. 2012, Judy. 2009]. For example, previous studies estimate the cost of software deficiencies in just the United States to be almost 60 billion dollars [Judy. 2009, Tassey. 2002].

Minor and trivial software defects might not cause serious issues for stakeholders [Black. 2012], and ordinary users might even perceive and largely accept them as technical issues, such as application or operating system crashes and delays in services [Leveson and Turner. 1993]. However, because software systems deployed with critical bugs are more vulnerable to safety and security threats, they might result in devastating damages for stakeholders and societies in general [Fonseca and Vieira. 2008, Leveson and Turner. 1993].

In response to such quality challenges during the last four decades, researchers and practitioners, mainly from the software engineering discipline, have been engaged in improving software development and quality assurance processes by proposing a variety of methods, good practices, and tools [Sommervile. 2011, Poth and Sunyaev. 2014]. Although utilizing these recommended practices, methods, and tools might enable developers to identify and resolve defects in software products, software defects might stay hidden even after delivery in some cases [Wijayasekara, et al. 2012]. In addition, fixing identified software deficiencies becomes more expensive and time-consuming in the later stages of projects, especially after software delivery [Banker, et al. 1998, Van Emden and Moonen. 2002]. Therefore, such deficiencies should be avoided in the first place, especially in more critical and complex systems [Leveson and Turner. 1993, Wijayasekara, et al. 2012].

While significant amount of effort have been made for improving process-related and technological aspects of software development, psychological and social aspects of software development have received considerably less attention from software research and practice [Lenberg, et al. 2015]. This knowledge gap becomes more problematic considering that recent literature hints that software deficiencies might be the result of omitting proper software development practices or following "quick-and-dirty" shortcuts by development teams [Ahonen and Junttila. 2003, Austin. 2001, Baskerville and Pries-Heje. 2004, Baskerville, et al. 2001, Baskerville, et al. 2003, Vartiainen, et al. 2011]. In such situations, developers may often, for example, trade software quality for short-term gains by deciding to implement a task as soon as possible rather than following best practices. In this study, we refer to such quality-compromising decisions as “omission of quality practices.”

By omitting quality practices, software professionals (e.g., requirement analysts, programmers, testers, or project managers) purposefully opt to not follow proper software development practices that are recommended by either development procedures and standards or the software community. Instead, they choose to follow a questionable practice that might compromise the quality of software. For example, imagine that a programmer has a coding task

that can be performed in two alternative ways: A or B. Following A, the developer spends enough time and effort to perform his task according to a certain coding standard that is recommended to improve the quality of code. Alternately, by following B, the developer knowingly ignores the coding standard and follows a "quick-and-dirty" approach to finish the task quickly. When the developer chooses to follow B while being aware of A, we call this an "omission of quality practices." Examining why software professionals engage in such questionable practices is extremely important because any proper software development practice becomes ineffective if ignored purposefully.

In this article, our goal is to understand why software development teams knowingly decide to omit quality practices as previously defined. To gain such understanding, we decided to conduct a Systematic Literature Review [Kitchenham and Charters. 2007, Okoli and Schabram. 2010] to discover to what extent this phenomenon has been investigated by previous research. Through an extensive search performed on previous studies, only 19 studies were considered to be relevant for answering our research questions. The results of our study show that, despite its importance, several aspects of this phenomenon deserve further scholarly investigation. In particular, further research is needed to deeply investigate the contextual factors and conditions under which the omission of quality practices is initiated. Another area that requires further research is the psychological processes through which software professionals decide to perform such questionable practices. Furthermore, while previous studies consider several short-term consequences of omission of quality practices, future research needs to study the long-term consequences of such questionable practices for developers, organizations, and societies. Finally, future research must identify and suggest different interventions and solutions that could enable the software community to overcome the omission of quality practices.

The rest of this paper is structured as follows. In Section 2, the research methodology is presented, and different stages of the planning and conduction of the literature review are explained. In Section 3, the results of the literature review are reported and discussed in detail. The paper continues by discussing a synthesis of the literature review and our proposal for future research in Section 4. Finally, Section 5 summarizes the key findings.

Research Methodology

Conducting a literature review enables scholars to identify neglected research themes and spot critical gaps in the body of knowledge that deserve further scholarly investigation [Rowe. 2014]. It is suggested by previous studies that a Systematic Literature Review (SLR) is a suitable methodology for aggregating and evaluating completed and recorded research regarding a certain topic of interest to both identify gaps in the body of knowledge and propose directions for conducting future research to address these identified gaps [Kitchenham and Charters. 2007, Okoli and Schabram. 2010, Rowe. 2014, Kitchenham, et al. 2010]. Although conducting an SLR requires a significant amount of time and effort due to the large number of previous studies that must be identified and evaluated [Okoli and Schabram. 2010, Kitchenham and Charters. 2007, Petersen, et al. 2008], following a well-defined and reliable process can improve the comprehensiveness and scientific rigor of the SLR while reducing researchers' biases [Okoli and Schabram. 2010, Rowe. 2014, Petersen, et al. 2008].

At the very beginning of this research project, we conducted an initial literature review to identify previous studies related to our research questions. During this initial examination, we identified a limited number of studies relevant to our research topic. Therefore, we decided to conduct an SLR according to the guidelines suggested by Kitchenham [2004; 2007] and Okoli and Schabram [2010]. Both have been widely used for conducting SLRs in the Software Engineering (SE) and Information Systems (IS) disciplines. According to the results of our initial literature review, we decided to choose a wide range of search terms to identify a larger number of studies and to cover all the potentially relevant studies. By this, we aimed to indicate the gap in the literature regarding the omission of quality software development practices and to provide directions for future research. Figure 1 shows an overview of the literature review process.

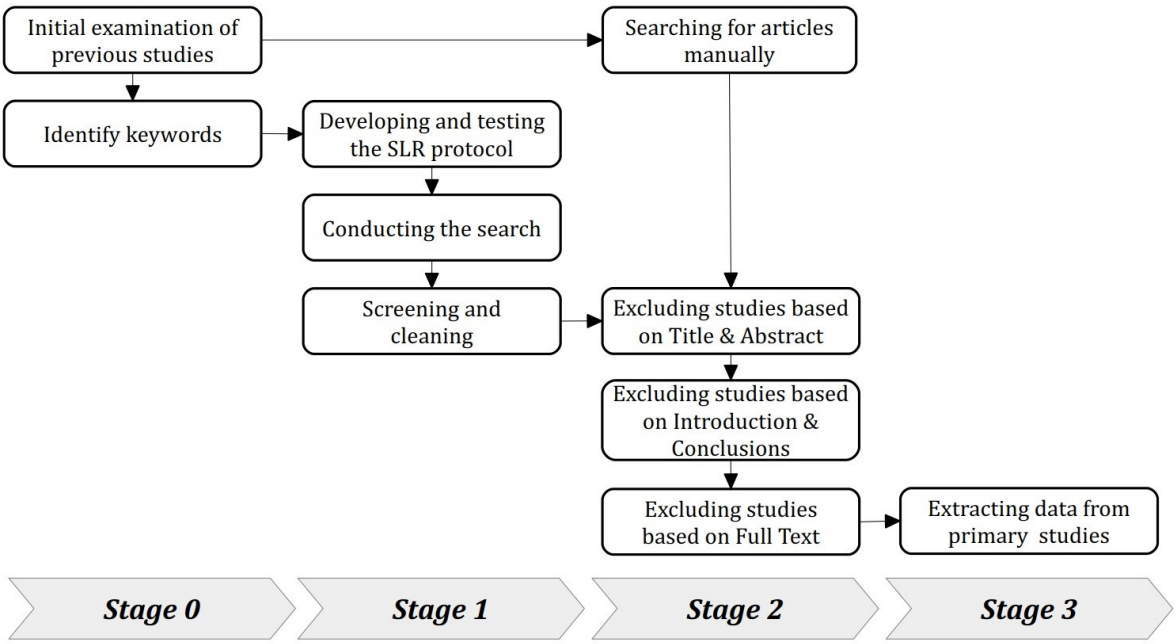

Fig. 1. The SLR was planned and conducted in four stages, as shown in this figure.

In the following sections, we discuss different stages of the process through which we planned and conducted our SLR.

Initial Literature Review Study (Stage 0)

To evaluate the state of research on the omission of quality software development practices, we conducted an initial literature review study in which we identified several studies reporting on the omission of software development methods and practices. Following this, and using the snowball technique, we searched the lists of references of these identified papers to discover additional relevant studies. Although this preliminary literature review did not return a considerable number of relevant studies, it helped us to identify a set of keywords that have been used by previous studies and software professionals, while also noting the issues regarding the

omission of software development methods and practices. These keywords were later used during Stage 1 of our SLR to search for and identify relevant literature.

Planning the Review (Stage 1)

During the planning stage, and according to the guidelines suggested by Kitchenham and Charters [2007], we prepared a search protocol to guide our SLR and increase the rigor of the review process. This search protocol was then tested by two of the authors and improved accordingly. This protocol consisted of our research questions, our search strategy (i.e., the search terms and resources which must be searched), study selection criteria and evaluation mechanism, data extraction strategy, and review timetable. In the following sub-sections we provide more detail about the contents of the review protocol.

Research Questions. According to the objectives of our research, we try to answer our main research question: *What is the state of research related to the omission of quality software development practices?* Based on this research question, we have formed the following sub-questions to be answered:

RQ1: How is the omission of quality practices explained by previous studies?

RQ2: What are the common instances of the omission of quality practices reported by previous studies?

RQ3: Under what conditions does the omission of quality practices take place?

Search strategy. After formulating these research questions, the search terms were chosen by identifying the keywords in the research questions and the results of our initial literature review study. By combining these search terms we have formed our search string (see Table 1).

Table 1. Search terms identified based on research objectives

Primary search terms	Software development, Software design, System* development, System* design
Secondary search terms	Omission, Omit, Questionable, Shortcut, Quick and dirty, Trade off, Technical debt, Dark side, Gray area, Dubious, Software quality
Search string	*("Software development" OR "software design" OR "system* development" OR "system* design") AND ("omission" OR "omit*" OR questionable OR shortcut OR "quick and dirty" OR "quick-and-dirty" OR "trade off" OR "trade-off" OR "technical debt" OR "dark side" OR "gray area" OR "grey area" OR "dubious" OR "Software quality")*

To identify the relevant studies, we performed the search on the *IEEE Xplore* (ieeexplore.ieee.org) and *ProQuest* (search.proquest.com) libraries during January 2015. After retrieving the results, we combined them into a single spreadsheet file containing records of 5072 studies. We then went through the list to identify and modify or remove any incorrect records or duplications. At this point, we added 17 articles that were manually identified by researchers but were not retrieved by the automatic search. After this step, our list consisted of a total number of 4838 unique studies.

Table 2. The results of the search conducted in January 2015

Database	Total number	Date range
IEEE Xplore Digital Library	3787	1968-2014
ProQuest	1285	1978-2015
Manual search	17	1998-2014
Total	5089	1968-2015
Total after screening	**4838**	**1968-2015**

Note: Google Scholar (scholar.google.com) was used for manual search.

Selection criteria and mechanism. In our review protocol, we agreed that each study must be evaluated by at least two reviewers and based on the predefined inclusion and exclusion criteria. A study was considered to be relevant if it recognizes the problem of ignoring quality software development practices or that of software professionals engaging in questionable practices during software or information system development processes. Studies were excluded if they were not peer-reviewed journal or conference articles published in English. Due to the large number of identified studies, we agreed to conduct the evaluation process in three consecutive rounds as explained in the following section.

Conducting the Review (Stage 2)

In the first round of Stage 2, two of the authors, Reviewers 1 and 2, independently evaluated the relevance of each study by reading its title and abstract. Following Kitchenham [2004; 2007], the reviewers tried to be quite liberal in performing this evaluation to decrease the chance of excluding any relevant studies. The results of the evaluation from each reviewer were then combined, and the disagreements between them were identified. Although the majority of these disagreements were resolved by reevaluating the studies and negotiation between the two reviewers, the reviewers' evaluations were contradictory in 26 cases. Therefore, according to our protocol, Reviewer 3 evaluated each of these 26 studies, and based on his evaluation, the disagreements between Reviewers 1 and 2 were resolved. At the end of this stage, a total of 91 studies were selected for further evaluation.

During the second round of Stage 2, Reviewers 1 and 2 evaluated the 91 studies based on their title, abstract, introduction, and conclusion sections. As in the previous round, when the reviewers' independent evaluations were completed, the results were combined, and disagreements were identified and resolved. At the end of this stage, a total of 47 studies were selected for further evaluation. Finally, during the third round of evaluation, the full texts of these studies were evaluated based on the selection criteria, and a total of 19 papers were considered to be, to some extent, relevant to our research questions and were selected as primary studies (see Table 3).

Table 3. Primary studies were selected through three rounds of evaluations

Round	Number of articles	Excluded articles	Evaluated based on
1st	4838	4747	title and abstract
2nd	91	44	introduction and conclusions
3rd	47	28	full paper

Data Extraction and Synthesis (Stage 3)

In stage 3 of the review process, data extraction, a set of relevant data items was extracted from each primary study (see Table 4).

Table 4 Data items extracted from primary studies

ID	Data item	Data item description	Related RQ
DI1	Article title	The title of the primary study	Overview
DI2	Author list	The full list of authors of the primary study	Overview
DI3	Publication Year	The year in which the primary study was published	Overview
DI4	Publication Forum	The name of the forum in which the primary study was published	Overview
DI5	Publication Type	Journal, conference, workshop, or book chapter	Overview
DI6	Research Type	Empirical or conceptual	Overview
DI7	Research Settings	Summary of the empirical research settings	Overview
DI8	Research Focus	The phenomenon under study in the primary study	RQ 1
DI9	Omission Instantiations	The type of quality practices and in which stage of software development they are omitted	RQ 2
DI10	Summary	A summary of the explanation provided about the omission of practices	RQ 1
DI11	Factors	The factors causing the omission of quality practices	RQ 3
DI12	Development context	Is the omission of quality practices bound to any specific software development method, process or approach?	RQ 2, RQ 3

As observed from Table 4, we have extracted data items beneficial for providing an overview of the primary studies (i.e., D1- D6), as well as those necessary for answering our research questions (i.e., D7 – D12). After extracting the data from primary studies, we further evaluated the relevance of each primary study to our research objectives based on short descriptive summaries of primary studies prepared by each individual reviewer.

Finally, during the data synthesis process, each of the primary studies was carefully analyzed to identify the suggested factors leading to the omission of quality practices. In addition, we tried to identify any potential mechanism or process through which software professionals decide to ignore quality software development practices.

Results of the literature review

In this section, we present and discuss the results of our SLR. As mentioned earlier, our initial sample included 4838 studies, from which we have selected 19 primary studies through 3 rounds of evaluations (see Appendix A). These primary studies include both empirical and theoretical research published in peer-reviewed journals, conference proceedings, and workshops between 1994 and 2014. An overview of these primary studies is shown in Table 5.

Table 5. An overview of the primary studies (PS).

ID	Year	Type	Research Settings and data collection method
PS1	2012	Empirical	35 semi-structured interviews with software professionals employed by companies of different sizes from the US and Canada
PS2	2003	Empirical	Semi-structured interviews, modeling sessions, archival materials, and discussions with the representatives of a software firm in a multinational organization
PS3	2014	Empirical	Data were extracted from source code comments written by software developers in four large open-source projects
PS4	2014	Empirical	7 focus group interviews with software professionals from 5 large Scandinavian firms producing embedded and general-purpose software
PS5	1994	Conceptual	No empirical data
PS6	1996	Conceptual	No empirical data
PS7	2013	Empirical	29 interviews, informal discussions, observations, and team meetings of 3 testing teams in India, the UK, and US
PS8	2013	Empirical	Data were extracted from defect log-files and responses to questionnaires from individuals active in 5 projects of a software vendor and a telecom operator in Turkey
PS9	2013	Empirical	Interviews, questionnaire, and ethnography in software development department of an industrial firm
PS10	2014	Empirical	18 interviews with software professionals and CEOs from 11 companies of different sizes active in a variety of business domains
PS11	2010	Empirical	Interviews and recorded log-files from one the key game development providers in Chinese online entertainment market
PS12	2004	Empirical	47 open-ended interviews with technical and business staff of 12 firms of various sizes from Denmark and the US
PS13	2014	Empirical	Data were collected by questionnaire from 54 software developers employed by organizations engaged in software development in Finland
PS14	2001	Conceptual	No empirical data
PS15	2011	Empirical	Data were extracted from software configuration management databases of 10 embedded-software development projects in a Dutch industrial company
PS16	1999	Empirical	Observations during a software project in a small but rapidly growing telecommunications company in US
PS17	2008	Empirical	Semi-structured interviews and archival documents from 7 international firms producing embedded software for automation, telecommunication, and transportation domains
PS18	2002	Empirical	Over 3 years of observation in a large project with a team of 50 software professionals developing an enterprise system for the leasing industry
PS19	2006	Empirical	Observations made during several experiments and case studies conducted in industrial firms. Additionally, data were collected from subjects participating in experiments

In regard to the publication venues, while the majority of the primary studies (i.e., 17 studies) are published in SE journals and conference proceedings, only two of the primary studies are

published in IS journals. From the 17 studies published in SE venues, 6 are journal articles, 8 are conference papers, and 3 are workshop papers (see Table 6).

Table 6. Overview of publication forums

Publication Venue	Type	#	PS
IEEE Software	Journal	3	PS1, PS6, PS16
Software Quality Journal	Journal	2	PS8, PS15
Information Systems Research	Journal	1	PS14
Information Systems Journal	Journal	1	PS12
Information and Software Technology	Journal	1	PS7
ACM/IEEE International Conference on Software Engineering (ICSE)	Conference	2	PS11, PS18
Euromicro Conference on Software Engineering and Advanced Applications (SEAA)	Conference	2	PS4, PS17
IEEE International Workshop on Managing Technical Debt	Workshop	2	PS9, PS13
IEEE International Conference on Research Challenges in Information Science (RCIS)	Conference	1	PS10
IEEE International Conference on Software Science, Technology and Engineering (SWSTE)	Conference	1	PS2
IEEE International Conference on Software Maintenance and Evolution (ICSME)	Conference	1	PS3
IEEE International Conference on Software Testing, Reliability and Quality Assurance	Conference	1	PS5
IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS)	Workshop	1	PS19

As observed from Table 6, of the only two primary studies published in IS venues, one is published in Information Systems Research (i.e., PS14), and the other is published in Information Systems Journal (i.e., PS12), which are both amongst the top IS journals. Because both of these studies were published in the early 2000s and only one of these studies is based on empirical observations (i.e., PS12), it seems that the omission of quality practices has not received enough attention from IS scholars in recent years.

Alternately, from the SE studies, 8 journal articles and conference papers are published in reputable SE venues, including 3 articles in IEEE Software (i.e., PS1, PS6, PS16), 2 papers in Software Quality Journal (i.e., PS8, PS15), 1 paper in Information and Software Technology Journal (i.e., PS7), and 2 papers in the proceedings of the ACM/IEEE International Conference on Software Engineering (i.e., PS11, PS18). However, none of the primary studies are published in top SE journals, such as IEEE Transactions on Software Engineering and ACM Transactions on Software Engineering and Methodology. This, in addition to the number of SE studies that were published in recent years (i.e., 10 studies since 2010), indicates that, while there has been increasing interest amongst SE scholars in studying different aspects of the omission of quality software development practices in recent years, these studies lack solid theoretical foundations.

After providing a short descriptive summary of the selected primary studies, we use data extracted from these primary studies and analyses prepared by each of the individual reviewers to answer our research questions in the following sections.

RQ1: How is the Omission of Quality Practices Reported By Previous studies?

As it can be seen in Table 7 below, the primary studies explain the intentional omission of quality practices from both organizational and individual perspectives. While the former perspective suggests that the decision to omit quality software development practices is made at the organizational level and due to certain business motivations or obligations, the later perspective explains the omission of quality practices as a result of developers' thought processes in favor of certain personal goals.

Table 7. A summary of the main findings of the primary studies

ID	Research focus	Summary of findings
PS1	Technical debt	Under time pressure and based on short-term thinking, developers ignore quality practices or perform temporary workarounds while making tradeoffs between quality, time and cost.
PS2	Software quality	Most of the common issues in software projects are caused by neglect or low-quality work. Poor feasibility studies, estimation, and planning decisions lead to resource constraints in projects and, in the absence of proper control mechanisms, lead to neglecting testing.
PS3	Technical debt	More experienced developers tend to produce more technical debt due to personal goals (which are not mentioned) regardless of release pressure or the complexity of the code.
PS4	Technical debt	Poor requirement specifications, approaching deadlines, the evolution of technology, and the splitting of development and maintenance budgets lead to violations of the architecture and ignoring refactoring, especially when firms are obliged to meet deadlines.
PS5	Challenges of software testing	Since the delivery of software, rather than quality, has higher priority for managers when coding and design are delayed, they prefer to shortcut testing to catch up with deadlines.
PS6	Challenges of software development	Due to bad estimates, development plans and schedules are often not accurate. Thus, time pressure leads to the elimination of 'non-essential' activities, such as requirements analysis, or software design and QA activities, such as reviews, test planning, and testing.
PS7	Challenges of software testing	Although testers work under more time pressure than developers and designers, their role is often underrated by managers. This might lower their motivation in performing testing, especially when they face the dilemma of missing deadlines or compromising the quality.
PS8	Software quality	Due to their confirmation bias, developers have a tendency to verify the quality of their code, and therefore, may avoid performing certain unit tests that would detect defects.
PS9	Technical debt	Developers want to ensure speedy releases and responsiveness to requirement

		changes. However, due to resource constraints and the evolution of technology, and in the absence of a disciplined development environment, they make trade-offs that lead to technical debt.
PS10	Software testing	Managers sometimes over-trust developers in producing high-quality software and do not involve testers in planning which leads to overlooking testing scope and underestimating necessary testing efforts. Thus, when managers decide to skip tests, due to resource constraints, this puts developers under stress and leads to low motivation for testing.
PS11	Software quality	Implementing penalty policies in software firms produces the fear of punishment among developers. As a result, software developers, especially novice ones, try to pay extra attention to maximize software quality and avoid intentional omission of quality practices.
PS12	Short-cycle time systems development	In an e-commerce context, due to evolving market demands, development cycles are compressed to be able to respond to constant market change. In such a context, trading software quality for the rapid delivery of high-priority features has become acceptable.
PS13	Technical Debt	Due to frequent requirement changes and scarce resources, the delivery of complete software becomes difficult, which may lead to violating best practices or design guidelines.
PS14	Shortcutting	Due to poor resource estimation and allocation, developers may face difficulties to meet deadlines. Especially in the absence of proper control mechanisms, developers who are concerned about quality of software may decide to take shortcuts to meet deadlines and avoid negative consequences of missing deadlines on their career.
PS15	Omission of software tasks	When there is slow start-up in projects or the budget is wasted, firms face problems delivering on time and within the budget. Especially when managers do not have a commitment to the firm's official software processes, developers ease up on these processes and omit important tasks.
PS16	Challenges of software development	Due to a lack of proper understanding of software quality amongst software professionals and in the absence of clear software development guidelines, software professionals may consider QA activities to be a waste of time and take shortcuts to improve productivity.
PS17	Software quality	Focusing on achieving short-term goals, such as shorter delivery times and higher productivity, motivates firms to minimize software processes. Resource constraints and managers' low architectural awareness are other factors leading to the taking of shortcuts.
PS18	Extreme Programming	Due to incorrect estimates, developers take shortcuts and ignore refactoring to ensure speedy development and perform minimal work, especially if they believe too much in their methods.
PS19	Omission of software tasks	Often documentation is ignored or postponed in practice due to the lack of proper documentation guidelines or due to a lack of attention to the importance of documentation.

To explain such organizational and individual decisions, the primary studies use a variety of terminologies, including 'shortcutting' [Austin. 2001], 'systematic omission of software tasks'

[Samalikova, et al. 2011], 'technical debt' [Cunningham. 1992], and 'short-cycle time development' [Baskerville and Pries-Heje. 2004]. In addition, there are a few primary studies that report the research problem by discussing the common issues and challenges of software development projects in general and those of performing software quality assurance and testing activities in particular.

In the following sections, we discuss each of these different viewpoints on the omission of quality practices in more detail.

Omission of quality practices under organizational constraints. The majority of the primary studies explain the omission of quality practices in terms of eliminating certain steps or activities recommended by firms' official software development processes [McConnell. 1996, Fleming. 1999, Ahonen and Junttila. 2003, Samalikova, et al. 2011, Shah, et al. 2014, Murugesan. 1994, Seth, et al. 2014]. Such shortcuts are mainly taken under resource constraints (e.g., time and money) or due to the lack of attention to certain software development tasks and activities (e.g., documentation or testing) by managers and developers.

As argued by McConnel [1996], often due to incorrect estimations at the beginning of software projects, development teams prepare inaccurate plans and overly aggressive schedules, and therefore, often during the later stages of projects, developers face scheduling problems. As a result, when a development team is under time pressure, they often eliminate certain activities that they consider to be 'non-essential,' such as requirements analysis or architectural design, or quality assurance activities, such as reviews and testing [McConnell. 1996]. These findings are in line with observations reported by Fleming [1999] regarding an industrial software development and maintenance project. In this study, the author explains how software development processes, and especially quality assurance activities, such as design reviews, are ignored by managers and developers simply because they consider such activities as wastes of time, and therefore, they prefer to just concentrate on producing the "real" software [Fleming. 1999].

The omission of software development activities under the influence of resource constraints is also reported by Ahonen and Junttila [2003]. Conducting case studies and interviewing software developers Ahonen and Junttila [2003] suggest that development teams usually face with lack of sufficient time and resources because the early phases of software projects usually becomes longer than what has been planned. As a result of such resource constraints the quality assurance activities, such as inspection and testing, are often postponed and eventually skipped entirely [Ahonen and Junttila. 2003]. Another primary study that supports these findings is [Samalikova, et al. 2011]. This study reports that due to delays in the initial phases of software development, development teams are often faced with resource constraints. In such situations, especially when the management is not committed to the firm's official process, developers do not pay attention to the quality practices, and they might take shortcuts to address scarce resources [Samalikova, et al. 2011].

An empirical study by Shah, et al. [2014] reports that test engineers often experience more pressure while performing their tasks compared to other software professionals, such as developers and designers. Such extra pressure is because when software design and development phases are delayed, testers are the ones who must accommodate such delays [Shah, et al. 2014]. However, the importance of testing activities and consequently the contribution of testers to the software development is not highly appreciated by managers and other stakeholders [Shah, et al.

2014]. Thus, testers usually face a dilemma: to either meet deadlines by compromising the quality of software or miss the deadline but perform their tasks in a high-quality manner [Shah, et al. 2014]. In such situations, developers' motivations and the appreciation of testing activities by managers and other stakeholders are considered to be a key factor influencing how well testing activities are performed by testers. Such negative attitudes towards testing are also reported by two other primary studies [Murugesan. 1994, Seth, et al. 2014]. In his study, Murugesan [1994] argues that, even though testing is "a key contributor to software quality" assurance, it often receives less attention from management. Additionally, the author suggests that, for many developers, testing is like a 'cushion' that can be squeezed whenever needed during the development process. Therefore, whenever the design and coding stages take longer than planned, project managers prefer to reduce the testing time to deliver the software before the deadline [Murugesan. 1994]. Finally, the results from another empirical study on software testing [Seth, et al. 2014] suggest that project managers deliberately do not involve testers in various project activities, mainly project planning, because they believe too much in the abilities of development teams to produce high-quality software. Therefore, testing scope and necessary testing efforts are often overlooked in the contracts. Consequently, later on during the projects and due to the lack of sufficient resources, project managers decide to skip important software tests [Seth, et al. 2014].

As reported by the first group of primary studies, due to improper estimation and planning activities, software projects are often faced with scarce resources. In such situations, if quality practices do not receive sufficient attention and appreciation from organizations, software developers might not be motivated to perform such quality practices and, as a result, compromise software quality.

Omission of quality practices for gaining strategic competitiveness. Another group of primary studies explains the omission of quality practices in terms of strategic business decisions made by organizations to gain competitive advantages in the market environment and to achieve short-term goals. This group of primary studies uses either technical debt [Cunningham. 1992] or agile software development [Fowler and Highsmith. 2001] terminologies to note such strategic business decisions. In the following sub-sections, these viewpoints are discussed.

Occurrence of Technical Debt. A group of primary studies explain the omission of quality practices [Lim, et al. 2012, Potdar and Shihab. 2014, Martini, et al. 2014, Codabux and Williams. 2013, Holvitie, et al. 2014, Lindgren, et al. 2008] in terms of decisions leading to occurrence of technical debt [Cunningham. 1992]. The metaphor of technical debt [Cunningham. 1992] denotes the consequences of producing low-quality software in situations where organizations make conscious business decisions to achieve short-term goals by compromising or fully eliminating certain software development activities [Lim, et al. 2012, Martini, et al. 2014] to speed up delivery times [Brown, et al. 2010, Lim, et al. 2012].

According to this group of primary studies, such quality-compromising trade-offs are mainly tactically and reactively made by firms under the influence of market demands. From a business perspective, software companies are motivated to increase their productivity mainly in terms of reducing time-to-market and development costs [Lindgren, et al. 2008]. Alternately, software companies need to be responsive to market demands and customers changes [Codabux and Williams. 2013]. Therefore, in such a business environment, taking on technical debt in the short-term might be beneficial or even unavoidable for software companies [Brown, et al. 2010] to catch

market share [Lim, et al. 2012, Lindgren, et al. 2008] or fulfill their contractual obligations [Martini, et al. 2014]. However, because such short-term decisions affect the quality of software, development teams are supposed to go back and fix such workarounds as soon as possible to maintain the quality of the software products in the long run [McConnell. 2007, Brown, et al. 2010]. However, if the skipped tasks are not implemented during the later stages of software development (i.e., the short-term technical debt is not paid back), this leads to higher levels of software deficiency and complexity and, as a result, incurs increased maintenance costs over time [Codabux and Williams. 2013].

Use of Agile software development. Another group of primary studies reports the omission of quality practices in terms of utilizing novel software development approaches, such as Internet-speed or short-cycle time system development [Baskerville and Pries-Heje. 2004] and Extreme Programming [Beck. 1999]. At the turn of the millennium, the rise of electronic commerce provided firms with an opportunity to access a wider range of customers by distributing their products or services through the Internet. However, fierce competition in this fast-changing environment put firms under constant pressure to deliver new software products to market faster [Baskerville, et al. 2001, Baskerville, et al. 2003]. As a result of such 'Internet Time' [Baskerville and Pries-Heje. 2004] rush to the marketplace, companies had to shorten the length of their software development cycles [Baskerville, et al. 2001, Baskerville and Pries-Heje. 2004]. It must be noted that Internet-speed and short-cycle time software methodologies are similar to the agile school of thought [Baskerville, et al. 2003].

As suggested by Baskerville and Pries-Heje [2004], Scrum [Schwaber and Beedle. 2001] and Extreme Programming [Beck. 1999], which are two of the most popular agile software development methods, were developed based on the short-cycle development practices used by Microsoft and Netscape during their competition in developing web browsers. Generally speaking agile software development aim at minimizing development costs and delivery times by avoiding nonessential activities during software development processes [Martin. 2003, Codabux and Williams. 2013]. Due to such demands for shorter development cycles, development teams might become more eager to focus on software functionality and therefore do not pay enough attention to other software activities, such as design, testing, and maintenance [Baskerville and Pries-Heje. 2004, Codabux and Williams. 2013, McConnell. 1996]. As a result, the overall complexity of the software and the likelihood of producing defective software are increased [Agrawal and Chari. 2007, Gibson and Senn. 1989].

Based on qualitative interviews conducted with members of 12 companies from the US and Denmark producing software for fast changing markets , Baskerville and his colleagues determined that such a fast-paced development requires development teams to follow quick and parallel release-oriented prototyping approach in where "quality is negotiable" [Baskerville and Pries-Heje. 2004]. Another study by [Elssamadisy and Schalliol. 2002] reports similar observations from a large 3-year-long software project. In this study, the authors suggest that, following principals suggested by extreme programming in large projects, developers try to speed-up development and perform minimal work. However, due to incorrect effort estimates and their excessive belief in the processes, they have to take shortcuts and ignore refactoring to reach their goals within short development cycles [Elssamadisy and Schalliol. 2002].

According to the second group of primary studies, it seems that the overemphasis of short-term goals, such as the delivery of new software features and shorter delivery times, by the software industry increases firms' eagerness to speed-up development processes and therefore

might reduce developers' attention to the importance of software quality. As a result, the omission of quality practices with the hope of increasing productivity becomes acceptable within the software community.

Omission of quality practices to achieve personal goals. Finally, a group of primary studies explain the omission of quality practices in terms of developers' thought processes towards achieving certain personal goals [Austin. 2001, Çalıklı and Bener. 2013, Wang and Zhang. 2010].

The first study by Austin [2001] suggests that under time pressure and in response to unexpected difficulties during the software development processes, developers might become motivated to take quality-compromising shortcuts to stay on schedule, especially if they consider the deadline to be unachievable. In this conceptual study, the author [Austin. 2001] argues that taking shortcuts is not necessarily a deliberate subversive act but rather the result of developers' strategic decisions to address the situation in the most convenient way. In making such quality-compromising decisions, developers often have two main concerns: concern for their career and concern for the quality of the software [Austin. 2001]. From the career perspective, developers might take shortcuts to avoid the consequences of being behind schedule and losing their professional reputation by being the only developer who cannot be on time. Alternately, from a quality perspective, developers might avoid taking shortcuts because they are concerned with being penalized for compromising the quality of the software and, as a result, endangering the success of the project [Austin. 2001].

Another study by [Çalıklı and Bener. 2013] explains how developers' confirmation biases may cause the emergence of software defects. Confirmation bias, as explained by Çalıklı and Bener [2013], is a "tendency of people to seek evidence that verifies hypotheses" rather than seeking evidence that could falsify those hypotheses [Çalıklı and Bener. 2013]. In this empirical study, the authors found some indications that, under the influence of their confirmation biases, developers might try to provide evidence that their code is working properly. Therefore, they might only run certain unit tests that prove the code is working and avoid performing those unit tests that break the code [Çalıklı and Bener. 2013]. As a result of such quality-compromising decisions, the defects in the code might not be discovered.

Finally, in their field of study, Wang and Zhang [2010] discuss the influence of organizational punishment on the quality of software development. In this study, the authors investigated the influence of penalty policies employed by a large Chinese software company on the quantity of software defects identified in the code. Based on this penalty policy implemented in the company, those individual developers who delivered defective software were punished by taking away a specific amount of money, per defect, from their salary. The results of the study suggest that penalty policies partly affect novice developers' performances, leading to less defective software [Wang and Zhang. 2010]. As an example, an interviewee explained that the penalty policy made them avoid defects that were based on carelessness.

The results of the third group of primary studies suggest that the omission of quality practices might be an individual decision privately made by developers to gain certain career-related advantages. It seems that, in such situations, if developers perceive the omission of quality practices to be beneficial for them, while there is a small chance that such quality-compromising decisions will be revealed, they might decide to ignore the quality practices. This might be the

reason that implementing penalty policies could affect the avoidance of the omission of quality practices.

RQ2: What Are the Common Instances of the Omission of Quality Practices Reported By Previous studies?

To answer our second research question, we have identified all instances of ignoring software development tasks and activities that are reported by primary studies. We have categorized all of these instances into 6 groups according to the nature of the software development activities that are ignored (see Figure 2).

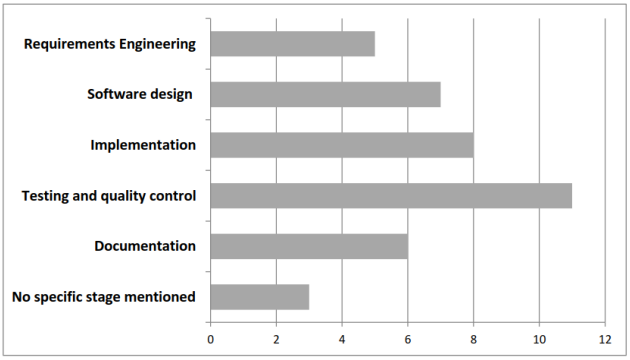

Fig. 2 Common instances of the omission of quality practices reported by primary studies

As it can be seen from Figure 2, the majority of primary studies (i.e., 57%) reported at least one instance of testing and quality control activities being ignored by development teams. Such activities include, for example, planning and scoping testing activities [Lim, et al. 2012, McConnell. 1996], writing automated unit tests [Codabux and Williams. 2013], conducting formal reviews [Ahonen and Junttila. 2003], and performing testing [Murugesan. 1994, Baskerville and Pries-Heje. 2004]. In addition to this, 42% of the primary studies mention the occurrence of quality-compromising activities during the design and implementation stages, while 31% of the primary studies report instances of ignoring documentation. Finally, 26% of the primary studies report that the omission of quality practices takes place during the requirements analysis and specification phase. It must be noted that 15% of the primary studies report the omission of quality practices in general and do not provide any specific instance nor mention the particular stages of software development that were compromised.

These results show that the omission of quality assurance and testing activities is considerably high among development teams. Keeping in mind that such activities play a vital role in ensuring the quality and reliability of software products, it becomes obvious that this specific aspect of software development has received less attention from the software community. It seems that the constant demands from the software industry and fierce competition between software companies motivate development teams to concentrate more on the delivery of new functional features rather than evaluation of the quality of the software. Such oversight can be a good explanation for high rates of software defects and project failures.

RQ3: Under What Conditions Does the Omission of Quality Practices Take place?

In response to our third research question, we identified a variety of factors during the data analysis that are reported by previous studies as affecting developers' behaviors during the software development processes and, as a result, leading to the omission of quality practices (see Table 8).

Table 8. Factors causing the omission of quality practices.

Identified factors	Description	Instances
Business goals	From a business perspective, it is desirable or even vital for companies to increase their market share and consequently increase their revenue. As a result, organizations might ignore quality practices to achieve such short-term goals.	Eagerness to increase sales (PS2, PS9), Reduce development costs (PS2, PS3, PS8, PS17), Rapid delivery of high-priority features (PS4, PS17), Collect external funding (PS1), Capture market share (PS1, PS17), Reduce time-to-market (PS1, PS3, PS6, PS9, PS12, PS14, PS16, PS17, PS18)
Customers' requirements	Customers' requirements are often not clear at the beginning of projects, which makes requirement changes unavoidable. Thus, sometimes developers might ignore quality practices to address these issues.	Collect early feedback from customers (PS1), Customers' wish lists are too long (PS1), Fuzzy requirements (PS1, PS12), Requirement changes (PS1, PS4, PS12, PS13, PS17)
Project constraints	The extent to which software activities are followed highly depends on the availability of necessary resources, such as time, budget, workforce, and the quality of official development guidelines and control mechanisms in the company.	Lack of time (PS1, PS2, PS4, PS5, PS6, PS7, PS9, PS10, PS13, PS15, PS18), Lack of human resources (PS9, PS13, PS17), Lack of financial resources (PS2 PS9, PS13), Lack of technical skills (PS2, PS9, PS13, PS17), Lack of clear process guidelines (PS4, PS9, PS10, PS19), Lack of clear architectural documentation (PS4), Lack of effective quality control mechanisms (PS14, PS15)
Technical issues	In some situations, the performance of quality practices is ignored due to technical difficulties associated with software development.	Technology evolution (PS4, PS9, PS12), Use of legacy code (PS4), Use of third-party software (PS4)
Psychological factors	In some cases, ignoring quality practices is an individual decision made by managers, developers, or both and due to their attitudes, feelings, beliefs, or cognitive characteristics.	Lack of commitment to development processes (PS1, PS5, PS15, PS16, PS19), Lack of motivation to perform tasks (PS7, PS19), Developers' confirmation bias (PS8), Interpret requirements conveniently (PS14), No fear of punishment (PS11, PS14), Risk-taking behavior (PS10), Poor buy-in for testing (PS5, PS7, PS10)

As observed from Table 8, a variety of reasons are reported by primary studies as possibly leading to the omission of quality practices. While the majority of the primary studies emphasize

the role of resource constraints as a key driver of ignoring quality software development practices, other reasons, such as constant market demands, lack of understanding of customers' requirements, individuals' attitudes and motivations, and technical difficulties associated with software development, are also suggested to cause such questionable practices.

Based on the nature and similarity of identified factors, we have divided them into five main categories: *Business goals*, *Customers' requirements*, *Project constraints*, *Technical issues*, and *Psychological factors*. These categories are succinctly described and different instances of them are reported in Table 8. In the next section we discuss about these identified factors and propose a theoretical model accordingly.

A Synthesis of the Literature Review

Using the five categories of factors illustrated in Table 8, we produced a synthesis of the five categories of factors that entail the omission of quality practices (see Figure 3). By indicating their scope of effects and interrelationships between these identified factors, our model represents the context in which the psycho-social process of the omission of quality practices is initiated and emerged overtime. We call this process psycho-social because the omission of quality practices occurs in a social context but is implemented by single individuals involved in software development and under the influence of their psychological factors. Based on their scope of effects, these categories are organized into three contextual levels, the market level, organizational level, and individual level.

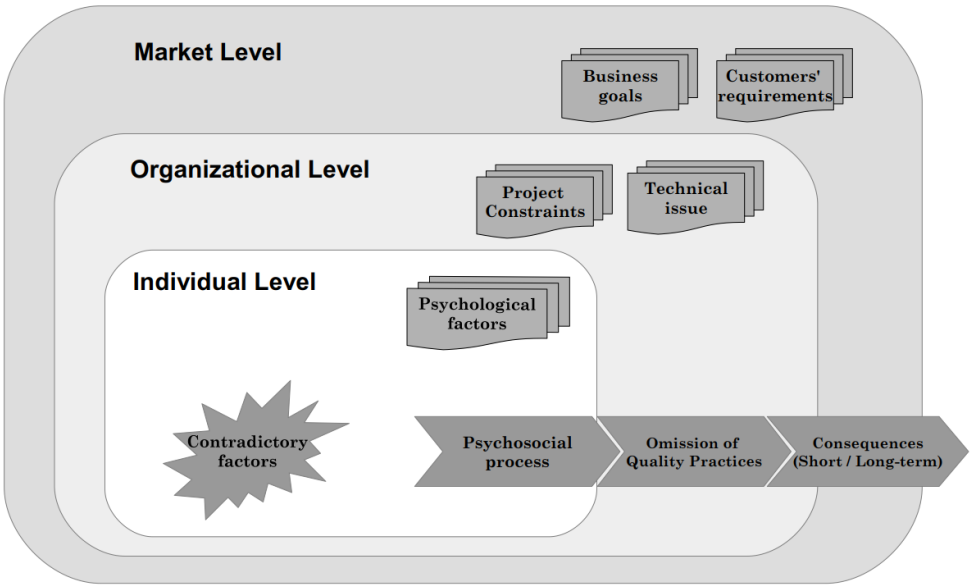

Fig. 3. The context of the psycho-social process of omitting quality practices

As shown in Figure 3, in the market level, *business goals* and *customers' requirements* are two main factors influencing the extent to which quality software practices are followed. In the

organizational level, in contrast, the decision to ignore quality practices is influenced by *project constraints* and *technical issues*. Finally, in the individual level, *psychological factors* affect individuals' decisions regarding the omission of quality practices. These different levels of context, together with the psycho-social process that entails omission instantiations and their consequences, are presented in the following sections.

Market Level

Based on our analysis, software development approaches and the extent to which they are followed by organizations are influenced by the market environments that firms are active in. Therefore, two main factors that influence these approaches are firms' business goals and customers' requirements in such market environments.

From a business perspective, increasing sales [Ahonen and Junttila. 2003, Codabux and Williams. 2013] and extending market share [Lim, et al. 2012, Lindgren, et al. 2008] play important roles in increasing firms' revenues. Alternately, reducing time to market [Potdar and Shihab. 2014, McConnell. 1996, Codabux and Williams. 2013, Baskerville and Pries-Heje. 2004] and development costs [Ahonen and Junttila. 2003, Potdar and Shihab. 2014, Shah, et al. 2014] enables software companies to increase their profits or might even be critical for a firm's survival in highly competitive markets. Achieving such business goals in the short term might increase companies' eagerness to speed up their development processes and rapidly deliver new products or novel functional features to the market. By this, not only are firms able to reach the market before their competitors, but they also might be able to increase their revenue and, in some cases, collect external funding [Lim, et al. 2012] to further develop and improve their products. Such strategies are especially vital for small companies active in highly competitive and turbulent market environments with fierce competition. Therefore, and as a result of companies' strategic decisions as explained in section 3.1.2, development teams might decide to ignore certain quality software development practices. As an example, Ahonen and Junttila [2003] report that while preparing project offers for clients, it is tempting for sales people to reduce unnecessary costs and delays by implementing feasibility studies without proper technical and managerial knowledge. This is because sales people might consider involvement of technical people in this process as an additional cost and delay.

Another factor that influences firms' strategic decisions to ignore certain quality practices is customers' requirements in markets. Customers' needs and requirements are often vague and fuzzy, especially in the initial stages of software projects [Lim, et al. 2012, Baskerville and Pries-Heje. 2004]. Lack of adequate understanding of requirements among software stakeholders in general, and customers in particular, often leads to rework as developers make design assumptions that later need to be changed [Lim, et al. 2012]. Therefore, especially during the early stages of projects, development teams might decide to follow "quick and dirty" practices to quickly deliver mockups or even prototypes with minimal functionality to collect feedback from customers [Lim, et al. 2012] and improve the requirements.

Alternately, stakeholders gain a better understanding of customers' needs over time, and therefore initial requirements need to be changed. To satisfy customers, firms often try to increase their response to be able to accommodate such requirement changes [Martini, et al. 2014, Baskerville and Pries-Heje. 2004, Holvitie, et al. 2014, Lindgren, et al. 2008]. Thus,

development teams might decide to ignore certain quality practices to aid in being responsive to customers' needs. For example, as reported by Baskerville and Pries-Heje [2004], Internet-time development is characterized by fuzzy requirements and market pressures. In such an environment, customers appreciate the fast delivery of changing requirements, and they do not even expect high-quality products. As a result of such "negotiable quality" [Baskerville and Pries-Heje. 2004], the omission of quality practices becomes acceptable to satisfy customers' needs and expectations.

Our interpretation is that these external market-level factors have a determining role in the omission of quality practices. This is because, depending on the market environment, organizations follow different strategies to increase their sales (and consequently profit) and to satisfy their customers. Such underlying factors motivate firms to speed up negotiation and development processes and to extend their market share by delivering their products to market faster.

Organizational Level

Based on our analysis, in the organizational level, project constraints and technical issues are the main factors influencing firms' decisions regarding the omission of quality practices.

Project constraints point to the lack of resources necessary for performing quality software development practices. Such resources include time, budget, skilled workforce, official development guidelines and procedures, and control mechanisms in the company. In software projects, unreliable cost and effort estimation and schedule errors [McConnell. 1996, Elssamadisy and Schalliol. 2002] often lead to a lack of necessary resources to perform each development phase. A lack of sufficient time [Lim, et al. 2012, Ahonen and Junttila. 2003, Martini, et al. 2014, Murugesan. 1994], financial resources [Ahonen and Junttila. 2003, Seth, et al. 2014, Holvitie, et al. 2014], or skilled human resources [Codabux and Williams. 2013, Holvitie, et al. 2014, Lindgren, et al. 2008] are among the main reasons that often force development teams to ignore quality software development practices. To deal with scarce resources, developers are often encouraged to skip those development tasks and activities that, from their perspective, are considered unnecessary [McConnell. 2007, Potdar and Shihab. 2014, Fleming. 1999]. Additionally, when there is a split of budget and resources between different development phases, for example, between implementation and testing or development and maintenance, software professionals might become motivated to skip certain tasks and practices during the development phase and postpone them to the maintenance phase [Martini, et al. 2014].

While lack of time, financial, and human resources restrict developments teams' abilities to follow quality practices, the lack of clear software development guidelines [Martini, et al. 2014, Codabux and Williams. 2013, Seth, et al. 2014, Bayer and Muthig. 2006] and inadequate inspection and quality control mechanisms [Austin. 2001, Samalikova, et al. 2011] facilitate the omission of quality practices [Martini, et al. 2014, Murugesan. 1994, Codabux and Williams. 2013, Seth, et al. 2014, Austin. 2001]. In the absence of proper requirements identification and analysis practices, not only is it very difficult for developers to identify and explicitly document software requirements, but cost and effort estimation also becomes unreliable. For example, according to Martini, et al. [2014] and Lim, et al. [2012], inadequate requirements specification and a lack of clear architectural documentation might be misinterpreted by developers in

subsequent development stages and eventually lead to the implementation of incorrect functionalities that must be fixed in the future.

Alternately, because software development processes are invisible to non-developer stakeholders [Austin. 2001], this might provide developers with an opportunity to consciously ignore certain quality practices without managers or customers being aware of it [Lim, et al. 2012, Austin. 2001]. In such situations, the lack of adequate inspection and quality control mechanisms facilitates the omission of quality practices. For example, Ahonen and Junttila [2003] report that the lack of formal inspections causes obvious mistakes to be retained in documents, as experienced people seem to think that an obvious mistake must be there for a reason.

In the organizational level, technical issues are, in some situations, the underlying reasons for the ignoring of quality practices. Such issues include technology evolution [Martini, et al. 2014, Codabux and Williams. 2013, Baskerville and Pries-Heje. 2004], the use of legacy code, and the use of third-party software [Martini, et al. 2014]. The software field is a fast changing environment due to rapid technological improvements. With such technological evolution, it is possible for software and hardware to become obsolete over time [Martini, et al. 2014], and therefore it might not be beneficial for firms to invest too many resources in improving the quality of their software products. This creates a constant need to replace old software and hardware components with new ones. If legacy code or third-party software [Martini, et al. 2014], for example, is used, any potential architectural debt underlying these components will be transferred to the new software [Martini, et al. 2014]. This means that, if refactoring is not performed and the debt is not paid back, software complexity grows, and future development of the software becomes problematic [Martini, et al. 2014]. In some situations, software developers might be forced to perform temporary workarounds to address such structural issues and complexities. For example, [Murugesan. 1994] suggests that, due to the complexity of systems, software testing and evaluation become more challenging, and as a result, it is more likely for developers to ignore quality practices.

As discussed in this section, different organizational-level factors, under the influence of the market environment, might force or even motivate development teams to ignore quality practices. However, such institutional-level constraints or motivators cannot be seen as sufficient for the omission of quality practices because the decisions to ignore such practices are made and implemented by individuals. In the next section, we discuss the psychological factors underlying the omission of quality practices.

Individual Level

At the individual level, managers and developers engage with the decision-making processes regarding the omission of quality practices. This decision-making is a psycho-social process because it is influenced by individuals' psychological characteristics and thought processes, as well as the characteristics of the development context (i.e., the market-level and organizational-level factors). Here, the psychological factors relate to attitudes, beliefs, and cognitive tendencies that may incline managers and developers to omit quality practices.

For example, as suggested by previous studies, the lack of commitment to firms' software procedures [Samalikova, et al. 2011, Fleming. 1999, Bayer and Muthig. 2006] and lack of

appreciation for quality control and testing activities [Murugesan. 1994, Shah, et al. 2014, Seth, et al. 2014] might decrease developers' motivations to perform quality practices [Shah, et al. 2014, Bayer and Muthig. 2006]. In the previous section, we explained that the lack of clear procedures might facilitate the omission of quality practices. However, it must be noted that, even if there are proper software development procedures available in firms, the lack of commitment to these guidelines from managers might simply lead to the neglect of those procedures by development teams [Ahonen and Junttila. 2003, Murugesan. 1994]. In such situations, development teams might prefer to concentrate on producing 'real software' [Fleming. 1999] rather than planning [Lim, et al. 2012, Murugesan. 1994, Fleming. 1999], and as a result, they decide to jump directly to coding and skip project planning activities and other important steps, such as requirements analysis and architectural design. It seems that such negative attitudes towards quality practices are more common in the case of performing quality control activities. Software testing has often received so-called 'second-rate' consideration from stakeholders, which leads to the undermining of testing activities, and therefore it is common for quality control and testing activities to be ignored [Murugesan 1994].

Underestimation of the importance of quality practices by managers alongside cognitive characteristics of individuals, such as confirmation bias [Çalıklı and Bener. 2013] and risk-taking [Seth, et al. 2014], might lead to developers' decisions to ignore quality practices. For example, Seth [2014] reports that testers are not always involved in project planning because managers overly trust development teams' abilities to produce high-quality software and may take the risk of deciding to skip certain important tests [Seth et al. 2014]. Çalıklı and Bener [2013] suggest that, under the influence of confirmation bias, developers might skip certain tests that could possibly break their code and reveal its underlying defects. Such decisions are more likely to be taken if developers do not have any fear of getting caught and being punished by organizations [Wang and Zhang. 2010, Austin. 2001]. Because such questionable practices are not easily observable, developers do not feel any fear of facing punishment, and as a result, they might decide to ignore quality practices. The result from an empirical study by Wang and Zhang [2010] supports this finding, as they show that implementing penalty policies could lead to the avoidance of intentional technical debt [Wang and Zhang. 2010]. This means that the existence of penalty policies may prevent intentional omission of quality practices.

Based on these exemplary studies, we suggest that psychological factors affect the decision-making processes regarding the omission of quality practices, whether pro or against. For example, cognitive tendencies, such as risk-taking or cognitive bias, may positively affect the emergence of omission behavior, and the fear of penalties may work against omission behavior.

The Psycho-Social Process and Consequences of Omission Behavior

During the review and synthesis, we have tried to identify mechanisms or processes through which software professionals decide to neglect quality practices. The majority of the studies deemed that developers simply decide to neglect quality practices, either under schedule and management pressure or based on some personal motives. However, none of the primary studies provide any in-depth explanation regarding the psycho-social mechanisms underlying the omission of quality practices. While this psycho-social process is still unknown, managers and

developers go through it by producing omission instantiations on a daily basis in every software development project.

The psycho-social process is likely to be affected by the factors we identified and discussed in this review. However, while this psycho-social process and its underlying mechanisms are undiscovered, the current literature reports a variety of possible factors that may affect this process. Our current understanding is that the identified factors may play different positive or negative roles in the omission of quality practices depending on the context or situation. Despite the fact that a group of studies suggest that the omission of quality practices is associated with a lack of technical skill [Holvitie, et al. 2014, Lindgren, et al. 2008, Codabux and Williams. 2013], Potdar and Shihab [2014] found in their empirical study that a higher amount of technical debt is produced by developers who are more experienced. These findings show that, if the level of individuals' skills is a factor influencing the psycho-social process, it may have a negative or positive role, meaning that, in the case of skills, both a lack of skill and skillfulness may positively affect the omission behavior.

Based on our analysis, we believe that the values of both productivity and quality seem to play an important role in the psycho-social process of the omission of quality practices. In the software industry, higher productivity is often associated with faster delivery of more features [Fleming. 1999, Lindgren, et al. 2008]. However, concentrating on producing more and doing so faster might increase the number of software defects, which require extra effort and rework to be fixed and consequently decrease both the quality of the software and the long-term productivity [Lindgren, et al. 2008]. Therefore, from a technical perspective, producing high-quality software might be seen as the key to higher productivity [Fleming. 1999]. It seems that a typical scenario of the omission of quality practices relates to the conflict between these contradictory individual and organizational concerns. Developers are mainly concerned with performing quality work because they have to work with the code and face its issues on a daily basis [Lim et al. 2012]. Managers, in contrast, experience the pressures of business demands and therefore are concerned with getting work done quickly and with the available resources. Such conflicts might trigger the psycho-social processes that lead to the omission of quality practices.

Regarding its outcome, the omission of quality practices can have different short-term and long-term consequences for individuals, organizations, and societies. In the short term, ignoring quality practices might enable firms to speed up software delivery to capture market share and obtain early feedback with which to improve the software [Lim, et al. 2012]. Alternately, the consequences of the omission of quality practices might occur only after the completion of projects and have long-term effects on the organizational level as well as with respect to the firm's position within the market environment. For example, because the omission of quality practices increases software defects [Lim, et al. 2012], organizations may spend extra time and resources to solve these defects, while facing too many issues makes customers unhappy [Lim, et al. 2012] and eventually might decrease the firm's market share.

As discussed earlier, the omission of quality practices might be unavoidable in certain business contexts, and therefore, firms need to find the best possible compromises. To identify such trade-offs, software development teams can use lessons learned from other projects to develop context-dependent solutions suitable for their needs. Such approach has been promoted by advocates of Software Engineering Method and Theory (SEMAT) initiative in recent years

[Jacobson and Seidewitz. 2014]. By suggesting a theoretical foundation which is called the kernel for software engineering, the SEMAT initiative aims at assisting software practitioners and teams to create and expand appropriate practices within their firms or even across their industrial domain. Therefore, it becomes apparent that considering the consequences of ignoring quality practices plays a key role in considering the omission of quality practices.

Future Research

In this study we have identified a number of gaps in the current literature about the omission of quality practices. Therefore, further research is needed to address these gaps and to suggest preventive or developmental means for considering the identified issues. In the following paragraphs we suggest a number of research areas which could be addressed by future research.

What are the instantiations of the omission of quality practices and their nature? Our results indicate that the omission of quality practices may occur as a result of skipping certain stages of software development (e.g. testing) or certain tasks or activities (e.g. unit testing). However, there is a lack of studies providing a comprehensive description of omission instantiations, their exact timings with respect to the stages of software development, or analysis of professionals' key roles in such omission instantiations (e.g. decision-makers vs. implementers). Additionally, the current literature lacks studies analyzing and explaining the nature of omission instantiations with respect to dimensions, such as voluntariness. Therefore, as a first step, a comprehensive description of this phenomenon is needed to motivate future research that attempts to explain different aspects of the omission of quality practices (e.g., reasons) and propose solutions and interventions that aim to prevent omission instantiations or to mitigate their consequences.

What are the psycho-social mechanisms underlying the omission of quality practices? Previous studies mainly suggest that the omission of quality practices occurs as a result of strategic organizational decisions under the influence of different market-level, organizational-level, or human factors. However, there is a lack of studies examining the individual and psychological underpinnings of such questionable behaviors. For that reason, it is not clear why developers decide to omit software development practices while these practices are recommended to improve the overall quality of software. This shortcoming becomes even more meaningful when considering a group of previous studies which argues that software developers have a tendency to develop high-quality software [Austin. 2001, McConnell. 2007, Yang, et al. 2008]. Therefore, to gain a better understanding of this phenomenon, further empirical research is needed to investigate both the psychological and social processes through which developers decide to ignore quality practices. In recent years, such human aspects of software development have received increasing attention from Behavioral Software Engineering [Lenberg, et al. 2015], which is a growing subfield of software engineering research. Drawing from behavioral and social science theories, future research needs to identify psychological processes in developers' cognition and social processes that occur as developers interact with other entities in the development context.

What are the consequences of the omission of quality practices? In our literature review, we identified several short-term and long-term consequences of the omission of quality practices that mainly concern organizations. In the short term, the omission of quality practices might lead to a reduction of development costs and delivery times or even higher levels of customer satisfaction. Alternately, if quality practices are ignored, it might increase software complexity and decrease

software quality in the long term and, as a result, lead to user dissatisfaction, escalating maintenance costs, and financial loss. While such consequences have major importance in today's competitive business environment, the identified primary studies do not provide any clear explanation of how the omission of quality practices might affect software development stakeholders and society beyond such financial factors. Therefore, further research on behavioral and economic aspects of software development is needed to address this gap in the literature.

Future behavioral software engineering research is needed to investigate how the omission of quality practices might affect developers' perceptions of "quality," which eventually influences moral standards and ethics in the software development community and society as well. Furthermore, it is necessary to understand how the omission of quality practices might affect customers' expectations and users' experiences and how such influences might affect the role of information technology in society. Finally, future software economics research [Boehm and Sullivan. 2000], is needed to investigate and explain the consequences of omission of quality practices from an economic perspective. For instance, in their study Slaughter et al. [1998] stated that software quality improvement should be perceived as an investment. They showed that it is possible to assess the cost of conformance (i.e. amount spent on achieving quality products) and non-conformance costs (i.e. expenses incurred when thing go wrong) for the optimization of the total cost of software quality development. Based on their findings we speculate that initiatives targeting to the behavior of omitting quality practices might entail to reduction of costs for software stakeholders and societies.

How to consider omissions of quality practices? Our current wisdom is that the omission of quality practices is ultimately an unwanted behavior that is caused by contextual constraints (e.g. lack of resources) and contradictory stakeholders' concerns (e.g. reducing costs vs. increasing quality) within software projects. Therefore, future intervention research that aims to change the attitudes or the underlying values of software development is needed to develop novel means and solutions to prevent omission instantiations or at least mitigate their negative consequences. To prevent the omission behavior, future research needs to identify and propose methods or programs by promoting and improving a quality culture among software community. On the other hand, in situations where omission of quality practices might be unavoidable, software development teams must mitigate the negative consequences of omission behavior by identifying the best possible trade-offs according to their development context. To identify such context-dependent solutions, software developers can draw on observations made in and lessons learned from previous software projects as well as adapting best practices and tools recommended by software community. However, considering the tremendous amount of software projects and available practices and tools, identifying and creating the most suitable solution becomes challenging itself. Therefore, to address this issue, future research must provide theoretically sound and empirically proven recommendation to enable developers in creating their own solutions.

Conclusions

Despite the significant amount of resources that have been spent in software development projects, problems in software are widely reported in the research and practice literature. Recent literature hints that software deficiencies might be the result of omission of quality software

development practices. In this paper, our goal was twofold: first, to discover the state of research on the omission of quality practices and to understand the extent to which this phenomenon has been investigated previously; and second, to determine the root causes underlying the omission of quality practices as suggested by previous studies.

To reach these goals we conducted a systematic literature review and produced a synthesis of our findings. We identified five categories of factors underlying the omission of quality practices. Each of these categories, which are originated from different levels of context, affects the omission of quality practices. In the market level, specific characteristics of the business environment, such as highly competitive and turbulent markets, put development teams under pressure or encourage them to gain competitive advantages through the omission of quality practices. In the organizational level, different factors, including available resources and technical obstacles, might create conditions under which developers decide to omit quality software development practices. Finally, in the individual level, human factors, such as attitudes and cognitive tendencies, under the influence of market- and organizational-level factors might push managers and developers to neglect quality practices.

The results of this study shows that the current literature does not consider the omission of quality practices adequately with respect to why and how software developers make the decision to omit a quality practice and how to address this phenomenon in practice. Even though, based on the analysis of primary studies, we hypothesize that contradictory contextual factors trigger a psycho-social process pertaining to omission instantiations, further empirical research is needed to provide an in-depth understanding of mechanisms underlying this process and its outcomes. To reach this goal, we have proposed several avenues for future research to develop knowledge on the omission of quality practices.

The first research area concerns the determination of instantiations of the omission of quality practices, their timing with respect to stages of software development, tasks they relate to, and the nature of those instantiations. This information is needed to motivate further study to explain omission behavior and to target the interventions that aim to prevent omission instantiations to correct stages and tasks of software development. Alternately, the second research area concerns revealing the psychosocial process of decision-making regarding omission behavior. It is necessary to investigate the psychological processes in developers' cognition and social processes that occur as developers interact with other entities in the development context. The third research area concerns the consequences of the omission of quality practices. Further research is needed to investigate how such omission practices might affect developers' perceptions of "quality practices" and, consequently, moral standards and ethics within the software development community and society as well. Furthermore, it is necessary to understand how the omission of quality practices might affect customers' expectations and users' experiences and how such influences might affect the role of information technology in society. Finally, the fourth area concerns possible solutions for considering the omission of quality practices. Our current wisdom is that the omission of quality practices is undesirable behavior and that there is therefore a need to develop novel means (e.g., guidelines, methods, culture) to prevent omission instantiations.

Appendix A. List of the Primary Studies (PS)

ID	Title	Year	Author(s)
PS1	A Balancing Act: What Software Practitioners Have to Say about Technical Debt	2012	Lim, Taksande and Seamn
PS2	A case study on quality-affecting problems in software engineering projects.	2003	Ahonen and Junttila
PS3	An Exploratory Study on Self-Admitted Technical Debt	2014	Potdar and Shihab
PS4	Architecture Technical Debt: Understanding Causes and a Qualitative Model	2014	Martini, Bosch, and Chaudron
PS5	Attitude towards testing: a key contributor to software quality	1994	Murugesan
PS6	Avoiding classic mistakes [software engineering]	1996	McConnel
PS7	Global software testing under deadline pressure: Vendor-side	2013	Shah, Harrol, and Sinha
PS8	Influence of confirmation biases of developers on software quality: an empirical study	2013	Çalikli and Bener
PS9	Managing technical debt: An industrial case study	2013	Codabux and Williams
PS10	Organizational and customer related challenges of software testing: An empirical study in 11 software companies	2014	Seth, Taipale and Smolander
PS11	Penalty policies in professional software development practice: a multi-method field study	2010	Wang and Zhang
PS12	Short cycle time systems development	2004	Baskerville and Pries-Heje
PS13	Technical Debt and the Effect of Agile Software Development Practices on It - An Industry Practitioner Survey	2014	Holvitie, Leppänen, and Hyrynsalmi
PS14	The effects of time pressure on quality in software development: An agency model	2001	Austin
PS15	Toward objective software process information: experiences from a case study	2011	Samalikova, Kusters, Trienekens, Weijters, and Siemons
PS16	A Fresh Perspective on Old Problems	1999	Fleming
PS17	A Method for Balancing Short- and Long-Term Investments: Quality vs. Features	2008	Lindgren, Wall, Land, and Norström
PS18	Recognizing and Responding to “Bad Smells” in Extreme Programming	2002	Elssamadisy and Schalliol
PS19	A View-based Approach for Improving Software Documentation Practices	2006	Bayer & Muthig

References

Manish Agrawal and Kaushal Chari. 2007. Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects. *IEEE Transactions on Software Engineering* 33, 3, 145–156.

Jarmo J. Ahonen and Tuukka Junttila. 2003. A Case Study on Quality-Affecting Problems in Software Engineering Projects. In *Proceedings of IEEE International Conference on Software: Science, Technology and Engineering (SwSTE'03)*. IEEE Press, 145–153.

Robert D. Austin. 2001. The Effects of Time Pressure on Quality in Software Development: An Agency Model. *Information Systems Journal* 12, 2, 195–207.

Rajiv D. Banker, Gordon B. Davis, and Sandra A. Slaughter. 1998. Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. *Management Science* 44, 4, 433–450.

Richard Baskerville, Linda Levine, Jan Pries-Heje, and Sandra A. Slaughter. 2001. How Internet Software Companies Negotiate Quality. *IEEE Computer* 34, 5, 51–57.

Richard Baskerville and Jan Pries-Heje. 2004. Short Cycle Time Systems Development. *Information Systems Journal* 14, 3, 237–264.

Richard Baskerville, Balasubramaniam Ramesh, Linda Levine, Jan Pries-Heje, and Sandra A. Slaughter. 2003. Is Internet-Speed Software Development Different? *IEEE Software* 20, 6, 70–77.

Joachim Bayer and Dirk Muthig. 2006. A view-based approach for improving software documentation practices. In *IEEE International Conference on the Engineering of Computer-Based Systems*. IEEE Computer Society, 269–278.

Kent Beck. 2000. Extreme programming explained: embrace change. Addison-wesley professional.

Paul E. Black. 2012. Static Analyzers: Seat Belts for Your Code. *IEEE Security and Privacy* 10, 3, 48–52.
Barry W. Boehm and Kevin J. Sullivan. 2000, Software economics: a roadmap. In *Proceedings of the conference on The future of Software engineering*. ACM, 319–343.
Frederick P. Brooks. 1995. *The Mythical Man-Month: Essays on Software Engineering* (Anniversary Edition ed.) Addison-Wesley, MA, USA.
Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, et al. 2010. Managing Technical Debt in Software-Reliant Systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 47-52.
Gül Çalıklı and Ayşe Başar Bener. 2013. Influence of Confirmation Biases of Developers on Software Quality: An Empirical Study. *Software Quality Journal* 21, 2, 377–416.
Zadia Codabux and Byron Williams. 2013. Managing Technical Debt: An Industrial Case Study. In *Proceedings of the 4th International Workshop on Managing Technical Debt*. IEEE Press, 8–15.
Ward Cunningham. 1992. The WyCash Portfolio Management System. Addendum to the *Proceedings on Object-Oriented Programming Systems, Languages, and Applications*. British Columbia, Canada: 29-30.
Amr Elssamadisy and Gregory Schalliol. 2002. Recognizing and responding to bad smells in extreme programming. In *Proceedings of the 24th International conference on Software Engineering*. ACM, 617-622.
Ryan Fleming. 1999. A fresh perspective on old problems. *IEEE Software* 16, 1, 106-113.
Jose Fonseca and Marco Vieira. 2008. Mapping Software Faults with Web Security Vulnerabilities. In *Proceedings of IEEE International Conference on Dependable Systems and Networks With FTCS and DCC(DSN 2008)*. 257–266. DOI: 10.1109/DSN.2008.4630094
Martin Fowler and Jim Highsmith. 2001. The agile manifesto. *Software Development* 9, 8, 28-35.
Steven Fraser and Dennis Mancl. 2008. No Silver Bullet: Software Engineering Reloaded. *IEEE Software* 25, 1, 91-94.
Virginia R. Gibson and James A. Senn. 1989. System Structure and Software Maintenance Performance. *Communications of the ACM* 32, 3, 347–358.
Johannes Holvitie, Ville Leppanen, and Sami Hyrynsalmi. 2014. Technical Debt and the Effect of Agile Software Development Practices on it-an Industry Practitioner Survey. In *Proceedings of 6th International Workshop on Managing Technical Debt (MTD)*. IEEE, 35–42.
Ivar Jacobson and Ed Seidewitz. 2014. A new software engineering. *Communications of the ACM* 57, 12, 49–54.
Ken H. Judy. 2009. Agile principles and ethical conduct. In*42nd Hawaii International Conference on System Sciences, 2009. HICSS'09*. IEEE, 1-8.
Barbara Kitchenham and Stuart Charters. 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. *EBSE Technical Report EBSE-2007-01*. 1–57.
Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. 2010. Systematic Literature Reviews in Software Engineering–a Tertiary Study. *Information and Software Technology* 52, 8, 792–805.
Per Lenberg, Robert Feldt, and Lars G. Wallgren. 2015. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and Software* 107, 15-37.
Nancy G. Leveson and Clark S. Turner. 1993. An Investigation of the Therac-25 Accidents. *IEEE Computer* 26, 7, 18–41.
Erin Lim, Nitin Taksande, and Carolyn Seaman. 2012. A Balancing Act: What Software Practitioners have to Say about Technical Debt. *IEEE Software* 29, 6, 22–27.
Kurt R. Linberg. 1999. Software Developer Perceptions about Software Project Failure: A Case Study. *Journal of Systems and Software* 42, 9, 177–192.
Dennis Mancl, Steven D. Fraser, and William F. Opdyke. 2007. No Silver Bullet: A Retrospective on the Essence and Accidents of Software Engineering. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications*. ACM, 758–759.
Robert C. Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
Antonio Martini, Jan Bosch, and Michel Chaudron. 2014. Architecture Technical Debt: Understanding Causes and a Qualitative Model. In *Proceedings of 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Press, 85–92.
Steve McConnell. 1996. Avoiding Classic Mistakes. *IEEE Software* 13, 5, 111–112.
Steve McConnell. 2007. Technical Debt. (November 2007). Retrieved April 1, 2015 from http://www.construx.com/10x_Software_Development/Technical_Debt/.
San Murugesan. 1994. Attitude Towards Testing: A Key Contributor to Software Quality. In *Proceedings of First International Conference on Software Testing, Reliability and Quality Assurance*. IEEE Press, 111–115.
Chitu Okoli and Kira Schabram. 2010. A Guide to Conducting a Systematic Literature Review of Information Systems Research. (May 2010). Available at SSRN: http://ssrn.com/abstract=1954824 or http://dx.doi.org/10.2139/ssrn.
Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michaek Mattsson. 2008. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering*, 1-10.
Aniket Potdar and Emad Shihab. 2014. An Exploratory Study on Self-Admitted Technical Debt. *In Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Press, 91–100.
Alexander Poth and Ali Sunyaev. 2014. Effective Quality Management: Risk-and Value-Based Software Quality Management. *IEEE Software* 31, 6, 79–85.
Frantz Rowe. 2014. What Literature Review is Not: Diversity, Boundaries and Recommendations. *European Journal of Information Systems* 23, 3, 241–255.
Jana Samalikova, Rob Kusters, Jos Trienekens, Ton Weijters, and Paul Siemons. 2011. Toward Objective Software Process Information: Experiences from a Case Study. *Software Quality Journal* 19, 1, 101–120.

Ken Schwaber and Mike Beedle. 2001. *Agile Software Development with Scrum*. Prentice Hall PTR, NJ, USA.

Frank P. Seth, Ossi Taipale, and Kari Smolander. 2014. Organizational and Customer Related Challenges of Software Testing: An Empirical Study in 11 Software Companies. In *Proceedings of 8th IEEE International Conference on Research Challenges in Information Science (RCIS)*. IEEE Press, 1–12.

Hina Shah, Mary Jean Harrold, and Saurabh Sinha. 2014. Global Software Testing Under Deadline Pressure: Vendor-Side Experiences. *Information and Software Technology* 56, 1, 6–19.

Sandra A. Slaughter, Donald E. Harter, and Mayuram S. Krishnan. 1998. Evaluating the cost of software quality. *Communications of the ACM*, 41, 8, 67–73.

Ian Sommervile. 2011. *Software Engineering*. (9th ed.). Addison-Wesley, Boston, USA.

Gregory Tassey. 2002. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*, RTI Project 7007.011. Retrived April 1, 2015 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.3316&rep=rep1&type=pdf

Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An Exploration of Technical Debt. *Journal of Systems and Software* 86, 6, 1498–1516.

Eva Van Emden and Leon Moonen. 2002. Java Quality Assurance by Detecting Code Smells. In *Proceedings of 9th Working Conference on Reverse Engineering*. IEEE Press, 97–106.

Yi Wang and Min Zhang. 2010. Penalty Policies in Professional Software Development Practice: A Multi-Method Field Study. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. ACM Press, 39–47.

Dumidu Wijayasekara, Milos Manic, Jason L. Wright, and Miles McQueen. 2012. Mining Bug Databases for Unidentified Software Vulnerabilities. In *Proceedings of 5th International Conference on Human System Interactions (HSI)*. IEEE Press, 89–96.

Bo Yang, Huajun Hu, and Lixin Jia. 2008. A Study of Uncertainty in Software Cost and its Impact on Optimal Software Release Time. *IEEE Transactions on Software Engineering* 34, 6, 813–825.