Received May 6, 2022, accepted June 2, 2022, date of publication June 13, 2022, date of current version June 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3182703

Iteration Causes, Impact, and Timing in Software Development Lifecycle: An SLR

MAMOONA MUMTAZ[1], NAVEED AHMAD[2], M. USMAN ASHRAF[3], AHMED MOHAMMED ALGHAMDI[4], ADEL A. BAHADDAD[5], AND KHALID ALI ALMARHABI[6]

[1]Department of Software Engineering, University of Management and Technology, Sialkot 53310, Pakistan
[2]Faculty of Computing, National University of Computer and Emerging Sciences (FAST), Islamabad 44000, Pakistan
[3]Department of Computer Science, GC Women University Sialkot, Sialkot 51310, Pakistan
[4]Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, Jeddah 21493, Saudi Arabia
[5]Department of Information System, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[6]Department of Computer Science, College of Computing in Al-Qunfudah, Umm Al-Qura University, Makkah 24381, Saudi Arabia

Corresponding authors: Mamoona Mumtaz (mamona.mumtaz@skt.umt.edu.pk) and M. Usman Ashraf (usman.ashraf@gcwus.edu.pk)

This work was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, Saudi Arabia, under Grant RG-11-611-43.

ABSTRACT Context: Iteration—performing an activity once it has already been done—is unavoidable and omnipresent during software development. Management of iteration is a challenging task due to the lack of detailed analysis and use of different terms for the iteration at different places in software engineering. Objective: In order to manage iteration in a better way during software development processes, we investigate different iterative situations, the causes, the stages at which it can occur during the development, and its impacts. Method: We use the systematic literature review (SLR) method to search using six bibliographic databases. The SLR includes 153 primary studies published from 2007 to February 2017. Results: We identify twenty-two leading causes, five stages, and positive and negative impacts of iteration. Then, we develop a lucid taxonomy of iteration perspectives based on the causes and timing at which it occurs during the software development lifecycle. Conclusions: The frequently reported causes of iteration are defects, code smell, and conflicts, whereas the least referenced causes are poor management and different methods followed by teams. The most cited phase at which iteration occurs during the software development is maintenance. The most cited positive consequence of iteration is quality improvement, whereas the negative impacts of iteration are increasing time, effort, and cost. Our study provides a framework to understand the nature of iteration, what sources can lead to which iterative perspective, a particular iterative situation can have what kind of impacts on project milestones, and also provide research directions.

INDEX TERMS Artificial intelligence, iteration, software engineering, software development lifecycle, impact, causes, taxonomy.

I. INTRODUCTION

Software engineering is about translating user needs into a software product. It involves converting user needs into requirements, interpreting the requirements into design, coding, and testing the code to make the software function according to a user's demands [1]. These activities do not move linearly during software development. Deviation from a linear movement of activities is unavoidable during software development and is defined as iteration [2]. Most of the problems that occur during software development are complex and involve many contradictory goals that require iteration to make a compromise [3]. sometimes, We plan to work in increments during software development as we do in agile and iterative software development [4], [5]. Iteration can also be spontaneous, i.e., unplanned. When we compare the planned iteration with the spontaneous one, the random unplanned iterations are expensive and may influence project results [6].

There are numerous causes that lead to unplanned iteration, like defects [7]–[9], complexity [10], ambiguity [11], [12], user feedback [4], conflicts [11]–[13], and requirement changes [14], [15]. According to a study, unaligned plans of different participants, poor management, frequent changes, and defects cause delays and iteration [16]. Sometimes, wrong priority assignment leads to rework [17]. Iterative

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

prototyping is a prime activity during user-centered software development [18]. Different design options are explored at the start of the project and during architectural design during software development. For example, the negotiated decisions are considered better with different stakeholders' concerns and alternative choices [17]. Whenever a change occurs due to any reason, it causes iteration linked with delays, extra cost, additional efforts, and risks. The reason for redoing can be defects, unexpected errors, or ambiguous user requirement specifications. Another study suggests that in the embedded system development, adjustment of the requirement is made in later phases. It causes unplanned and expensive iterations for making many adjustments to deal with defects at this stage [19].

Detailed studies of iteration exist in product development [20]–[23], construction [24], design [25]–[29], and engineering disciplines [25], [28], [30], [31], whereas very few authors discuss them from software development viewpoint [6]. A study contribute by gathering and summarizing insights into iteration. They also create a comprehensive taxonomy of iteration in design and development discipline with the selection of a few articles from software engineering literature [29]. Another study contribute by modeling iteration in software engineering and simulating its impact on project completion time but it lacks detailed analysis of different iterative situation [6].

Iteration is unavoidable and part of the software development process. Despite the iterative nature of the software development life cycle, the least attention is paid to the detailed iteration analysis in software engineering. Research literature addresses different benefits and weaknesses of iteration that are ambiguous for the readers. For project managers and researchers, it is often impossible to understand the context. The problem gets complex by using vague terms to refer to iteration, sources, and impacts at different places. In this way, the management of iteration becomes a challenging task. Though iteration is ubiquitous during software development, a detailed analysis does not exist. Therefore, the management of iteration is a difficult task.

This article analyzes current knowledge about iteration in the software development discipline to understand iteration by conducting an SLR. We investigate the sources of iteration, impacts of an iterative situation, and timings of the development cycle at which it can occur. The contribution of this study is the development of a lucid taxonomy of iteration in software engineering, which describes that the different perspectives on iteration have different sources, timings of occurrence, and impacts. Furthermore, this study also demonstrates how taxonomy can be based on literature and used to map present studies. It is a step toward analyzing iteration in software engineering to understand different iterative situations that highlight new research directions.

The remainder of this paper continues in eight sections. Section II presents the research goal, questions, methodology to conduct SLR, and quality assessment of the studies. Next, statistical findings from selected primary studies are described in Section III. Section IV discusses the different sources that lead to iteration during software development. Stages at which iteration occurs during SDLC are discussed in Section V. Section VI presents the positive and negative consequences of iteration which exist in the software engineering literature. In section VII, we explain the proposed iteration taxonomy and its validation based on the finding of SLR. Threats to this review are explained in Section VIII. Finally, conclusions and future directions are presented in Section IX.

II. RESEARCH METHODOLOGY

The research was conducted through a systematic literature review. According to the guidelines, the SLR process comprises of three core phases, i.e., planning, conducting, and reporting [32]. The methodology followed, and the activities performed in each phase of the SLR process are shown in Figure 1.

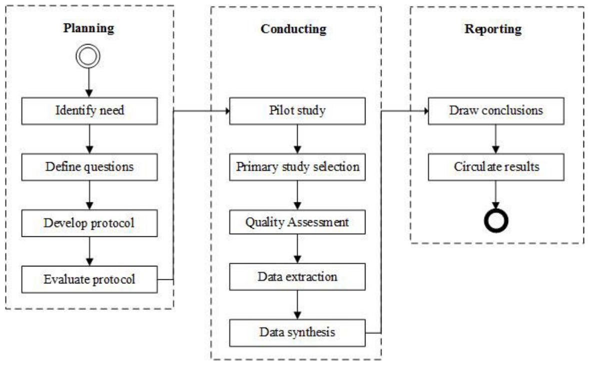


FIGURE 1. Systematic Literature Review (SLR) methodology adapted from [32].

The first step in the methodology (Figure 1) is describing the need for the SLR. This SLR was driven by the lack of a detailed analysis of iteration in software engineering. Iteration is omnipresent during software development [2], [17], [33]–[36]. Despite the iterative nature of software development, very little attention is paid to iteration in software engineering. None of the publications in software engineering has treated iteration as the main subject. The use of diverse terms to refer to iteration at different places in the literature also contributes to this SLR's need.

We searched ACM, IEEE, Science Direct, Scopus, Google Scholar, and Springer digital libraries in September 2016. In the retrieved studies, the authors do not explicitly mention different iteration perspectives. A detailed, in-depth analysis was conducted to get insights into iteration.

A. RESEARCH QUESTIONS

This SLR aims to investigate the different perspectives of iteration that exist in software engineering. In order to define different perspectives and create a taxonomy of iteration in the software engineering discipline, the stage at which iteration can occur, the cause or source of iteration, and its

TABLE 1. Research questions.

#	Question
RQ1	What is currently known about iteration in software engineering?
RQ1.1	What are the sources of iteration?
RQ1.2	What are the stages at which iteration can occur during software development lifecycle?
RQ1.3	What can be the impact of an iteration in software development?
RQ1.4	Can iteration in software engineering be categorized based on its source and timing?

impacts in the development cycle need to be determined. This is reflected in one primary and four secondary research questions described in Table 1 (Step2, Figure 1).

B. SEARCH STRATEGY

In developing the review protocol (Step 3, Figure 1), the search strategy is defined to include related studies and exclude unrelated ones. The strategy involved an automatic search using a string validated by the authors of the present article. Search in digital libraries was performed using search string. The string was specified using keywords derived from research questions. We identified alternative spellings and synonyms for the keyword to compose the search string. Our search string comprised of three parts: *iteration, software development*, and *lifecycle*. Keywords are connected through AND operator and synonyms are connected through OR operator.

Automatic searches consisted of the web search of the most relevant indexed databases. The digital libraries searched and results returned are given in Table 2. We refined the search string through a pilot study and omitted the synonyms, which did not influence the paper returned via searches. We come to the following string after refining the string through several iterations.

C. INCLUSION AND EXCLUSION CRITERIA

For a study to be included in the review as a primary study, it must fulfill the inclusion criteria summarized in Table 3, and it must not meet exclusion criteria presented in Table 4. We were concerned only in primary studies, between 2007 to February 2017, which are on software development lifecycle or any phase of the development process and present some contribution on the iteration during development. Validation of the review protocol (Step4, Figure 1) is performed by the authors of the article.

```
(Iteration OR Rework OR Refinement OR Refactoring
OR Negotiation OR Exploration)
AND
(Agile software development OR Software engineering
OR software project)
AND
(Lifecycle OR process OR activities)
```

TABLE 2. Electronic sources used for literature search.

Database	Search Results
ACM Digital Library	172
IEEE Xplore	507
ScienceDirect	678
Scopus	185
Google Scholar	425
SpringerLink	105
Total	**2072**

TABLE 3. Inclusion criteria.

#	Inclusion criteria
1	Primary studies
2	Studies published from 2007 until February 2017
3	Studies from both qualitative and quantitative research
4	Studies focused on any stage of software development life cycle
5	Academics and industry studies
6	Studies that show experience report on software development
7	Studies on software development
8	Studies written in English

TABLE 4. Exclusion criteria.

#	Exclusion criteria
1	Secondary studies
2	Studies which was earlier than 2007
3	Studies which was not focusing on software development
4	Studies having less than three pages
5	Studies written in other languages except English
6	Studies which where not available through search engines
7	Studies which was about software development but did not contain any information about iteration
8	Books, book chapters, Symposium, and news letter on software development processes
9	Redundant studies
10	Literature reviews

D. STUDIES SELECTION PROCESS

Primary studies were selected by following four stages process presented in Figure 2. Each stage of the study selection process is described in the following paragraphs.

In stage 1, automatic search using search string was performed to obtain studies from digital libraries. A total of 2072 search results were received, among which Google scholar returned 425 studies, IEEE Xplore 507, Scopus 185, Science Direct 678, ACM 172, and Springer Link 105 results.

Out of 2072 studies, 1391 studies were discarded at stage 2. At this stage, the exclusion of unrelated studies was performed based on title, abstract, publication venue, and keywords analysis. If there was a doubt about any study, including or excluding, it was added for later consideration. After this stage, 681 studies remained, which were downloaded and organized.

Selected studies from the last stage were revised by reading the introduction, conclusion, and discarded book chapters, newsletters, dissertations, short papers with a length of fewer

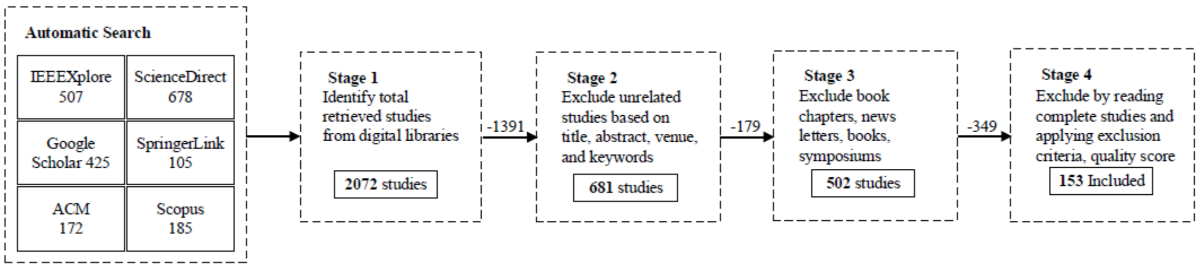

Automatic Search
IEEEXplore 507
ScienceDirect 678
Google Scholar 425
SpringerLink 105
ACM 172
Scopus 185
Stage 1 Identify total retrieved studies from digital libraries
2072 studies
-1391
Stage 2 Exclude unrelated studies based on title, abstract, venue, and keywords
681 studies
-179
Stage 3 Exclude book chapters, news letters, books, symposiums
502 studies
-349
Stage 4 Exclude by reading complete studies and applying exclusion criteria, quality score
153 Included

FIGURE 2. **Study selection process.**

TABLE 5. **Data extraction form.**

#	Information extracted	Description	Concern/RQ
1	ID	unique study identifier	Overview
2	Title,Authors		Overview
3	Year, Geographic location		overview
4	Source	Publisher, Google scholar, IEEE Xplore, Scopus, Science Direct, ACM, SpringerLink	Overview Overview
5	Type	Journal, conference	
6	Citation	Number of citations	Overview
7	Research type [37]	Quantitative evaluation/validation, qualitative evaluation/validation, solution proposal, and experience/opinion	Overview
8	Iteration confirmation	What is currently known about iteration in software engineering?	RQ1
9	Iteration Source	What are the sources or causes of an iteration?	RQ1.1
10	Iteration Stage	What are the stages at which iteration can occur during software development lifecycle?	RQ1.2
11	Iteration impacts /consequences	What can be the impact of an iteration in software development?	RQ1.3

than three pages, and symposiums. This stage excluded 179 studies leaving 502 studies in the selection.

Three hundred forty-nine studies were discarded in the last stage by applying detailed inclusion-exclusion criteria and reading the full text. Afterward, 158 studies were obtained in the selection, which was then subjected to quality assessment. Five studies having low-quality scores were discarded, leaving 153 primary studies.

E. DATA EXTRACTION AND SYNTHESIS

We included just journal articles and conference papers in this study. The selected primary studies were categorized into quantitative evaluation or validation, qualitative evaluation or validation, solution proposal, and experience or opinion study. A data extraction form shown in Table 5 was used to record details about the study overview and to answer the research questions. To extract information presented in Table 5 a detailed analysis of all the 153 selected primary studies has been conducted. A spreadsheet editor (Microsoft Excel) was used to record all the extracted data.

Data extraction involved all of the authors of the present article. One of the authors extracted the data, and others reviewed it based on convenience. There were disagreements related to some studies, which were resolved by discussion.

According to a study

> Data synthesis involves collating and summarizing the results of the included primary studies [32].

During the data synthesis, we extracted and normalized the terms used to refer to iteration, its source, the stage at which it occurs, and its consequences during software development. We built a comprehensive taxonomy of iteration based on its cause and stage of occurrence.

F. QUALITY ASSESSMENT

Quality assessment (QA) is critical in SLR [32]. The selected studies were evaluated against a set of 30 quality criteria given in Table 6.

To assess the quality of primary studies, we categorized the 153 studies into four categories. The four categories are Quantitative evaluation/validation (Qnt), Qualitative evaluation/validation (Qlt), Solution (Sol), and Experience/opinion (Exp/Op) studies. Each study was assessed against different quality criteria based on the category it belongs to. We used 16 quality assessment questions for Qlt papers, 19 for Qnt, 9 for Sol, and 6 for Exp/Op studies as presented in Table 6 and proposed by previous studies [32], [37]–[41].

Each question is graded on the scale of three optional answers: yes=1, no=0, or partial=0.5. Thus, a quality score for a study is calculated by adding the scores of all the

TABLE 6. Quality assessment criteria for primary studies.

Questions	Qlt	Qnt	Sol	Exp/Op	Ref
Q1. Is the goals of the research clearly stated?(Is the clear statement of the problem? Is the technique to be validated clearly described?)	✓	✓	✓	✓	[32, 38] [39, 40] [37, 41]
Q2. Is the research method sound? Is the research method appropriate to address the aims of research? How defensible is research design? How adequately has the research process been documented?	✓	✓			[37, 39]
Q3. Is the knowledge claim made by the paper novel (is there a significant increase of the knowledge of situation)? Is the technique novel (is the application of technique to this kind of problem is novel)?	✓	✓	✓		[32, 37]
Q4. Is there sufficient discussion of related work (are competing techniques discussed and compared with this one)?	✓	✓	✓		[32, 38] [37, 39]
Q5. Is the problem to be solved by the technique clearly explained?			✓		[37]
Q6. Is the technique sufficiently well described so that author or others can validate it in later research?			✓		[37, 39]
Q7. Is the technique/framework/report about experience is sound?			✓	✓	[37, 39]
Q8. Are the study participants and observational units adequately described?	✓	✓			[32, 38]
Q9. Is the technique intended use/working explained with example?			✓		[37]
Q10. Is the experience original?				✓	[37]
Q11. Is the report surprising and provoke discussion (is it revealing)?				✓	[37, 39]
Q12. Is the study relevant for practitioners?	✓	✓	✓	✓	[32, 38] [37, 39]
Q13. Is the report explicitly contains lesson learned by the author?				✓	[37, 38] [39]
Q14. Is there an adequate description of the context in which research carried out?		✓			[38, 41]
Q15. Was data analysis sufficiently rigorous? How well have detail, depth, and complexity (i.e., richness) of the data been conveyed? Do the researchers explained the data types?		✓			[38, 39]
Q16. Is there discussion about results of the study?	✓	✓			[32, 38] [41]
Q17. Are the limitations of the study explicitly discussed?	✓	✓	✓		[37, 38] [40, 41]
Q18. If the study involves assessment of technology, is the technology clearly defined?		✓			[32]
Q19. Are the variables used in the study fully defined, adequately measured, and most relevant ones?		✓			[32, 40]
Q20. Are the data collection method adequately described? How well was the data collection carried out?	✓	✓			[32, 39] [40]
Q21. Was real data set used in evaluation/validation?		✓			[32, 40]
Q22. Are statistical methods used to analyze data described? If statistical tools are used, is practical significance discussed?		✓			[32]
Q23. Is the data, code etc used in evaluation accessible or available online?		✓			[32]
Q24. Is there clear statement of findings? (Are negative findings presented?)	✓	✓			[32, 39]
Q25. How credible and important are the findings?	✓				[32]
Q26. How well does the evaluation address its original aims and purpose?	✓	✓			[32]
Q27. How well has diversity of perspective and context been explored? How clear are the theoretical perspectives that have shaped the form and output of the evaluation?	✓				[32, 41] [39]
Q28. How clear and lucid is the reporting of study?	✓				[32]
Q29. How clear are the links between data, interpretation, and conclusions?	✓				[32]
Q30. Is the study explicitly mentioned the assumptions that have been taken?	✓	✓			[32, 41]

answers to quality criteria questions. The quality assessment scores for all the primary studies are presented in Table 12.

III. STATISTICAL FINDINGS

This section presents and summarizes this systematic review. In total, we selected the 153 primary studies, and we extracted data from these studies using Table 5. In the following subsections, we start by presenting quality assessment results and an overview of selected primary studies. The answers to each research question are described in the later sections.

A. QUALITY ASSESSMENT RESULTS

The selected studies fall into four categories, i.e., qualitative, quantitative, solution proposal, and experience/opinion. Different types of studies cannot be assessed according to the same criteria [32], [37]. So, we formulated different assessment questions for each study type by following guidelines of [37], [38], [40], [41]. Table 12 presents the quality assessment score for all the selected primary studies based on the quality assessment checklist given in Table 6. The average overall quality of the selected primary studies is 80.56, which is quite reasonable.

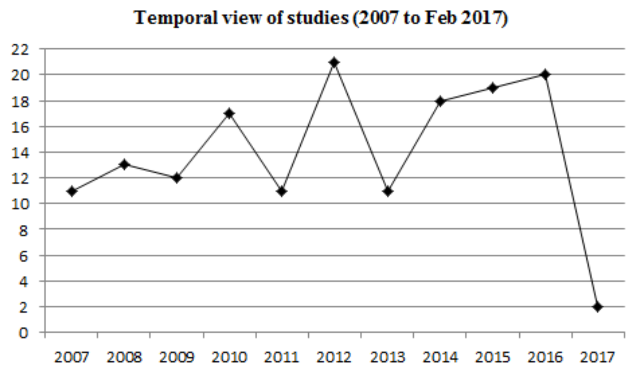


FIGURE 3. Studies included in SLR by year of publication.

B. STUDIES OVERVIEW

Our SLR included the studies published from 2007 to Feb 2017. Temporal and geographical view of the selected primary studies are shown in Figure 3 and 4 respectively.

TABLE 7. Study types.

Type	Studies	Count	Percentage
Solution	[3, 19, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68] [69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95] [7, 8, 10, 11, 12, 13, 14, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115]	83	54.25%
Eva/Val (Quantitative)	[3, 7, 13, 51, 52, 53, 54, 55, 57, 58, 59, 60, 99, 101, 102, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129] [61, 62, 68, 82, 86, 90, 91, 110, 111, 114, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143]	53	34.64%
Eva/Val (Qualitative)	[16, 17, 18, 19, 33, 34, 36, 52, 56, 89, 131, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154] [4, 9, 108, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164]	34	22.22%
Experience/ Opinion	[165, 166, 167, 168, 169, 170, 171, 172] [173, 174, 175, 176]	12	7.84%

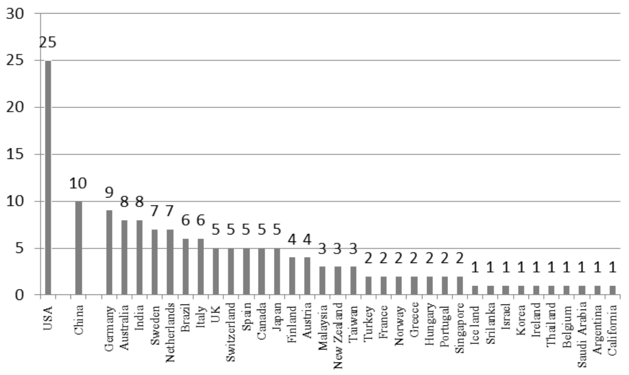


FIGURE 4. Geographical view of the studies included in SLR.

We grouped the selected primary studies into four categories demarcated by [37]. Table 7 describes the research types of selected studies.

The majority of the studies are in the solution category with 83 studies (54.25%) followed by the quantitative evaluation/validation category with 53 studies (34.64%), qualitative evaluation/validation category with 34 studies (22.22%), and 12 studies (7.84%) in experience/opinion category.

IV. SOURCES OF ITERATION

This question aims to find the reasons that lead towards iteration during software development. We answer this research question by reading 153 studies returned by our SLR, given in Table 12. Sources of iteration gained from literature are summarized in Table 8 and explained in the following paragraphs.

The authors are not explicitly mentioning the causes of iteration in these primary studies. From a detailed analysis of these articles, we extracted different sources for the iteration. Due to many extracted causes, we grouped those into 22 leading causes. The most cited causes are conflicts, defects, and code smells, whereas the least referenced causes are poor management and different development methods followed by teams. Conflicts, referenced in 21 studies (13.56%), 28(18.06 %) studies cited errors/defects/faults cause for the iteration, code and design smell referenced by 23 studies (14.08 %), while 1 study (0.65%) cited poor management and 2 studies (1.29%) cited different methods followed by different teams cause.

A. CONCEPTUAL GAPS AND AMBIGUITIES

During software development, conceptual gaps and ambiguities become a reason to iterate. It includes vague and unclear requirements, imprecise and incomplete information, large and poorly defined features, missing the detailed requirement, ambiguity, and assumptions. Due to any one of the reasons mentioned above, either an enhancement occurs, leading to reattempting the task.

B. CONFLICTS

Conflict includes confusion due to the disparate industrial background of teams, contradictory requirements, conflicting requirements, different views and opinions of different stakeholders, mismatching goals, different levels of understanding of team members, uncertain and contradictory goals, and multiple quality objectives. Whenever there is any conflict during development, iteration occurs to resolve that conflict and get a mutual agreement.

C. UNCERTAIN DECISIONS

Uncertainty and inconsistency in early decisions lead to bad decisions, e.g., taking poor or incorrect architecture structure as a blueprint, reassessments, and recalculating schedules. During software development, an uncertain and lousy decision is a source to perform a task again.

D. ERRORS, FAULTS, DEFECTS

Iteration occurs during software engineering processes due to unexpected errors, faults, misunderstandings, developers' mistakes, unclear specifications, problems discovered in the later phases, errors, defects. Cognitive bias leads to reasoning errors and becomes a cause of iteration. Defects in the requirement lead to incomplete, inconsistent, inadequate, and ambiguous requirements, which become a source of iteration.

E. COSMETIC CHANGES AND ENHANCEMENTS

When software development products are of aesthetic appeal or assessment criteria are subjective, there are lots of cosmetic changes that occur during development. These cosmetic

TABLE 8. Sources of iteration.

Causes	Studies	Count
Errors,faults, defects	[7, 12, 13, 14, 115][8, 33, 36, 46, 49, 51, 124, 125, 141, 144, 145, 149] [64, 90, 95, 105, 109, 110, 129, 130, 133, 161, 162]	28
Code and design smells	[7, 48, 53, 55, 57, 63, 68, 76, 86, 103, 112, 121, 124, 128, 134, 135, 138, 155, 158, 162, 167, 176, 177]	23
Conflicts	[10, 11, 12, 13, 17, 19, 42, 50, 58, 70, 87, 96, 98, 113, 114, 116, 148, 154, 166, 174, 178]	21
Bugs	[9, 59, 62, 91, 106, 117, 120, 126, 134, 142, 147, 156, 167, 173, 176]	15
Cosmetic changes and enhancements	[8, 9, 64, 65, 78, 89, 90, 91, 106, 117, 122, 145, 147, 177]	14
Corrective and non-corrective changes	[36, 47, 60, 61, 99, 101, 105, 123, 127, 140, 149, 150, 151, 160]	14
Complexity	[3, 10, 72, 79, 91, 107, 123, 124, 161, 162, 168, 179]	12
Conceptual gaps and ambiguities	[11, 12, 47, 54, 59, 67, 79, 111, 136, 166, 172]	11
Technical Debt	[33, 34, 52, 76, 134, 146, 156, 160, 170]	9
User feedback	[4, 36, 44, 60, 117, 118, 120, 127, 152]	9
Enormous design space	[17, 42, 71, 77, 82, 85, 100, 104, 114]	9
Uncertain decisions	[45, 62, 81, 82, 93, 96, 97, 157]	8
Requirement evolvement	[14, 33, 49, 113, 124, 144, 163]	7
Abstraction	[43, 60, 74, 156, 159, 173]	6
Inaccurate and assumptions based–implementation	[19, 43, 133, 151, 152, 158]	6
Limited resources, unknown risks,–scope adjustments	[10, 43, 118, 147, 160]	5
Non-defect corrections	[84, 98, 127, 130, 140]	5
Violation of standards	[57, 107, 112, 124, 126]	5
Environmental issues	[8, 49, 52, 160]	4
Poor Understanding	[62, 91, 175]	3
Different methods followed	[16, 151]	2
Poor management	[16]	1

changes, e.g., format, alignment, comment, are a source of iteration—code changes due to removal, refactoring, or extension of code cause iteration during software development. Enhancement to add a new feature, accommodate changes, and enhance existing features is also a reason to iterate.

F. POOR MANAGEMENT

Insufficient planning, continuous changes in requirements, configuration problems, defects, and critical stakeholders being overlooked are consequences of poor management, leading to performing work again.

G. LIMITED RESOURCES, UNKNOWN RISKS, SCOPE ADJUSTMENTS

Scope adjustments to meet schedule, multiple people involved in a decision and limited budget, different priority levels, unknown and risky conditions, unfordable and infeasible requirements, unforeseen events, and limited resources occur during software development and are the reason for iteration.

H. BUGS

Iteration in the software development lifecycle can be due to bugs that occur due to miscommunication, programming errors, and continuous changes.

I. CODE AND DESIGN SMELLS

Whenever there are bad smells in code or design, it becomes rigid and challenging to modify. Iteration occurs to remove bad smells and enhance the code and design flexibility. Code erosion, duplicated code, lazy—large, complex, unwieldy, and less cohesive– classes, coding style violation, design anomalies, and flaws, evolving code, non-compliance to design principles, extreme metric values, antiquated code, unstructured code, clones, rigidity and immobility in design, weak attributes in coding, and deteriorated code are all the different forms or reasons of bad smells which are the source of iteration during development and maintenance. Time pressure leads to a higher workload which also becomes a reason to introduce bad smells by developers.

J. TECHNICAL DEBT

During development, it is the situation when there is a need to take a shortcut. An advantage of technical debt is that it accelerates development but has to repay it later in the development lifecycle. Technical debt due to lack of trace structure, unimplemented features, outdated documentation, poorly organized logic creates iteration. Deferred technical debt during development creates much rework.

K. USER FEEDBACK

Maintenance requests, feedback discussions, late customer feedback, lack of user involvement, documentation unavailability, user feedback on prototypes, and usability evaluation of user interface are different causes of iteration in software development.

L. NON-DEFECT CORRECTIONS

It includes perfective and preventive changes, which cause refinements.

M. ENVIRONMENTAL ISSUES

Changes in technology, business context, market trends, policies and legislation, commercial strategies, technical renova-

TABLE 9. Stages of development cycle at which iteration occurs.

Stages	Studies	Count
Maintenance	[7, 9, 52, 55, 62, 63, 64, 103, 121, 124, 127, 130, 136, 140, 142, 144, 155, 167, 176] [63, 64, 106, 138, 139, 160, 162, 174]	27
Requirement	[11, 19, 36, 42, 50, 54, 87, 88, 96, 106, 110, 111, 113, 116, 118, 136, 149, 154, 166, 168, 174, 178]	22
Implementation	[9, 34, 51, 53, 55, 62, 63, 65, 65, 80, 82, 89, 103, 106, 112, 119, 130, 136, 138, 162, 164]	21
Design	[10, 17, 18, 19, 42, 45, 47, 70, 71, 77, 80, 81, 88, 108, 113, 132, 162, 172, 178, 179]	20
Throughout SDLC	[33, 49, 68, 70, 72, 95, 101, 105, 109, 110, 125, 150, 158]	13
Testing	[13, 46, 65, 144]	4

TABLE 10. Impacts of iteration.

Factors	Impacts	Studies	count
Quality	↑	[3, 33, 48, 51, 53, 55, 60, 62, 116, 154, 155, 165] [9, 66, 68, 76, 77, 103, 110, 158, 176]	21
	↓	[49, 122, 127]	3
Understanding and clarity	↑	[9, 19, 33, 54, 57, 62, 63, 68, 76, 80, 86, 105, 112, 121, 128, 155, 166]	17
Flexibility	↑	[33, 44, 48, 53, 55, 62, 63, 68, 76, 105, 112, 128, 158, 162, 165]	15
Maintainability	↑	[33, 48, 53, 62, 68, 80, 86, 89, 98, 105, 122, 138, 167]	13
Decision making	↑	[10, 12, 13, 96, 100, 104, 118, 127]	8
Cost	↑	[45, 46, 49, 63, 109, 125, 129, 152]	8
	↓	[19, 60, 61, 62, 66, 68, 103, 107, 110, 158]	10
Time	↑	[16, 49, 99, 109, 129, 141, 160]	7
	↓	[10, 48, 61, 110, 112, 119, 160, 162]	8
Defects	↑	[3, 49, 89, 155, 158, 177]	6
	↓	[11, 70, 76, 82, 107, 116, 127][9, 10, 87, 90, 110, 172, 177]	14
Effort	↑	[49, 109, 140, 152, 155]	5
	↓	[158, 167]	2
Productivity	↑	[44, 61]	2
Testability	↑	[62, 80]	2
Customer satisfaction	↑	[118]	1
Creativity and innovation	↑	[148]	1
Complexity	↓	[9, 60, 68, 76, 135, 139, 142, 166, 168, 176]	10

tion, outdated requirements, and plans are reasons to perform work again during software development projects.

N. INACCURATE ASSUMPTIONS BASED IMPLEMENTATION
Iteration during software development can be partial implementation, incorrect implementation, inaccurate assumptions, ignorance, not exploring all possible solution options, and development order violations.

O. REQUIREMENT EVOLVEMENT
Requirements evolve during development which in turn leads to performing a work again. There are many reasons for requirement evolvement; some of those are changes in the environment in which software is situated, external changes by the company context, stakeholders cannot envision at the start, clients change their minds later, and deeper exploration of technology.

P. ABSTRACTION
Whenever there is a lack of clarity or detailed initial design, it needs to transform from abstract to concrete specification, which requires the iteration of the initial one.

Q. ENORMOUS DESIGN SPACE
When there is the involvement of multiple factors in selecting variants against a set of objectives, then, to avoid local optimization and select the best option, evolve the initial ideas that require iteration.

R. VIOLATION OF STANDARDS
Lack of standards and guidelines, violation of heuristics, and lousy programming techniques are some of the causes of performing work again during a software development project.

S. CORRECTIVE AND NON-CORRECTIVE CHANGES

Change can be corrective due to defects, perfective due to enhancement or expansion, adaptive due to modification, and preventive due to malfunctions. Requirement change due to error, modifications of original requirements, change of operational purpose, or support user request leads to requirement instability, volatility, and creep, which generates rework. Some iteration occurs to implement any changes–corrective, non-corrective, or adaptive. When changes occur in a non-negotiable scope, it requires many adjustments.

T. DIFFERENT DEVELOPMENT METHODS FOLLOWED IN TEAMS

Unaligned agendas of different stakeholders and different methods followed by the teams, e.g., front end team follow agile and back end team follow plan-driven methodologies, lead to the iteration during software development.

U. COMPLEXITY

Wicked (underspecified and open-ended) problems, highly complex code, and volatile environment are reasons to perform a task again, develop a prototype to explore requirements, and refine initial ideas.

V. POOR UNDERSTANDING

Language differences, miscommunication, time zone differences lead to ambiguity in requirements and misunderstanding of design intent. The poor understanding causes much rework during software development.

V. STAGES AT WHICH ITERATIONS OCCUR

Iteration occurs during the whole software development lifecycle in different forms and at each stage has different impacts. Iteration occurs during requirement engineering to get a clear understanding and mutual agreements. Iteration invents an innovative and straightforward, less complex solution in the design phase, and making changes is easier later.

When products have aesthetic appeal, iteration occurs during user interface design for quality enhancement of initial one and perfection. During implementation and maintenance, iteration removes the bad smells and improves code quality. Iteration occurs to remove the defects during the whole software development lifecycle.

Stages at which iteration can occur during software development lifecycle from the extracted data against selected primary studies that cite these stages are summarized in Table 9.

VI. IMPACTS OF ITERATION

In software engineering, iteration has both positive and negative consequences. Impacts of iteration extracted from selected primary studies are summarized in Table 10 and explained in the following paragraphs.

A. ITERATION POSITIVELY IMPACT ON QUALITY

Enhancement of quality by iteration is referenced by 21 studies, whereas three studies write negative about iteration impacts on quality, as given in Table 10. Selected viewpoints of different authors about iteration impacts on quality are: refactoring improves code and software quality [33], [51], negotiation during requirement engineering improves requirements quality [116], refactoring improves quality by removing code smells [3], iteration enhances the various characteristics of the software quality [53], [165], iteration results in improving internal quality of the system while preserving its functionality [103]. In system evolvement, refactoring is performed to improve the code quality, but the analysis shows that some refactoring techniques improve whereas others depreciate the quality [122]. Iteration caused by wild requirements changes may damage the quality of the software [49].

B. ITERATION ENHANCES UNDERSTANDABILITY

The literature highlighted that iteration increases conceptual clarity and understandability. The positive impacts of iteration on understandability and clarity are cited by 17 studies, whereas none of the studies discuss its negative impacts on understanding. Iteration progressively refines the genuinely dubious objectives into precise requirements. Iteration enhances program understanding [128], helps the stakeholders to understand the requirements before they sign off [54], code gets simpler and easier to understandable after iteration [19].

C. ITERATION INCREASES PRODUCTIVITY

According to [44] and [61], iteration during development has a positive impact on productivity and quality. Refactoring leads to higher development productivity and assists developers in programming speed. Furthermore, intricate and intertwined code is harder to oversee and maintain than refactored code. Consequently, when maintenance is complex, development productivity will show a diminishing tendency after some time [167].

D. ITERATION MAKES MAINTENANCE EASIER

Iterations make the software easy to maintain and modify from the product and process perspective, i.e., make it scalable. Iteration enhances extensibility, makes it simpler to include new elements, and enhances maintainability. Fifteen studies cite the positive impact of iteration on maintenance, but none referenced its adverse impacts on maintainability. Iteration progressively refines the genuinely dubious objectives into clear requirements. [7] write that code clones are the one reason that makes maintenance difficult. Refactoring can remove code clones and enhance flexibility, although it may not always be a good option. Iteration makes the software adaptable, and adding new functions becomes less demanding [128].

E. COMPLEXITY REDUCES AS A RESULT OF ITERATION

Thirteen studies cite the positive impact of iteration on complexity, as shown in Table 10. None of the selected studies referenced the negative impacts of iteration on complexity. Studies have highlighted that iteration lessens the multifaceted nature of code, design, and general framework; furthermore, it diminishes the size of the code and reduces the complexity [60], [76], [142], [166], [168].

F. ITERATION HELPS IN MAKING CORRECT DECISIONS

Iteration leads to the right decisions by removing ambiguities and making realistic estimations [10]. None of the studies cited the negative impact of iteration on decision making, whereas 8 studies referenced better decision making as a result of iteration. It results in the selection of the best among multiple options for the defined criteria [104]. When conflicts of interest exist among multiple stakeholders' goals, iteration helps in making trade-offs and building consensus [13].

G. ITERATION LEADS TO INNOVATIVE PRODUCTS

Creativity is thinking of new thoughts, and innovation puts those thoughts into a practical project. It can be new thoughts on the best way to build up a superior product, a better design, or better processes for development. Iterations lead towards innovation, creativity, and secure upper hands through feature separation [148].

H. ITERATION HELPS IN REMOVING DEFECTS

Increment in the defects by iteration is referenced by 6 studies, while 14 studies write that iteration results in removing defects, as can see in Table 10. Iteration removes ambiguities and results in complete and less ambiguous requirements [11]. Iteration fixes bad design practices [76]. Sometimes, while repairing defects, iteration results in the introduction of new defects [3].

I. ITERATION INCREASES COST, WHILE, IT IS COST EFFECTIVE IN LONG RUN

From selected primary studies, 8 studies highlighted that iteration increases the cost of the development, while 10 studies cited that iterations are cost-effective in the long run. Iteration during software development lessens maintenance costs by improving internal structure [103]. Refactoring reduces the cost of making changes to the system [60].

J. DEVELOPMENT TIME INCREASES

Delays in development as a result of iteration is referenced by 7 studies, although 8 write that early iteration minimizes late rework and reduces development time. For instance, [16] write that iteration caused by poor management results in project delays. Code refactoring leads to rapid development [48]. According to [61], iteration results in reducing the time and cost of the project.

K. EFFORT

Iteration impacts negatively on the effort required to develop a software, as shown in Table 10, 5 studies cited that whenever there is an iteration effort required to complete a project increases, but, 2 studies also referenced a decline in the overall effort. Rework increases cost for customers and effort by developers [152]. According to another study, refactoring results in achieving the best outcome with the least effort [167].

VII. TAXONOMY OF ITERATION

The taxonomy classifies different iterative situations based on the source and timings of the development cycle at which it occurs. Iteration has different perspectives during software development because different iterative situations may originate from diverse sources, at different timings, and have impacts. So to handle different perspectives, different ways can be used. Four distinctive iterative perspectives are identified: Rework; Refinement; Exploration; Negotiation. Different perspectives of iteration defined based on source, stage, and impacts are described in the following subsections, and the taxonomy of iteration is presented in Figure 5.

A. REFINEMENT

Refinement is the enhancement of initial specification. It has subtypes in terms of its impacts. One of those is refactoring, which has minimal impact. This type of refinement is done when sufficient time is available or where products have aesthetic appeal or assessment criteria is subjective. It portrays a situation in which essential requirements have been satisfied and experience further iterations to upgrade optional qualities, such as enhancing style or diminishing cost. In general, refinement is the process of removing impurities and improving something by making small changes, e.g., refactoring. Refactoring is the way to change a software system not to modify the code's outer behavior but to enhance its interior structure.

B. REWORK

Rework is one of the iteration's perspectives that seem most regularly in literature. It is reattempting a work in the same manner as before due to changed information or suppositions. Rework requires the reiteration of an assignment since it has initially endeavored with incorrect data and suppositions. An example of rework in software development is a change in requirements, or simply requirements misunderstood. Rework may be produced because a procedure is unpredictable, so it is impossible to recognize the most productive order of work execution beforehand. It may be because of issues that appeared during analysis or requirement changes. If the timing constraints require starting it with incomplete information, it is impossible to eliminate rework because of changed input information later. Rework is adverse because of the increase in time and cost without any increase in software performance and quality.

C. EXPLORATION

Dynamics of exploration involve an iterative process of seeing different alternatives, assessing those solutions, and selecting the optimal one. It incorporates the investigation of new thoughts to tackle an emerging issue and iterative convergence to a solution.

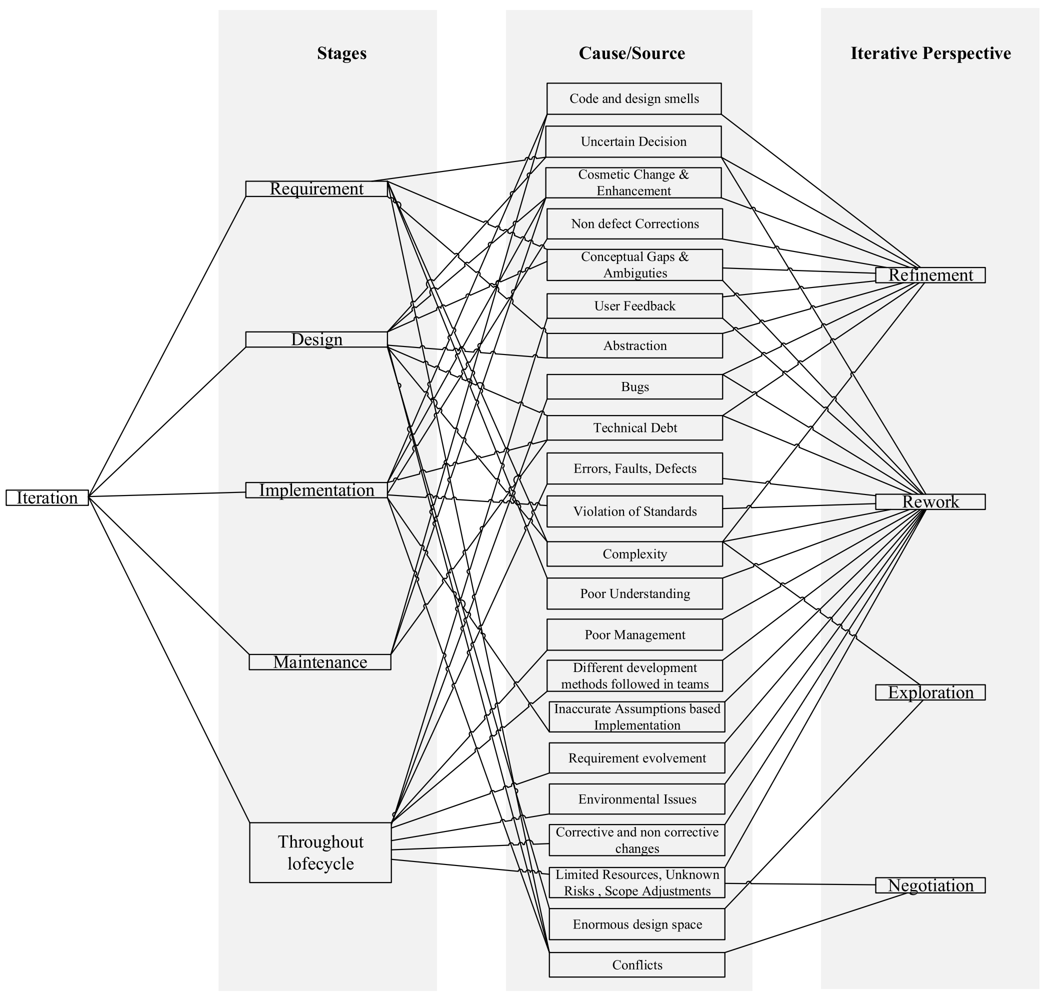


FIGURE 5. **Iteration taxonomy.**

D. NEGOTIATION

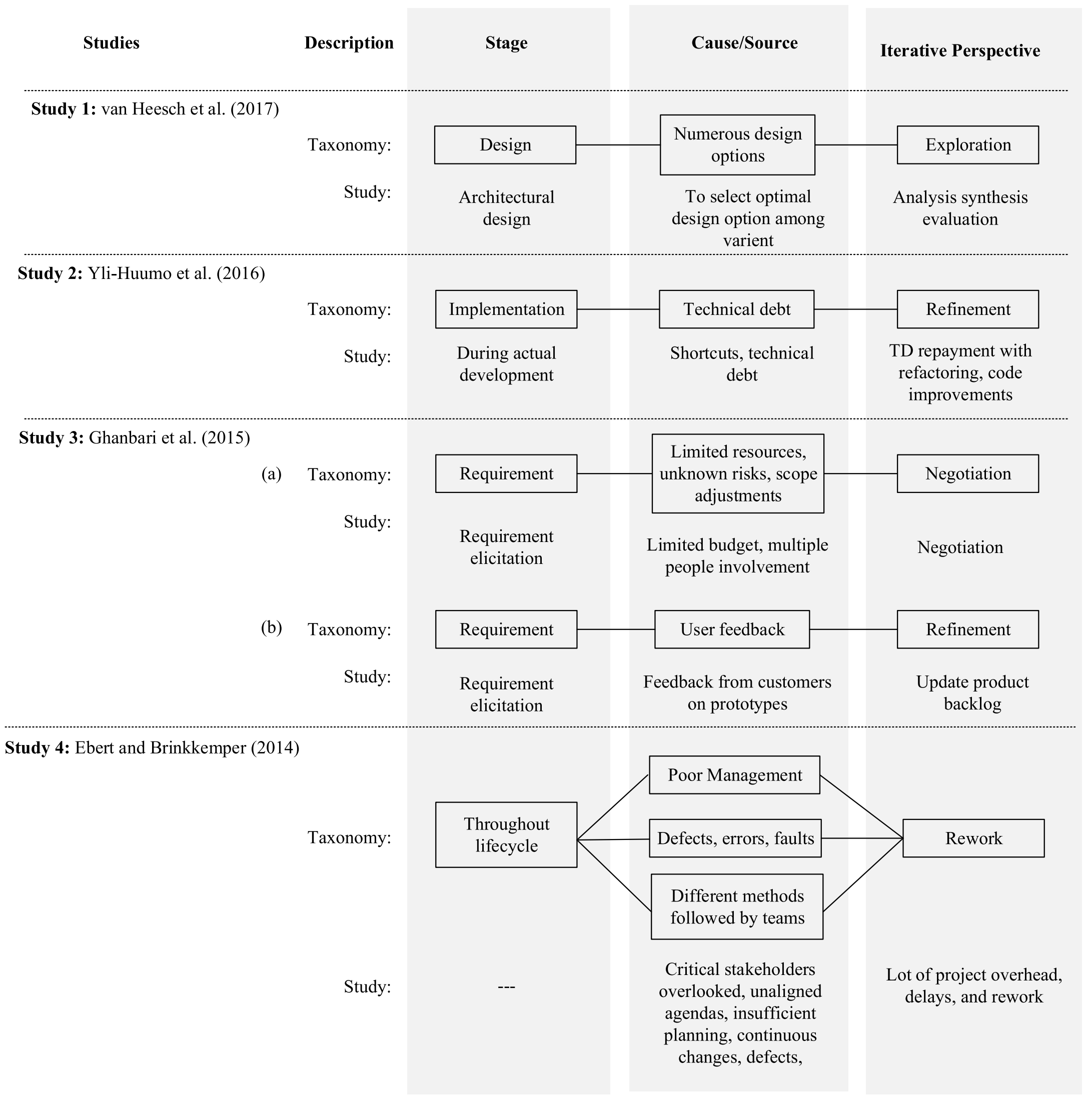
Negotiation is an iteration perspective that describes the circumstance in which the trade-off between various members' objectives and constraints is made by understanding and negotiating their conflicting goals. Negotiation is utilized to consolidate the commitment from various members who have little information about each-other's work, and they regularly have clashing targets. When too many conflicting parameters are involved, negotiation turns out to be excessively troublesome.

The taxonomy shown in Figure 5 presents four different perspectives of iteration that consistently exist in the software engineering literature. The objective is a detailed analysis of iteration, and the taxonomy will explain different terminologies used for iteration with its contextual description to researchers and managers. It will also help in making the comparison between different iterative situations.

To validate the taxonomy, four studies ([16], [17], [34], [118]) were selected to completely map and describe the different iterative perspectives to the taxonomy. First, we describe in detail how the studies can be mapped to taxonomy by selecting four studies then present a summary of all selected primary studies of the SLR. All four studies contain specific iterative situations mapped to the taxonomy. This mapping is summarized in Figure 6 and explained in the following paragraphs. For each paper, the paths from taxonomy, its classification, and its context are explained. In many primary studies, the iterative striations are deduced from the text, and it involves a detailed analysis of each study to extract the related evidence.

In the first study, the authors talk about an iterative situation that occurs during the architectural design for selecting an optimal option among the multiple variants, the analysis, synthesis, and evaluation cycle continue for choosing the best options in the given circumstances [17]. This study maps on exploration (iteration) in the design phase of software development due to the enormous design space.

In the second study, authors code an iterative scenario that arises during actual development time due to taking shortcuts for short-term benefit; repayment occurs in the long run in the different form of refactoring and code improvements [34].


Studies
Description
Stage
Cause/Source
Iterative Perspective
Study 1: van Heesch et al. (2017)
Taxonomy:
Design
Numerous design options
Exploration
Study:
Architectural design
To select optimal design option among varient
Analysis synthesis evaluation
Study 2: Yli-Huumo et al. (2016)
Taxonomy:
Implementation
Technical debt
Refinement
Study:
During actual development
Shortcuts, technical debt
TD repayment with refactoring, code improvements
Study 3: Ghanbari et al. (2015)
(a) Taxonomy:
Requirement
Limited resources, unknown risks, scope adjustments
Negotiation
Study:
Requirement elicitation
Limited budget, multiple people involvement
Negotiation
(b) Taxonomy:
Requirement
User feedback
Refinement
Study:
Requirement elicitation
Feedback from customers on prototypes
Update product backlog
Study 4: Ebert and Brinkkemper (2014)
Taxonomy:
Throughout lifecycle
Poor Management
Defects, errors, faults
Different methods followed by teams
Rework
Study:

Critical stakeholders overlooked, unaligned agendas, insufficient planning, continuous changes, defects,
Lot of project overhead, delays, and rework

FIGURE 6. Illustration of taxonomy.

This study maps on refinement (iteration) in the implementation phase due to the technical debt.

Two different scenarios of iteration are extracted from the textual description of the third study [118]. The first iterative situation is due to the limited budget and multiple people involved in the requirement elicitation that leads to the negotiation. In the second scenario, the product backlog is updated due to the feedback on prototypes from the customers in the requirement eliciting stage. This study maps on both refinement and negotiation.

In the fourth selected study for the validation, authors illustrate the rework perspective of the taxonomy, and multiple causes of it are discussed [16]. Overall, this study describes three paths from the taxonomy shown in Figure 6. The multiple causes extracted from the study are: critical stakeholders overlooked unaligned agendas, insufficient planning, continuous changes, defects that cause the rework throughout the whole software development lifecycle.

Further, we have revised the same 153 studies selected initially from the SLR search process to validate the taxonomy. We then extracted the iterative perspectives that exist in the study according to the taxonomy and summarized the paths from each study in Table 11. We followed the same data extraction, and mapping process explained in Figure 6. The number of studies verifying each iterative perspective against the stage of the development life cycle is shown in 7. Out of 152 studies, 41 studies were mapped on multiple perspectives of iteration, 10 studies on exploration, 59 studies on refinement, 28 studies on rework, and 13 studies mapped on negotiation. Note that it is also possible that a study can be mapped to multiple iterative perspectives if it contains several iteration scenarios that exist in software development

TABLE 11. Taxonomy paths illustration by selected studies.

			Paths				Studies
Iteration	→	Requirement	→	Uncertain Decisions	→	Refinement	[81, 149]
"	→	"	→	"	→	Rework	[14, 81, 163]
"	→	"	→	Conceptual gaps & ambiguities	→	Refinement	[54, 59, 111, 136, 166]
"	→	"	→	"	→	Rework	[12, 115]
"	→	"	→	Abstraction	→	Refinement	[43, 74, 156]
"	→	"	→	Complexity	→	Refinement	[72]
"	→	"	→	"	→	Rework	–
"	→	"	→	"	→	Exploration	[113, 168]
"	→	"	→	Poor understanding	→	Rework	[175]
"	→	"	→	Conflicts	→	Negotiation	[11, 12, 19, 50, 87, 96, 113, 116, 118, 154, 166, 174, 178]
Iteration	→	Design	→	Uncertain Decisions	→	Refinement	–
"	→	"	→	"	→	Rework	[45, 93, 157]
"	→	"	→	Cosmetic changes & enhancement	→	Refinement	[90]
"	→	"	→	Conceptual gaps & ambiguities	→	Refinement	[47, 67, 79, 154]
"	→	"	→	"	→	Rework	–
"	→	"	→	Abstraction	→	Refinement	[60, 66, 74, 132, 159]
"	→	"	→	Technical debt	→	Refinement	[76]
"	→	"	→	"	→	Rework	–
"	→	"	→	Complexity	→	Refinement	[79, 123]
"	→	"	→	"	→	Rework	[162]
"	→	"	→	"	→	Exploration	[10, 70, 168, 172]
"	→	"	→	Enormous design space	→	Exploration	[17, 42, 45, 71, 77, 82, 85, 100, 114]
"	→	"	→	Conflicts	→	Negotiation	[58, 70, 108, 148]
Iteration	→	Implementation	→	Code & design smells	→	Refinement	[48, 53, 55, 68, 76, 86, 103, 112, 134, 138]
"	→	"	→	Cosmetic changes & enhancement	→	Refinement	[65, 78, 106, 145]
"	→	"	→	Non defect corrections	→	Refinement	[84, 98, 127, 130]
"	→	"	→	Inaccurate assumptions based implementation	→	Rework	[19, 133, 151, 158]
"	→	"	→	Conflicts	→	Negotiation	[145]
"	→	"	→	Technical debt	→	Refinement	[34, 134, 146, 156]
"	→	"	→	"	→	Rework	[146]
"	→	"	→	Violation of standards	→	Rework	[124, 126]
Iteration	→	Maintenance	→	Code & design smells	→	Refinement	[7, 33, 55, 57, 62, 63, 103, 121, 124, 135, 138, 155, 167, 176]
"	→	"	→	Cosmetic changes & enhancement	→	Refinement	[78, 122, 144]
"	→	"	→	Technical debt	→	Rework	[52, 160, 170]
"	→	"	→	"	→	Refinement	–
Iteration	→	Throughout lifecycle	→	Limited resources, unknown risks	→	Rework	[97, 160]
"	→	"	→	"	→	Negotiation	[43, 97, 118]
"	→	"	→	Corrective & non-corrective changes	→	Rework	[47, 61, 99, 105, 127, 140, 149, 150, 151, 160]
"	→	"	→	Environmental issues	→	Rework	[8, 49, 160]
"	→	"	→	Requirement evolvement	→	Rework	[14, 36, 49, 124]
"	→	"	→	Different methods followed by teams	→	Rework	[16, 151]
"	→	"	→	Poor management	→	Rework	[16, 133]
"	→	"	→	Errors/faults/defects	→	Rework	[8, 13, 36, 46, 49, 90, 95, 105, 109, 110, 124, 125, 129, 130, 145, 161]
"	→	"	→	Bugs	→	Refinement	[9, 33, 51, 62, 139, 142, 147, 156, 158, 167, 173, 176]
"	→	"	→	"	→	Rework	[126]
"	→	"	→	User feedback	→	Refinement	[4, 18, 36, 44, 118, 127]
"	→	"	→	"	→	Rework	[4, 36, 120, 152]

projects. For example, both rework and refinement are mentioned in a study as given in Table 11 [127].

VIII. VALIDITY THREATS AND EVALUATION

We used internal, external, construct, and conclusion validity threats classification adapted from the work of [180].

A. INTERNAL VALIDITY THREATS

According to [180], internal threats are related to the wrong conclusion drawn about the causal relationship. This type of threat can occur due to personal bias on study understanding. We tried to minimize internal threats by mitigating personal biases. We performed the selection process iteratively by multiple authors to remove personal biasness. Another threat to internal validity can be erroneous data extraction as the number of primary studies is significant, and data extraction involves subjectivity. Data extraction was harrowing because many studies did not explicitly mention sources, stages, and impacts of iteration, and an interpretation of data was required, which involves personal biasness. This threat was minimized by performing data extraction by multiple researchers.

B. EXTERNAL VALIDITY THREATS

The external validity of the literature review depends on the selected primary studies. The selected studies should be valid and representative of the review topic in the external evaluation. We applied inclusion, exclusion, and quality assessment criteria to minimize this type of threats to select good literature. To mitigate external validity threats, we defined the search process iteratively by performing some trials and getting agreement from all authors. The threat can be due to assessing all the studies by a single researcher for inclusion and exclusion decisions. The inclusion/exclusion decisions were discussed with the advisor, and a test-retest approach was applied, as recommended by the [32] to mitigate this threat. To check reliability of the decisions, reevaluations of inclusion and exclusion were performed for randomly selected studies.

C. CONSTRUCT VALIDITY THREATS

Construct-related threats are about generalizing the results [180]. We used search strings with different synonyms and six different databases to search related studies to minimize this type of threat. In synthesis, we extracted the common

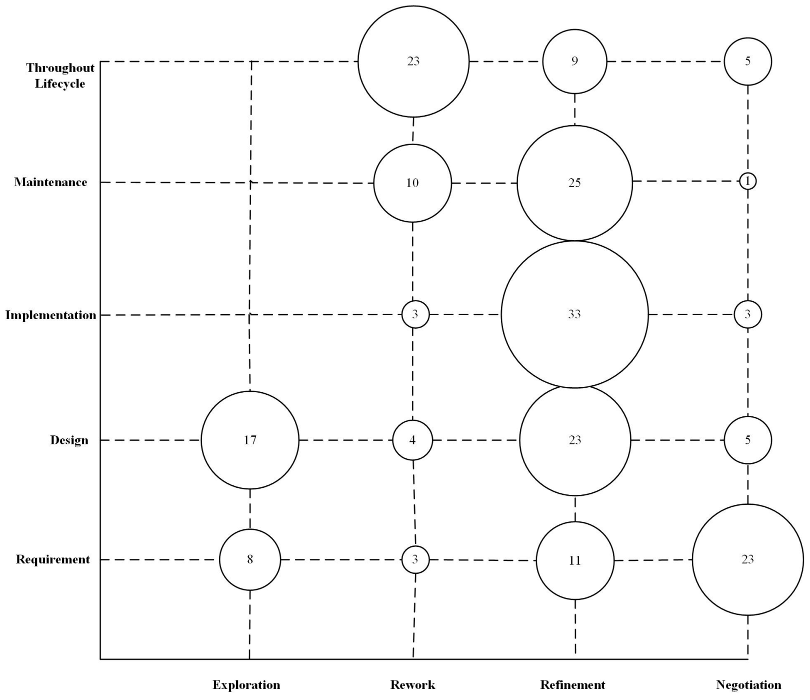


FIGURE 7. Summary of the selected studies.

concepts described using different terms and normalized the terms used. Additionally, we created a lucid taxonomy based on these concepts. In the development of research questions, we have not completely followed the PICOC guidelines proposed by [32].

D. CONCLUSION VALIDITY THREATS

We cannot identify all the primary studies that exist related to research questions [32]. We used multiple synonyms for the keywords to cover maximum studies and minimize this threat while designing search string. To mitigate conclusion validity threats, the whole process was developed carefully and verified.

IX. CONCLUSION

The investigation into the iteration suggests that additional care is necessary for dealing with iteration during software development. In software engineering literature, different authors use different terms to refer to iteration at different places. Additionally, many different sources contribute to iteration at different phases of the development cycle and impact the project in different ways. In this context, the management of iteration becomes a troublesome task. Therefore, to resolve the issues related to iteration, we created the taxonomy of iteration. So, the main objective of this article was to combine the existing knowledge about the different iterative situations and enhance understanding of these situations that consistently exist in software engineering. Different characteristics of each iterative situation were analyzed, such as sources/causes that lead to that situation, stages of the development process at which it can occur, and its impacts on the development project, positive and negative.

Our SLR is based on 153 primary studies, selected out of 2072, through four stages. The review encompasses the whole software development lifecycle, i.e., the review is not restricted to a particular phase or domain. The broader scope of the review gives us deeper insights into iteration during the whole software development lifecycle. We have verified the created taxonomy using selected primary studies.

The significant findings from this review and suggestions for further research are given in the following subsection.

A. PRINCIPLE FINDINGS

In answer to RQ1.1, we have found from the literature that there are lots of multiple causes/ sources that yield iteration in software engineering projects. It is also noted that different terms are being used in literature for the same cause/ source concept. Due to many extracted causes, we synthesized those into twenty-two leading causes. The most cited causes are defects, code smells, and conflicts, whereas the least referenced causes are poor management and different development methods followed by teams.

When responding to RQ1.2, we observed that the iteration could occur at different phases in different forms during the whole software development lifecycle. The 22 studies cited iteration at requirement stage, 20 studies at design, 21 papers at implementation, 4 studies mention iteration at the testing phase, 27 studies talk about iteration at maintenance, while 13 articles mention the iteration during the whole software development lifecycle.

TABLE 12. Selected primary studies with their quality score.

Studies	Citations	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30	Quality
[17]	0	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	1	100
[18]	2	1	0.5	1	1				1				1				1	0			1				1	1	1	0.5	1	1	0.5	84.38
[36]	5	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	0.5	96.88
[34]	7	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	1	100
[144]	10	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	1	100
[51]	4	1	1	1	1	1	1	1	1	1			1		0.5	1	1	0	1	1	1	1	0.5	1	1		1				1	91.30
[145]	15	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	0.5	96.88
[33]	3	1	0.5	1	1				1				1				0.5	1			1				1	1	0.5	0.5	1	1	1	87.5
[116]	0	1	1	0.5	0.5				0.5				0.5		0.5	0.5	0.5	0.5	0.5	0.5	0.5	1	0	0	1		0.5				0	52.63
[3]	6	1	1	1	1				1				1		1	1	1	1	1	1	1	1	1	1	1		1				1	100
[117]	1	1	1	1	1				1				1		0.5	1	1	1	1	1	1	1	0.5	1	1		1				0.5	92.11
[147]	1	1	1	1	1				1				1				1	0			1				1	0.5	1	1	1	1	0.5	87.5
[52]	0	1	1	1	1	1	1	1	1	1			1		1	1	0.5	0.5	1	0.5	1	1	0	0.5	1	1	1	0.5	1	1	1	87.04
[118]	5	1	1	1	1				1				1		1	1	1	0.5	1	1	1	1	1	1	1		1				1	97.37
[42]	20	1		1	1	0.5	1	1		0.5			1					1														88.89
[19]	2	1	1	1	1	1	1	1	1	0.5			1				1	0.5			0.5				1	1	1	0.5	1	1	0.5	87.5
[148]	10	1	1	1	1				1				1				1	1			1				1	1	1	0.5	1	1	0.5	93.75
[43]	2	1		1	1	0.5	1	1		0.5			0.5					0														72.22
[146]	21	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	1	100
[44]	1	1		0.5	1	0.5	1	1		1			1					0														77.78
[53]	9	1	1	1	1	1	1	1	1	1			1		0.5	1	1	1	1	1	1	1	0.5	1	1		1				1	95.65
[165]	4	1						1			1	1	1	1																		100
[120]	0	1	1	0.5	1				1				1		1	0.5	1	0.5	1	1	1	1	0.5	1	1		1				1	86.84
[54]	22	1	1	1	1	1	1	1	0.5	1			1		0.5	1	0.5	1	1	1	0.5	1	0.5	0.5	1		1				1	86.96
[121]	7	1	1	1	1				1				1		0.5	1	0.5	0.5	1	1	1	1	1	0.5	0.5		1				0.5	84.21
[122]	2	1	1	0.5	1				1				0.5		0.5	1	0.5	0	0.5	1	0.5	1	0.5	0	1		1				0.5	68.42
[123]	17	1	1	1	1				1				1		1	0.5	1	0.5	1	1	1	1	0.5	0.5	1		1				1	89.47
[45]	4	1		1	0.5	1	1	1		0.5			1					0														77.78
[149]	9	1	1	1	1				1				1				1	1			1				1	1	1	0.5	1	1	0.5	93.75
[16]	21	1	1	1	1				1				1				0.5	1			1				1	1	1	0.5	1	1	0.5	90.63
[150]	69	1	1	1	1				1				1				1	0.5			1				1	1	1	1	1	1	1	96.88
[151]	50	1	1	1	0.5				1				1				1	1			1				1	1	1	1	1	1	0.5	93.75
[124]	35	1	1	1	1				1				1		1	1	1	1	1	1	1	1	0.5	0.5	1		1				1	94.74
[46]	7	1		1	1	1	1	1		1			1					0.5														94.44
[47]	28	1		1	1	0.5	1	1		1			1					0														83.33
[125]	22	1	1	1	1				0.5				1		1	1	1	0	0.5	1	1	1	0.5	0.5	1		1				0	78.95
[126]	26	1	0.5	1	1				0.5				1		0.5	1	1	0.5	0.5	1	0.5	1	1	0.5	1		1				0.5	78.95
[178]	84	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	1	100
[48]	9	1		1	1	1	1	1		1			1					0.5														94.44
[55]	16	1	0.5	1	1	1	1	1	0.5	0.5			1		0.5	1	0.5	0	0.5	1	0.5	1	0	0.5	1		1				1	73.91
[56]	2	1	1	1	1	1	1	1	0.5	1			1				0.5	0			1				1	1	1	1	1	1	0.5	87.5
[57]	47	1	1	1	1	1	1	1	1	1			1		1	0.5	1	0.5	1	1	1	1	0.5	0.5	1		1				0.5	89.13
[127]	16	1	1	1	1				1				1		1	1	0.5	0	1	1	0.5	1	0.5	1	0.5		1				0.5	81.58
[58]	31	1	1	0.5	1	1	1	1	0.5	1			1		1	1	1	0.5	0.5	1	0.5	0.5	0.5	0.5	1		1				1	82.60
[50]	3	0.5		0.5	1	0.5	1	0		0.5			0.5					0														50
[49]	7	1		1	1	1	1	0.5		1			1					0														83.33
[152]	121	1	1	1	1				1				1				1	1			1				1	1	1	1	1	1	1	100
[59]	154	1	1	1	1	1	1	1	1	1			1		1	1	1	0.5	1	1	1	1	0.5	0.5	1		1				1	93.48
[60]	45	1	1	1	0.5	1	1	1	0.5	1			1		1	1	0.5	0	0.5	1	1	1	0.5	0.5	1		1				1	82.61
[61]	15	1	1	1	1	1	1	1	1	1			1		1	0.5	0.5	0	0.5	0.5	1	1	0	0.5	1		1				0	76.09
[130]	41	1	1	1	1				1				1		1	1	1	1	0.5	1	1	1	0.5	1	1		1				0.5	92.11
[166]	74	1						0.5			1	0.5	1	1																		83.33
[131]	98	1	1	1	1				1				1		1	1	1	1	1	1	1	1	0.5	0.5	1	1	1	1	1	1	1	95.65
[132]	36	1	1	1	1				1				1		1	1	1	1	1	1	1	1	1	0.5	0.5		0.5				0.5	89.47
[62]	26	1	1	1	1	1	1	0.5	0.5	1			1		0.5	0.5	1	0.5	0.5	1	1	1	0.5	0.5	1		0.5				0	76.09
[69]	2	1		0.5	1	0.5	1	1	0.5		1							0														72.22
[153]	41	1	1	1	0.5				1				0.5				0.5	1			0.5				1	1	1	0.5	0.5	1	1	81.25
[154]	39	1	1	1	1				1				1				1	0			1				1	1	1	0.5	1	0.5	1	87.5
[133]	76	1	1	1	1				1				1		1	1	1	0	1	0.5	0.5	1	0.5	0.5	1		1				0.5	81.58
[143]	54	1	1	1	0.5				1				1		1	1	1	0.5	0.5	1	1	1	0.5	0.5	1		1				1	86.84
[109]	8	1		0.5	1	1	0.5	0.5		1			1					0.5														77.78
[142]	137	1	1	1	1				1				1		1	1	1	0.5	0.5	1	1	1	0.5	1	1		1				0.5	89.47
[108]	40	1	1	0.5	1	1	1	1	1	0.5			1				1	0.5			1				1	0.5	1	0.5	1	0.5	0.5	82.5
[70]	39	1		1	1	1	1	1	1		1							0.5														94.44?
[107]	0	1		1	0.5	0.5	1	0.5		1			1					0														72.22
[155]	1	1	0.5	0.5	1				0.5				1				1	0			0.5				1	0.5	1	0.5	1	1	0	68.75
[71]	7	1		1	1	1	1	0.5		1			1					0														83.33
[156]	5	1	1	1	0.5				1				0.5				1	1			1				1	1	1	0.5	0.5	0.5	0	78.13
[157]	3	1	1	1	0.5				0.5				1				1	0.5			1				1	0.5	1	1	0.5	0.5	0	75
[167]	0	1						1			1	0.5	0.5	1																		83.33
[72]	1	1		1	0.5	0.5	1	0.5		1			1					0														72.22
[73]	2	1		1	0.5	1	0.5	1		1			1					0														77.78
[74]	0	1		1	0.5	0.5	1	0.5		1			1					0														72.22
[75]	2	1		1	1	1	0.5	1		0.5			1					0														77.78
[76]	1	1		1	1	0.5	1	1		0.5			1					0														77.78
[168]	10	1						1			1	1	1	1																		100
[134]	61	1	1	1	1				1				1		1	1	1	0.5	1	1	1	1	0.5	0.5	1		1				1	92.11
[135]	4	1	0.5	1	1				1				1		1	1	1	1	0.5	1	1	1	0.5	1	1		1				0	86.84
[136]	0	1	0.5	1	1				0				0.5		0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0	0	0.5		0.5				0.5	50
[77]	0	1		0.5	1	0.5	1	0.5		1			0.5					0														66.67
[78]	9	1		1	1	1	1	1		1			1					0														88.89
[106]	0	1		0.5	1	1	1	0.5		0			1					0														66.68
[79]	5	1		1	0.5	0.5	1	0.5		0			1					0														61.11
[129]	1	1	0.5	1	0.5				0.5				1		1	1	0.5	0	0	0.5	0.5	1	0.5	0	1		0.5				0	57.89
[80]	1	1		1	1	1	1	1		1			1					0.5														94.44
[119]	0	1	0.5	1	0				1				1		0.5	1	0.5	0.5	0.5	0.5	0.5	1	0.5	0.5	1		1				0.5	68.42
[81]	0	1		0.5	0.5	1	1	0.5		1			1					0														72.22
[169]	2	1						1			1	0.5	1	0																		75
[158]	2	1	1	0.5	0.5				0.5				1				1	1			0.5				0.5	0.5	1	0.5	0.5	0.5	0.5	68.75
[82]	27	1	0.5	1	1	1	1	1	1	0			1		0.5	1	1	0	0.5	1	0.5	1	1	1	1		1				0.5	80.43
[83]	0	1		1	0.5	1	1	0.5		0.5			1					0														72.22
[159]	10	1	1	1	1				1				1				1	0.5			1				1	1	1	1	1	1	0	90.63
[170]	2	1						0.5			0.5	1	1	0																		66.67
[160]	2	1	0.5	1	0.5				0.5				1				0.5	0			1				1	1	0.5	0.5	0.5	0.5	0	62.5
[171]	3	1						1			1	1	1	0.5																		91.67
[137]	9	1	1	1	1				1				1		1	0.5	0.5	1	1	0.5	1	1	0.5	0	1		1				0.5	81.58
[84]	27	1		1	0.5	1	1	1		1			1					0														83.33
[85]	39	1		1	1	1	1	1		1			1					0.5														94.44
[161]	1	1	0.5	0.5	0.5				0.5				0.5				0.5	1			0.5				1	0.5	0.5	0	0.5	0.5	0	53.13
[86]	29	1	1	1	1	1	1	1	1	0.5			1		1	1	1	1	1	1	1	1	0.5	1	1		1				1	95.65
[87]	0	1		0.5	1	1	1	0.5		1			0.5					0														72.22
[88]	9	1		0.5	1	1	1	1		1			1					0														83.33
[89]	60	1		1	1	1	1	1		1			1					0.5														94.44
[90]	19	1	?	1	1	1	1	1	0.5	0.5			1		0.5	1	0.5	0	0.5	0.5	0.5	1	0.5	1	1		1				0.5	71.74
[91]	15	1	0.5	1	0.5	0.5	1	0.5	0.5	0			1		0.5	0.5	0.5	0	0.5	1	0.5	1	0.5	0.5	1		1				0.5	63.04
[92]	6	1		1	1	0.5	1	1		1			0.5					0														77.78
[105]	0	1		0.5	0.5	1	1	0.5		1			0.5					0														66.68
[172]	1	1						0.5			1	0.5	0.5	0.5																		66.67
[93]	0	1		0.5	0.5	0.5	1	0.5		1			0.5					0														61.11
[94]	3	1		1	0	0.5	1	0.5		1			1					0														66.67
[95]	1	1	0.5	0.5	0.5				0.5				0.5				0.5	0			1				0.5	0.5	1	0	0.5	0.5	0.5	53.13
[110]	0	1	0.5	1	0.5	1	1	0.5	1	0			0.5		1	1	0.5	0.5	0.5	0.5	1	1	0.5	0	1		0.5				0.5	67.39
[173]	8	1						1			1	1	1	0																		83.33
[111]	21	1	1	1	0.5	0.5	1	1	1	1			1		1	0.5	1	0.5	0.5	1	1	1	0.5	0	1		1				0.5	80.43
[112]	5	1		1	0.5	1	1	1		1								0														72.22
[113]	2	1		1	0.5	1	1	1		0.5								0														66.67
[162]	17	1	1	1	0.5				1				1				1	0.5			0.5				1	1	0.5	0.5	1	0.5	0	75

TABLE 12. ***(Continued.)*** **Selected primary studies with their quality score.**

Studies	Citations	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30	Quality
[174]	5	0.5						1			1	1	1	0																		75
[138]	63	1	0.5	1	1				1				1		0.5	1	1	0.5	1	0.5	0.5	1	0.5	1	1		1				1	84.21
[139]	3	1	0.5	0.5	1				1				1		0.5	0.5	0.5	0.5	1	0.5	1	1	0	1	1		0.5				0.5	71.05
[63]	6	1		1	0	1	1	1		0.5			1					0														72.22
[64]	20	1		1	0	1	1	1		1			1					0.5														83.33
[65]	43	1		1	1	1	1	1		1			1					0														88.89
[66]	2	1		1	1	0.5	0.5	0.5		1			0.5					0														66.67
[140]	7	1	0.5	0.5	1				0.5				1		0.5	1	0.5	0	0.5	0.5	0.5	1	0.5	0	1		1				0	60.53
[163]	33	1	1	1	0.5				1				1				1	0.5			1				1	1	1	0.5	1	1	0.5	87.5
[67]	4	1		1	1	1	1	1		1			1					0														88.89
[68]	13	1	0.5	0.5	0.5	1	0.5	1	0.5	0.5			1		0	0.5	0.5	0	0.5	0.5	0	0.5	0.5	0	1		0.5				0	50
[141]	27	1	0.5	1	0.5				1				1		1	1	1	1	0.5	1	1	1	0.5	0	1		1				0.5	81.58
[175]	5	1						0.5			1	0.5	1	0.5																		75
[9]	10	1	0.5	0.5	1				1				0.5				0.5	0.5			1				1	0.5	0.5	0	0.5	0.5	0.5	62.5
[114]	30	1	1	1	0.5	1	1	1	1	0			1		1	1	0.5	0	0.5	0.5	0.5	0.5	0.5	0.5	1		1				1	73.91
[176]	2	1						1			1	1	1	1																		100
[164]	19	1	1	1	0.5				0.5				0.5				0.5	1			1				1	1	1	0.5	0.5	1	0	75
[115]	6	1		1	1	0.5	1	1		0			1					0														72.22
[8]	1	1		1	0	0.5	1	0.5		0			1					0														55.56
[96]	27	1		1	1	1	1	1		1			1					0														88.891
[97]	12	1		0.5	0.5	1	1	1		0			1					1														77.78
[98]	7	1		1	1	1	1	1		1			1					0														88.89
[99]	31	1	0.5	1	1	1	1	1	1	0.5			1		0.5	0	0.5	0.5	0.5	0.5	0.5	1	0.5	0	1		1				1	71.74
[100]	12	1		0.5	1	1	1	1		1			1					0														83.33
[14]	1	1		0.5	1	1	1	1		0.5			1					0														77.78
[128]	109	1	1	1	1				1				1		1	1	1	1	1	0.5	1	1	0.5	1	1		1				0.5	92.11
[101]	14	0.5	0.5	1	0.5	1	1	1	0	0.5			1		0.5	0.5	0	0	1	0.5	0	1	0	0	1		1				0.5	56.52
[4]	119	0.5	1	1	1				1				1				1	0			0.5				1	1	1	1	0.5	1	0	78.13
[13]	5	1	1	1	1	1	1	1	1	0.5			1		1	1	0.5	0.5	1	0.5	0.5	0.5	0.5	0	1		1				0	76.09
[102]	26	1	0.5	1	1	1	1	1	0.5	1			1		0.5	0.5	1	0	1	0.5	0.5	0.5	0.5	1	1		1				0.5	76.09
[12]	8	1		0.5	1	1	0.5	1		0.5			1					0														72.22
[103]	49	1		1	1	1	1	1		0.5			1					0														83.33
[104]	48	1		1	1	1	1	1		0			1					1														88.89
[10]	13	1		0.5	1	1	1	1		1			1					0														83.33
[7]	37	1	1	1	1	1	1	1	1	1			1		1	0.5	1	0.5	1	0	0.5	1	0	1	1		1				1	84.78
[11]	18	1		1	0.5	1	0.5	1		0			1					0														66.67

In response to RQ1.3, it is found that iteration has both positive and negative impacts on the development project. The most cited positive consequences of iteration are quality improvement (21 studies), flexible and more accessible maintenance (28 studies), understandability enhancement (17 studies), complexity reduction (10 studies), and better decision making (8 studies). The negative impacts of iteration are increasing effort (5 studies), time (7 studies), and cost (8 studies).

Regarding RQ1.4, we observed that the different iterative situations were rarely defined in the literature. Moreover, different authors use their terms to refer to iteration at different places, which is misleading. Sometimes, an iterative situation is being denoted by different terminology and vice versa. This makes the management of iteration difficult and problematic for the practitioners, and it also makes problems for researchers trying to synthesize results from different research studies. A detailed analysis of different iterative situations does not exist to clarify differences among different iterative situations. On the way towards building the iteration taxonomy, we organized an SLR. Our investigation found that most synonyms are being used for iteration and its sources. Without defining clear and coherent terminology, researchers will continue choosing different terms for iteration. Additionally, it makes synthesizing process for reviews more complex.

The terms used in our proposed taxonomy do not entirely demonstrate the current use of the terms. However, we built the taxonomy of the different iterative perspectives based on the development cycle's stage and the iteration's source. Understanding different iterative situations and their after-effects play an essential role in the success of a software project. It increases the visibility into processes. Enhancing visibility into processes during development supports better management of the project.

B. AUDIENCE

Though the primary purpose of the taxonomy is the detailed analysis of iteration in software engineering to facilitate the understandability and enhance visibility into development processes. The target audience for our research contribution is both practitioners and researchers. We see inconsistent and different terms to refere iteration are a hurdle for understanding and managing iteration during software development processes. The taxonomy proposed in this article enhances the understanding of different perspectives on unplanned iteration and serves as a roadmap for practitioners and researchers in understanding different iterative situations.

C. IMPLICATION FOR FUTURE

1) RESEARCH

The contribution of this study is the taxonomy of iteration in software engineering. It also demonstrates how taxonomy can be based on literature and map present studies. This taxonomy of iteration is a step toward further research on iteration in software engineering. If the terms are not clear and consistent, the search process, developing search strings, and directing literature reviews are challenging. The taxonomy can also be used to synthesize existing knowledge, find the gaps, and further analyze. The method used for developing the taxonomy can also be used in other research areas. It is also possible to investigate the relationship among different iteration perspectives. It is also likely to conduct an expert survey to validate the terminologies, causes, impacts, and taxonomy.

2) PRACTICE

As a number of diverse scenarios for iterative situations are possible, it is reasonable to accept that an iterative situation that emerges from the requirement phase is different from that which emerges from another phase. It is also sound to understand that different causes at different stages lead to different iterative situations, and each of these scenarios cannot be managed in the same way. Thus, the context plays an essential role in managing iterative situations. The mapping of the studies to the taxonomy suggests that it is often challenging to understand the context of the iteration described in the software engineering literature. For better management support, an implication of the taxonomy of iteration in software engineering is that it is possible to extract the paths and incorporate them in planning and management tools. Contextual information about the different iterative scenarios can be extracted from taxonomy which can further be used in simulating the particular situation for better decision making.

APPENDIX
QA RESULTS

Quality assessment results of the selected studies are shown in the Table 12.

REFERENCES

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. New York, NY, USA: Palgrave Macmillan, 2005.

[2] M. Dowson, "Iteration in the software process; review of the 3rd international software process workshop," in *Proc. 9th Int. Conf. Softw. Eng.*, 1987, pp. 36–41.

[3] M. W. Mkaouer, M. Kessentini, S. Bechikh, M. Ó. Cinneéide, and K. Deb, "On the use of many quality attributes for software refactoring: A many-objective search-based software engineering approach," *Empirical Softw. Eng.*, vol. 21, no. 6, pp. 2503–2545, Dec. 2016.

[4] J. Ferreira, J. Noble, and R. Biddle, "Agile development iterations and UI design," in *Proc. Agile Conf. (AGILE)*, Aug. 2007, pp. 50–58.

[5] M. Hamid, F. Zeshan, A. Ahmad, F. Ahmad, M. A. Hamza, Z. A. Khan, S. Munawar, and H. Aljuaid, "An intelligent recommender and decision support system (IRDSS) for effective management of software projects," *IEEE Access*, vol. 8, pp. 140752–140766, 2020.

[6] M. Mumtaz, N. Ahmad, M. U. Ashraf, A. Alshaflut, A. Alourani, and H. J. Anjum, "Modeling iteration's perspectives in software engineering," *IEEE Access*, vol. 10, pp. 19333–19347, 2022.

[7] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue, "Simultaneous modification support based on code clone analysis," in *Proc. 14th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2007, pp. 262–269.

[8] J.-F. Tang, "An adaptive model of health diagnosis for agile software development," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 2, Jul. 2008, pp. 655–659.

[9] K. O. Elish and M. Alshayeb, "Investigating the effect of refactoring on software testing effort," in *Proc. 16th Asia–Pacific Softw. Eng. Conf.*, Dec. 2009, pp. 29–34.

[10] M. Riebisch and S. Wohlfarth, "Introducing impact analysis for architectural decisions," in *Proc. 14th Annu. IEEE Int. Conf. Workshops Eng. Comput.-Based Syst. (ECBS)*, Mar. 2007, pp. 381–392.

[11] V. Laporti, M. R. S. Borges, and V. P. Braganholo, "A collaborative approach to requirements elicitation," in *Proc. 11th Int. Conf. Comput. Supported Cooperat. Work Design*, Apr. 2007, pp. 734–739.

[12] H. Kitapci and B. Boehm, "Formalizing informal stakeholder decisions—A hybrid method approach," in *Proc. 40th Annu. Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2007, p. 283.

[13] A. De Lucia, F. Fasano, G. Tortora, and G. Scanniello, "Assessing the effectiveness of a distributed method for code inspection: A controlled experiment," in *Proc. Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2007, pp. 252–261.

[14] K. Mu, Z. Jin, and D. Zowghi, "A measurement-driven process model for managing inconsistent software requirements," in *Proc. 15th Asia–Pacific Softw. Eng. Conf.*, 2008, pp. 291–298.

[15] F. M. Hamid, A. Ahmad, and E. Aimeur, "Factors contributing in failures of software projects," *Int. J. Comput. Sci. Netw. Secur.*, vol. 19, no. 5, pp. 62–77, 2019.

[16] C. Ebert and S. Brinkkemper, "Software product management—An industry evaluation," *J. Syst. Softw.*, vol. 95, pp. 10–18, Sep. 2014.

[17] U. van Heesch, A. Jansen, H. Pei-Breivold, P. Avgeriou, and C. Manteuffel, "Platform design space exploration using architecture decision viewpoints—A longitudinal study," *J. Syst. Softw.*, vol. 124, pp. 56–81, Feb. 2017.

[18] M. Larusdottir, J. Gulliksen, and Å. Cajander, "A license to kill–improving UCSD in agile development," *J. Syst. Softw.*, vol. 123, pp. 214–222, Jan. 2017.

[19] J. Pernstål, T. Gorschek, R. Feldt, and D. Florén, "Requirements communication and balancing in large-scale software-intensive product development," *Inf. Softw. Technol.*, vol. 67, pp. 44–64, Nov. 2015.

[20] N. Bhuiyan, D. Gerwin, and V. Thomson, "Simulation of the new product development process for performance improvement," *Manage. Sci.*, vol. 50, no. 12, pp. 1690–1703, Dec. 2004.

[21] D. Unger and S. Eppinger, "Improving product development process design: A method for managing information flows, risks, and iterations," *J. Eng. Des.*, vol. 22, no. 10, pp. 689–699, Oct. 2011.

[22] T. Taylor and D. N. Ford, "Tipping point failure and robustness in single development projects," *Syst. Dyn. Rev.*, vol. 22, no. 1, pp. 51–71, Mar. 2006.

[23] H.-B. Jun and H.-W. Suh, "A modeling framework for product development process considering its characteristics," *IEEE Trans. Eng. Manag.*, vol. 55, no. 1, pp. 103–119, Feb. 2008.

[24] M. Haller, W. Lu, L. Stehn, and G. Jansson, "An indicator for superfluous iteration in offsite building design processes," *Architectural Eng. Design Manage.*, vol. 11, no. 5, pp. 360–375, Sep. 2015.

[25] D. C. Wynn, C. M. Eckert, and P. J. Clarkson, "Modelling iteration in engineering design," School Technol., Cambridge Univ., Cambridge, U.K., Tech. Rep., 2007.

[26] J. Clarkson and C. Eckert, *Design Process Improvement: A Review of Current Practice*. Cambridge, U.K.: Springer, 2010.

[27] S.-H. Cho and S. D. Eppinger, "A simulation-based process model for managing complex design projects," *IEEE Trans. Eng. Manage.*, vol. 52, no. 3, pp. 316–328, Aug. 2005.

[28] R. Costa, "Productive iteration in student engineering design projects," Ph.D. dissertation, College Eng., Montana State Univ.-Bozeman, Bozeman, MT, USA, 2004.

[29] D. C. Wynn and C. M. Eckert, "Perspectives on iteration in design and development," *Res. Eng. Design*, vol. 28, no. 2, pp. 153–184, Apr. 2017.

[30] M. J. Safoutin, "A methodology for empirical measurement of iteration in engineering design processes," Tech. Rep., Univ. Washington, Seattle, WA, USA, 2003.

[31] R. Costa and D. K. Sobek, "Iteration in engineering design: Inherent and unavoidable or product of choices made?" in *Proc. 15th Int. Conf. Design Theory Methodol.*, Jan. 2003, pp. 669–674.

[32] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Univ. Washington, Seattle, WA, USA, Tech. Rep., 2007.

[33] J. Chen, J. Xiao, Q. Wang, L. J. Osterweil, and M. Li, "Perspectives on refactoring planning and practice: An empirical study," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1397–1436, Jun. 2016.

[34] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt?—An empirical study," *J. Syst. Softw.*, vol. 120, pp. 195–218, Oct. 2016.

[35] H. van Vliet and A. Tang, "Decision making in software architecture," *J. Syst. Softw.*, vol. 117, pp. 638–644, Jul. 2016.

[36] V.-P. Eloranta, K. Koskimies, and T. Mikkonen, "Exploring ScrumBut—An empirical study of scrum anti-patterns," *Inf. Softw. Technol.*, vol. 74, pp. 194–203, Jun. 2016.

[37] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requir. Eng.*, vol. 11, no. 1, pp. 102–107, Mar. 2006.

[38] J. Vilela, J. Castro, L. E. G. Martins, and T. Gorschek, "Integration between requirements engineering and safety analysis: A systematic literature review," *J. Syst. Softw.*, vol. 125, pp. 68–92, Mar. 2017.

[39] S. Tiwari and A. Gupta, "A systematic literature review of use case specifications research," *Inf. Softw. Technol.*, vol. 67, pp. 128–158, Nov. 2015.

[40] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Inf. Softw. Technol.*, vol. 55, pp. 2049–2075, Dec. 2013.

[41] D. Dermeval, J. Vilela, I. B. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva, "Applications of ontologies in requirements engineering: A systematic review of the literature," *Requirements Eng.*, vol. 21, no. 4, pp. 405–437, 2016.

[42] P. Ralph, "The sensemaking-coevolution-implementation theory of software design," *Sci. Comput. Program.*, vol. 101, pp. 21–41, Apr. 2015.

[43] C. Erbas and B. C. Erbas, "Modules and transactions: Building blocks for a theory of software engineering," *Sci. Comput. Program.*, vol. 101, pp. 6–20, Apr. 2015.

[44] T. Zäschke, S. Leone, T. Gmünder, and M. C. Norrie, "Improving conceptual data models through iterative development," *Data Knowl. Eng.*, vol. 98, pp. 54–73, Jul. 2015.

[45] L. Lagadec, C. Teodorov, J.-C. Le Lann, D. Picard, and E. Fabiani, "Model-driven toolset for embedded reconfigurable cores: Flexible prototyping and software-like debugging," *Sci. Comput. Program.*, vol. 96, pp. 156–174, Dec. 2014.

[46] B. Nikolik, "Software quality assurance economics," *Inf. Softw. Technol.*, vol. 54, no. 11, pp. 1229–1238, Nov. 2012.

[47] R. Weinreich and G. Buchgeher, "Towards supporting the software architecture life cycle," *J. Syst. Softw.*, vol. 85, no. 3, pp. 546–561, Mar. 2012.

[48] P.-H. Chu, N.-L. Hsueh, H.-H. Chen, and C.-H. Liu, "A test case refactoring approach for pattern-based software development," *Softw. Quality J.*, vol. 20, no. 1, pp. 43–75, Mar. 2012.

[49] K.-D. Mu, W. Liu, Z. Jin, J. Hong, and D. Bell, "Managing software requirements changes based on negotiation-style revision," *J. Comput. Sci. Technol.*, vol. 26, no. 5, p. 890, 2011.

[50] S. Ahmad and N. A. Muda, "An empirical framework design to examine the improvement in software requirements through negotiation," *Int. J. New Comput. Architectures Appl.*, vol. 1, no. 3, pp. 599–614, 2011.

[51] E. Erturk and E. A. Sezer, "Iterative software fault prediction with a hybrid approach," *Appl. Soft Comput.*, vol. 49, pp. 1020–1033, Dec. 2016.

[52] C. A. Siebra, R. G. Oliveira, C. B. Seaman, F. Q. B. Silva, and A. L. M. Santos, "Theoretical conceptualization of TD: A practical perspective," *J. Syst. Softw.*, vol. 120, pp. 219–237, Oct. 2016.

[53] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and K. Deb, "Multi-criteria code refactoring using search-based software engineering: An industrial case study," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, p. 23, 2016.

[54] V. B. R. V. Sagar and S. Abirami, "Conceptual modeling of natural language functional requirements," *J. Syst. Softw.*, vol. 88, pp. 25–41, Feb. 2014.

[55] A. Christopoulou, E. A. Giakoumakis, V. E. Zafeiris, and V. Soukara, "Automated refactoring to the strategy design pattern," *Inf. Softw. Technol.*, vol. 54, no. 11, pp. 1202–1214, Nov. 2012.

[56] L. Pareto, A. B. Sandberg, P. Eriksson, and S. Ehnebom, "Collaborative prioritization of architectural concerns," *J. Syst. Softw.*, vol. 85, no. 9, pp. 1971–1994, Sep. 2012.

[57] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Identification and application of extract class refactorings in object-oriented systems," *J. Syst. Softw.*, vol. 85, no. 10, pp. 2241–2260, 2012.

[58] X. Peng, B. Chen, Y. Yu, and W. Zhao, "Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2707–2719, Dec. 2012.

[59] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, "The agile requirements refinery: Applying SCRUM principles to software product management," *Inf. Softw. Technol.*, vol. 53, no. 1, pp. 58–70, Jan. 2011.

[60] L. Cao, B. Ramesh, and T. Abdel-Hamid, "Modeling dynamics in agile software development," *ACM Trans. Manage. Inf. Syst.*, vol. 1, no. 1, p. 5, 2010.

[61] M.-I. Sanchez-Segura, F. Medina-Dominguez, A. de Amescua, and A. Mora-Soto, "Improving the efficiency of use of software engineering practices using product patterns," *Inf. Sci.*, vol. 180, no. 14, pp. 2721–2742, Jul. 2010.

[62] C.-T. Chen, Y. C. Cheng, C.-Y. Hsieh, and I.-L. Wu, "Exception handling refactorings: Directed by goals and driven by bug fixing," *J. Syst. Softw.*, vol. 82, no. 2, pp. 333–345, Feb. 2009.

[63] T. Shimomura, K. Ikeda, and M. Takahashi, "An approach to GA-driven automatic refactoring based on design patterns," in *Proc. 5th Int. Conf. Softw. Eng. Adv.*, Aug. 2010, pp. 213–218.

[64] M. Mirzaaghaei, F. Pastore, and M. Pezze, "Automatically repairing test cases for evolving method declarations," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2010, pp. 1–5.

[65] D. Qi, A. Roychoudhury, and Z. Liang, "Test generation to expose changes in evolving programs," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2010, pp. 397–406.

[66] B. Gao, X. Ban, Q. Lv, and X. Li, "A component-based method for software architecture refinement," in *Proc. Int. Conf. Intell. Control Inf. Process.*, Aug. 2010, pp. 574–578.

[67] T. Heyman, R. Scandariato, and W. Joosen, "Security in context: Analysis and refinement of software architectures," in *Proc. IEEE 34th Annu. Comput. Softw. Appl. Conf.*, Jul. 2010, pp. 161–170.

[68] K. Usha, N. Poonguzhali, and E. Kavitha, "A quantitative approach for evaluating the effectiveness of refactoring in software development process," in *Proc. Int. Conf. Methods Models Comput. Sci. (ICMCS)*, Dec. 2009, pp. 1–7.

[69] S. C. Lu and N. Jing, "A socio-technical negotiation approach for collaborative design in software engineering," *Int. J. Collaborative Eng.*, vol. 1, nos. 1–2, pp. 185–209, 2009.

[70] K. Mohan and B. Ramesh, "Traceability-based knowledge integration in group decision and negotiation activities," *Decis. Support Syst.*, vol. 43, no. 3, pp. 968–989, Apr. 2007.

[71] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, A. Pierantonio, and M. Sjodin, "Handling uncertainty in automatically generated implementation models in the automotive domain," in *Proc. 42th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2016, pp. 173–180.

[72] A. Hudic, M. Flittner, T. Lorunser, P. M. Radl, and R. Bless, "Towards a unified secure cloud service development and deployment life-cycle," in *Proc. 11th Int. Conf. Availability, Rel. Secur. (ARES)*, Aug. 2016, pp. 428–436.

[73] F. Furtado and A. Zisman, "Trace++: A traceability approach to support transitioning to agile software engineering," in *Proc. IEEE 24th Int. Requirements Eng. Conf. (RE)*, Sep. 2016, pp. 66–75.

[74] I. Khlif, M. H. Kacem, P. Stolf, and A. H. Kacem, "Software architectures: Multi-scale refinement," in *Proc. IEEE 14th Int. Conf. Softw. Eng. Res., Manage. Appl. (SERA)*, Jun. 2016, pp. 265–272.

[75] H. Xu, S. Krusche, and B. Bruegge, "Using software theater for the demonstration of innovative ubiquitous applications," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 894–897.

[76] H. Wang, M. Kessentini, W. Grosky, and H. Meddeb, "On the use of time series and search based software engineering for refactoring recommendation," in *Proc. 7th Int. Conf. Manage. Comput. collective Intell. Digit. Ecosyst.*, Oct. 2015, pp. 35–42.

[77] K. Triantafyllidis, E. Bondarev, and P. H. N. De With, "Guided rule-based multi-objective optimization for real-time distributed systems," in *Proc. 41st Euromicro Conf. Softw. Eng. Adv. Appl.*, Aug. 2015, pp. 224–232.

[78] D. Tengeri, A. Beszedes, T. Gergely, L. Vidacs, D. Havas, and T. Gyimothy, "Beyond code coverage—An approach for test suite assessment and improvement," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2015, pp. 1–7.

[79] E. F. Cruz, R. J. Machado, and M. Y. Santos, "On the decomposition of use cases for the refinement of software requirements," in *Proc. 14th Int. Conf. Comput. Sci. Appl.*, Jun. 2014, pp. 237–240.

[80] A.-R. Han and D.-H. Bae, "An efficient method for assessing the impact of refactoring candidates on maintainability based on matrix computation," in *Proc. 21st Asia–Pacific Softw. Eng. Conf.*, Dec. 2014, pp. 430–437.

[81] D. Kwon and R. J. Hammell, "Refinement/verification of early stage probabilistic software project schedules in the planning stage," in *Proc. 15th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, Jun. 2014, pp. 1–6.

[82] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III," in *Proc. Annu. Conf. Genetic Evol. Comput.*, Jul. 2014, pp. 1263–1270.

[83] J.-J. Guo, N.-L. Hsueh, W.-T. Lee, and S.-C. Hwang, "Improving software maintenance for pattern-based software development: A comment refactoring approach," in *Proc. Int. Conf. Trustworthy Syst. Appl.*, Jun. 2014, pp. 75–79.

[84] C. Napoli, G. Pappalardo, and E. Tramontana, "Using modularity metrics to assist move method refactoring of large systems," in *Proc. 7th Int. Conf. Complex, Intell., Softw. Intensive Syst.*, Jul. 2013, pp. 529–534.

[85] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki, "Visualization and exploration of optimal variants in product line engineering," in *Proc. 17th Int. Softw. Product Line Conf. (SPLC)*, 2013, pp. 111–115.

[86] Y. Y. Lee, N. Chen, and R. E. Johnson, "Drag-and-drop refactoring: Intuitive and efficient program transformation," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 23–32.

[87] H. Yang and P. Liang, "Reasoning about stakeholder groups for requirements negotiation based on power relationships," in *Proc. 20th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2013, pp. 247–254.

[88] D. Dermeval, J. Pimentel, C. Silva, J. Castro, E. Santos, G. Guedes, and A. Finkelstein, "STREAM-ADD-Supporting the documentation of architectural design decisions in an architecture derivation process," in *Proc. IEEE 36th Annu. Comput. Softw. Appl. Conf.*, Jul. 2012, pp. 602–611.

[89] X. Ge, Q. L. DuBose, and E. Murphy-Hill, "Reconciling manual and automatic refactoring," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 211–221.

[90] A. B. Fadhel, M. Kessentini, P. Langer, and M. Wimmer, "Search-based detection of high-level model changes," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2012, pp. 212–221.

[91] V. B. Singh and K. K. Chaturvedi, "Entropy based bug prediction using support vector regression," in *Proc. 12th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Nov. 2012, pp. 746–751.

[92] R. Mzid, C. Mraidha, J.-P. Babau, and M. Abid, "A MDD approach for RTOS integration on valid real-time design model," in *Proc. 38th Euromicro Conf. Softw. Eng. Adv. Appl.*, Sep. 2012, pp. 9–16.

[93] L. Chen, L. Huang, C. Li, and W. Luo, "Software architecture matching by meta-model extension and refinement," in *Proc. 19th Asia–Pacific Softw. Eng. Conf.*, vol. 1, Dec. 2012, pp. 422–427.

[94] P. Petrov, U. Buy, and R. L. Nord, "Enhancing the software architecture analysis and design process with inferred macro-architectural requirements," in *Proc. 1st IEEE Int. Workshop Twin Peaks Requirements Archit. (TwinPeaks)*, Sep. 2012, pp. 20–26.

[95] P. Conroy and P. Kruchten, "Performance norms: An approach to rework reduction in software development," in *Proc. 25th IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, Apr. 2012, pp. 1–6.

[96] S. Ahmad, "Negotiation in the requirements elicitation and analysis process," in *Proc. 19th Austral. Conf. Softw. Eng. (ASWEC)*, Mar. 2008, pp. 683–689.

[97] F. M. Gon, C. I. M. Bezerra, A. D. Belchior, C. C. Coelho, and C. G. S. Pires, "Implementing causal analysis and resolution in software development projects: The MiniDMAIC approach," in *Proc. 19th Austral. Conf. Softw. Eng. (ASWEC)*, Mar. 2008, pp. 112–119.

[98] S. Hayashi, Y. Tsuda, and M. Saeki, "Detecting occurrences of refactoring with heuristic search," in *Proc. 15th Asia–Pacific Softw. Eng. Conf.*, 2008, pp. 453–460.

[99] P. Mader, O. Gotel, and I. Philippow, "Enabling automated traceability maintenance by recognizing development activities applied to models," in *Proc. 23rd IEEE/ACM Int. Conf. Automated Softw. Eng.*, Sep. 2008, pp. 49–58.

[100] Z. Racheva, M. Daneva, and L. Buglione, "Complementing measurements and real options concepts to support inter-iteration decision-making in agile projects," in *Proc. 34th Euromicro Conf. Softw. Eng. Adv. Appl.*, Sep. 2008, pp. 457–464.

[101] S. Afsharian, M. Giacomobono, and P. Inverardi, "A framework for software project estimation based on cosmic, dsm and rework characterization," in *Proc. 1st Int. Workshop Bus. Impact Process Improvements (BiPi)*, 2008, pp. 15–24.

[102] P. Anbalagan and T. Xie, "Automated inference of pointcuts in aspect-oriented refactoring," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, May 2007, pp. 127–136.

[103] E. Mealy, D. Carrington, P. Strooper, and P. Wyeth, "Improving usability of software refactoring tools," in *Proc. Austral. Softw. Eng. Conf. (ASWEC)*, Apr. 2007, pp. 307–318.

[104] T. Neubauer and C. Stummer, "Interactive decision support for multiobjective COTS selection," in *Proc. 40th Annu. Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2007, p. 283.

[105] X. Zhao, B. S. Lerner, and L. Osterweil, "The role of context in exception-driven rework," in *Proc. 5th Int. Workshop Exception Handling (WEH)*, Jun. 2012, pp. 41–45.

[106] M. Rahimi and J. Cleland-Huang, "Patterns of co-evolution between requirements and source code," in *Proc. IEEE 5th Int. Workshop Requirements Patterns (RePa)*, Aug. 2015, pp. 25–31.

[107] J. Cuenca, F. Larrinaga, and I. Arenaza-Nuño, "A software engineering process to develop services within the arrowhead project," in *Proc. 42nd Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2016, pp. 5232–5237.

[108] R. Klashner and S. Sabet, "A DSS design model for complex problems: Lessons from mission critical infrastructure," *Decis. Support Syst.*, vol. 43, no. 3, pp. 990–1013, Apr. 2007.

[109] L.-O. Damm, L. Lundberg, and C. Wohlin, "A model for software rework reduction through a combination of anomaly metrics," *J. Syst. Softw.*, vol. 81, no. 11, pp. 1968–1982, Nov. 2008.

[110] S. Ahmad, A. K. Muda, N. A. Muda, and Z. Othman, "An approach to estimate the savings from negotiation based on cost-benefit analysis model," in *Proc. Malaysian Conf. Softw. Eng.*, Dec. 2011, pp. 298–303.

[111] D. Aceituna, H. Do, G. S. Walia, and S.-W. Lee, "Evaluating the use of model-based requirements verification method: A feasibility study," in *Proc. Workshop Empirical Requirements Eng. (EmpiRE)*, Aug. 2011, pp. 13–20.

[112] P. Meananeatra, S. Rongviriyapanish, and T. Apiwattanapong, "Identifying refactoring through formal model based on data flow graph," in *Proc. Malaysian Conf. Softw. Eng.*, Dec. 2011, pp. 113–118.

[113] J. P. Carvallo and X. Franch, "Requirements negotiation for multilayer system components," in *Proc. IEEE 19th Int. Requirements Eng. Conf.*, Aug. 2011, pp. 285–290.

[114] A. Aleti, L. Grunske, I. Meedeniya, and I. Moser, "Let the ants deploy your software–An ACO based deployment optimisation strategy," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, Nov. 2009, pp. 505–509.

[115] D. de Almeida Ferreira and A. R. da Silva, "Wiki-based tool for requirements engineering according to the ProjectIT approach," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 359–364.

[116] S. Ahmad and S. A. Asmai, "Measuring software requirements quality following negotiation through empirical study," *Int. J. Appl. Eng. Res.*, vol. 11, no. 6, pp. 4190–4196, 2016.

[117] T.-M. Hesse, V. Lerche, M. Seiler, K. Knoess, and B. Paech, "Documented decision-making strategies and decision knowledge in open source projects: An empirical study on Firefox issue reports," *Inf. Softw. Technol.*, vol. 79, pp. 36–51, Nov. 2016.

[118] H. Ghanbari, J. Similä, and J. Markkula, "Utilizing online serious games to facilitate distributed requirements elicitation," *J. Syst. Softw.*, vol. 109, pp. 32–49, Nov. 2015.

[119] T. Saika, E. Choi, N. Yoshida, A. Goto, S. Haruna, and K. Inoue, "What kinds of refactorings are co-occurred? An analysis of eclipse usage datasets," in *Proc. 6th Int. Workshop Empirical Softw. Eng. Pract.*, Nov. 2014, pp. 31–36.

[120] R. R. Souza, C. F. Chavez, and R. A. Bittencourt, "Patch rejection in firefox: Negative reviews, backouts, and issue reopening," *J. Softw. Eng. Res. Develop.*, vol. 3, no. 1, pp. 1–22, Dec. 2015.

[121] S. Singh and K. S. Kahlon, "Object oriented software metrics threshold values at quantitative acceptable risk level," *CSI Trans. ICT*, vol. 2, no. 3, pp. 191–205, Nov. 2014.

[122] S. H. Kannangara and W. M. J. I. Wijayanayake, "An empirical exploration of refactoring effect on software quality using external quality factors," *Int. J. Adv. ICT Emerg. Regions*, vol. 7, no. 2, p. 36, May 2014.

[123] J. Díaz, J. Pérez, and J. Garbajosa, "Agile product-line architecting in practice: A case study in smart grids," *Inf. Softw. Technol.*, vol. 56, no. 7, pp. 727–748, 2014.

[124] A. Yamashita and L. Moonen, "To what extent can maintenance problems be predicted by code smell detection?—An empirical study," *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2223–2242, Dec. 2013.

[125] T. G. Nair, V. Suma, and P. K. Tiwari, "Significance of depth of inspection and inspection performance metrics for consistent defect management in software industry," *IET software*, vol. 6, no. 6, pp. 524–535, 2012.

[126] G. Concas, M. Marchesi, G. Destefanis, and R. Tonelli, "An empirical study of software metrics for assessing the phases of an agile project," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 22, no. 4, pp. 525–548, Jun. 2012.

[127] D. G. Feitelson, "Perpetual development: A model of the Linux kernel life cycle," *J. Syst. Softw.*, vol. 85, no. 4, pp. 859–875, Apr. 2012.

[128] E. Murphy-Hill and A. P. Black, "Breaking the barriers to successful refactoring: Observations and tools for extract method," in *Proc. 13th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 421–430.

[129] S. Khaiyum, Y. S. Kumaraswamy, and K. Karibasappa, "Significance of failure avoidance in software development process," in *Proc. Int. Conf. Intell. Comput. Appl.*, Mar. 2014, pp. 340–344.

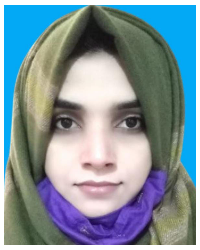
[130] T. Illes-Seifert and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs," *Inf. Softw. Technol.*, vol. 52, no. 5, pp. 539–558, May 2010.

[131] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices," *Empirical Softw. Eng.*, vol. 15, no. 6, pp. 654–693, 2010.

[132] A. Joshi, N. L. Sarda, and S. Tripathi, "Measuring effectiveness of HCI integration in software development processes," *J. Syst. Softw.*, vol. 83, no. 11, pp. 2045–2058, Nov. 2010.
[133] C. Ebert and J. D. Man, "Effectively utilizing project, product and process knowledge," *Inf. Softw. Technol.*, vol. 50, no. 6, pp. 579–594, May 2008.
[134] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk, "When and why your code starts to smell bad," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng. (ICSE)*, vol. 1, May 2015, pp. 403–414.
[135] G. Szoke, C. Nagy, P. Hegedus, R. Ferenc, and T. Gyimothy, "Do automatic refactorings improve maintainability? An industrial case study," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2015, pp. 429–438.
[136] H. Subramaniam, H. Zulzalil, M. A. Jabar, and S. Hassan, "Evaluation of early aspect formation technique for aspect refactoring," in *Proc. 9th Malaysian Softw. Eng. Conf. (MySEC)*, Dec. 2015, pp. 81–86.
[137] J. Diaz, J. Perez, J. Garbajosa, and A. Yague, "Change-impact driven agile architecting," in *Proc. 46th Hawaii Int. Conf. Syst. Sci.*, Jan. 2013, pp. 4780–4789.
[138] A. Chatzigeorgiou and A. Manakos, "Investigating the evolution of bad smells in object-oriented code," in *Proc. 7th Int. Conf. Quality Inf. Commun. Technol.*, Sep. 2010, pp. 106–115.
[139] M. A. Parande and G. Koru, "A longitudinal analysis of the dependency concentration in smaller modules for open-source software products," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2010, pp. 1–5.
[140] B. B. Chua, "Rework requirement changes in software maintenance," in *Proc. 5th Int. Conf. Softw. Eng. Adv.*, Aug. 2010, pp. 252–258.
[141] N. Ramasubbu and R. K. Balan, "The impact of process choice in high maturity environments: An empirical analysis," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, May 2009, pp. 529–539.
[142] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *J. Syst. Softw.*, vol. 81, no. 11, pp. 1868–1882, 2008.
[143] D. Damian, F. Lanubile, and T. Mallardo, "On the need for mixed media in distributed requirements negotiations," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 116–132, Jan. 2008.
[144] E. Alégroth, R. Feldt, and P. Kolström, "Maintenance of automated test suites in industry: An empirical study on visual GUI testing," *Inf. Softw. Technol.*, vol. 73, pp. 66–80, May 2016.
[145] M. B. Julian, "Artefacts and agile method tailoring in large-scale offshore software development programmes," *Inf. Softw. Technol.*, vol. 75, pp. 1–16, Jul. 2016.
[146] A. Martini, J. Bosch, and M. Chaudron, "Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study," *Inf. Softw. Technol.*, vol. 67, pp. 237–253, Nov. 2015.
[147] D. Dönmez, G. Grote, and S. Brusoni, "Routine interdependencies as a source of stability and flexibility. A study of agile software development teams," *Inf. Org.*, vol. 26, no. 3, pp. 63–83, Sep. 2016.
[148] M. Khurum, S. Fricker, and T. Gorschek, "The contextual nature of innovation—An empirical investigation of three software intensive products," *Inf. Softw. Technol.*, vol. 57, pp. 595–613, Jan. 2015.
[149] I. F. da Silva, P. A. da Mota Silveira Neto, P. O'Leary, E. S. de Almeida, and S. R. D. L. Meira, "Software product line scoping and requirements engineering in a small and medium-sized enterprise: An industrial case study," *J. Syst. Softw.*, vol. 88, pp. 189–206, Feb. 2014.
[150] M. Daneva, E. van der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar, and R. Wieringa, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1333–1353, May 2013.
[151] G. van Waardenburg and H. van Vliet, "When agile meets the enterprise," *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2154–2171, Dec. 2013.
[152] R. Hoda, J. Noble, and S. Marshall, "The impact of inadequate customer collaboration on self-organizing agile teams," *Inf. Softw. Technol.*, vol. 53, no. 5, pp. 521–534, May 2011.
[153] K. Cox, M. Niazi, and J. Verner, "Empirical study of Sommerville and Sawyer's requirements engineering practices," *IET Softw. J.*, vol. 3, no. 5, pp. 339–355, Oct. 2009.
[154] J. Paay, L. Sterling, F. Vetere, S. Howard, and A. Boettcher, "Engineering the social: The role of shared artifacts," *Int. J. Hum.-Comput. Stud.*, vol. 67, no. 5, pp. 437–454, May 2009.
[155] R. Malhotra and A. Chug, "An empirical study to assess the effects of refactoring on software maintainability," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2016, pp. 110–117.
[156] C. Vassallo, F. Zampetti, D. Romano, M. Beller, A. Panichella, M. Di Penta, and A. Zaidman, "Continuous delivery practices in a large financial organization," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2016, pp. 519–528.
[157] J. A. O. da Cunha, F. Q. da Silva, H. P. de Moura, and F. J. Vasconcellos, "Decision-making in software project management: A qualitative case study of a private organization," in *Proc. 9th Int. Workshop Cooperat. Hum. Aspects Softw. Eng.*, 2016, pp. 26–32.
[158] J. Chen, J. Xiao, Q. Wang, L. J. Osterweil, and M. Li, "Refactoring planning and practice in agile software development: An empirical study," in *Proc. Int. Conf. Softw. Syst. Process*, 2014, pp. 55–64.
[159] L. Chen and M. A. Babar, "Towards an evidence-based understanding of emergence of architecture through continuous refactoring in agile software development," in *Proc. IEEE/IFIP Conf. Softw. Archit.*, Apr. 2014, pp. 195–204.
[160] S. Koolmanojwong and J. A. Lane, "Enablers and inhibitors of expediting systems engineering," *Proc. Comput. Sci.*, vol. 16, pp. 483–491, Jan. 2013.
[161] B. Braunschweig and C. Seaman, "An examination of shared understanding in free/libre open source project maintenance," in *Proc. 6th Int. Workshop Cooperat. Hum. Aspects Softw. Eng. (CHASE)*, May 2013, pp. 113–116.
[162] G. Hanssen, A. F. Yamashita, R. Conradi, and L. Moonen, "Software entropy in agile product evolution," in *Proc. 43rd Hawaii Int. Conf. Syst. Sci.*, 2010, pp. 1–10.
[163] Z. Racheva, M. Daneva, A. Herrmann, and R. J. Wieringa, "A conceptual model and process for client-driven agile requirements prioritization," in *Proc. 4th Int. Conf. Res. Challenges Inf. Sci. (RCIS)*, May 2010, pp. 287–298.
[164] Y. Wang, "What motivate software engineers to refactor source code? Evidences from professional developers," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2009, pp. 413–416.
[165] W. G. Griswold and W. F. Opdyke, "The birth of refactoring: A retrospective on the nature of high-impact software engineering research," *IEEE Softw.*, vol. 32, no. 6, pp. 30–38, Nov. 2015.
[166] A. De Lucia and A. Qusef, "Requirements engineering in agile software development," *J. Emerg. Technol. Web Intell.*, vol. 2, no. 3, pp. 212–220, 2010.
[167] M. Wahler, U. Drofenik, and W. Snipes, "Improving code maintainability: A case study on the impact of refactoring," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2016, pp. 493–501.
[168] P. Newman, M. A. Ferrario, W. Simm, S. Forshaw, A. Friday, and J. Whittle, "The role of design thinking and physical prototyping in social software engineering," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, May 2015, pp. 487–496.
[169] B. Weitzel, D. Rost, and M. Scheffe, "Sustaining agility through architecture: Experiences from a joint research and development laboratory," in *Proc. IEEE/IFIP Conf. Softw. Archit.*, Apr. 2014, pp. 53–56.
[170] K. D. Palmer, "The essential nature of product traceability and its relation to agile approaches," *Proc. Comput. Sci.*, vol. 28, pp. 44–53, Jan. 2014.
[171] D. X. Houston and D. J. Buettner, "Modeling user story completion of an agile software process," in *Proc. 2013 Int. Conf. Softw. Syst. Process*, 2013, pp. 88–97.
[172] P. C. Brebner, "Experiences with early life-cycle performance modeling for architecture assessment," in *Proc. 8th Int. ACM SIGSOFT Conf. Quality Softw. Architectures (QoSA)*, 2012, pp. 149–154.
[173] P. Pohjalainen, "Bottom-up modeling for a software product line: An experience report on agile modeling of governmental mobile networks," in *Proc. 15th Int. Softw. Product Line Conf.*, Aug. 2011, pp. 323–332.
[174] W. Heider, P. Grunbacher, and R. Rabiser, "Negotiation constellations in reactive product line evolution," in *Proc. 4th Int. Workshop Softw. Product Manage.*, Sep. 2010, pp. 63–66.
[175] K. E. Madsen, "Collaboration strategies for distributed teams: A case study of CAD systems integration," in *Proc. 4th Int. Conf. Syst.*, 2009, pp. 222–227.
[176] P. Adamczyk, A. Zambrano, and F. Balaguer, "Refactoring big balls of mud," in *Proc. 31st Int. Conf. Softw. Eng.*, 2009, pp. 50–60.
[177] I. M. Bertran, "Detecting architecturally-relevant code smells in evolving software systems," in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 1090–1093.
[178] S. Adolph, P. Kruchten, and W. Hall, "Reconciling perspectives: A grounded theory of how people manage the process of software development," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1269–1286, Jun. 2012.

[179] R. Bellamy, M. Desmond, J. Martino, P. Matchen, H. Ossher, J. Richards, and C. Swart, "Sketching tools for ideation: NIER track," in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 808–811.

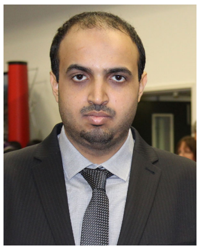
[180] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer, 2000.



MAMOONA MUMTAZ received the M.S. degree in software engineering from the COMSATS University Islamabad, Pakistan, in 2018. She is currently working as a Lecturer at the University of Management and Technology. Her research interests include change in software development, software process improvements, and human–computer interaction.

NAVEED AHMAD received the Ph.D. degree in engineering design from the University of Cambridge, in 2011. In January 2019, he joined FAST-NUCES as a Professor. His research interests include modeling and simulation, understanding the behavior of complex systems, information systems and security, software engineering, and human–computer interaction (user experience).

M. USMAN ASHRAF received the Ph.D. degree in computer science from King Abdulaziz University, Saudi Arabia, in 2018. He was a HPC Scientist at the HPC Centre, King Abdulaziz University. He is currently an Assistant Professor and the Head of the Department of Computer Science, GC Women University Sialkot, Pakistan. His research interests include exascale computing systems, high performance computing (HPC) systems, parallel computing, HPC for deep learning, and location-based services system has appeared in IEEE Access, *IET Software*, *International Journal of Advanced Research in Computer Science*, *International Journal of Advanced Computer Science and Applications*, *International Journal of Information Technology and Computer Science*, *International Journal of Computer Science and Security*, and several International IEEE/ACM/Springer conferences.



AHMED MOHAMMED ALGHAMDI He received the B.Sc. degree in computer science and the first M.Sc. degree in business administration from King Abdulaziz University, Jeddah, Saudi Arabia, in 2005 and 2010, respectively, the second master's degree in internet computing and network security from Loughborough University, U.K., in 2013, and the Ph.D. degree in computer science from King Abdulaziz University. He has over 11 years of working experience before attending the academic carrier. He is currently an Assistant Professor at the Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia. His research interests include high-performance computing, big data, distributed systems, programming models, software engineering, and software testing.

ADEL A. BAHADDAD received the B.S. degree in computer science from the Science's College, Saudi Arabia, in 2002, and the M.S. and Ph.D. degrees in information and communication technology from the School of Information and Communication Technology, Griffith University, Australia, in 2012 and 2017, respectively. He is currently an Assistant Professor with the Faculty of Computing and Information Technology, King Abdulaziz University (KAU), where he has been the Head of the Department of Systems and Educational Programs, Deanship of E-Learning and Distance Education, since 2018. He has participated in a number of executive committees concerned with automating operations at the Educational Curriculum Center and the Strategic Plan of the Strategic Center to achieve the Kingdom's vision at King Abdulaziz University. His research interests include diffusion and technology adoption and digital transformation, M-service, M-commerce, LMS, and M- government. He has many publications in these fields.

KHALID ALI ALMARHABI received the B.Sc. degree in computer science from King Abdulaziz University, Jeddah, Saudi Arabia, in 2009, the M.Sc. degree in information technology from the Queensland University of Technology, Brisbane, QLD, Australia, in 2014, and the Ph.D. degree in computer science from King Abdulaziz University and the Queensland University of Technology. He is currently an Assistant Professor at the Department of Computer Science, College of Computing in Al-Qunfudah, Umm Al-Qura University, Saudi Arabia. His research interests include information security, access control policies, information system management, and cloud computing.