How systematic reviews cover practitioners' issues: A study on Stack Exchange communities

Bruno Cartaxo [1], **Gustavo Pinto** [Corresp., 2], **Fernando Kamei** [3, 4], **Danilo Monteiro** [3], **Fabio Queda** [3], **Sergio Soares** [3]

[1] Instituto Federal de Pernambuco, Recife, Brazil

[2] Universidade Federal do Pará, Belém, Brazil

[3] Universidade Federal de Pernambuco, Recife, Brazil

[4] Instituto Federal de Alagoas, Maceió, Brazil

Corresponding Author: Gustavo Pinto
Email address: gpinto@ufpa.br

Context: One of the goals of Evidence-Based Software Engineering is to leverage evidence from research to practice. However, some studies suggest this goal has not being fully accomplished.

Objective: This paper proposes a strategy to assess how systematic reviews cover practitioners' issues in software engineering.

Method: We selected 24 systematic reviews identified by a comprehensive tertiary study. Using search strings of the selected systematic reviews, we queried most relevant practitioners' issues on five active Stack Exchange communities, a professional and high-quality Question & Answer platform. After examining more than 1,800 issues, we investigated how findings of the selected systematic reviews could help to solve (i.e. cover) practitioners' issues.

Results: After excluding false positives and duplicates, a total of 424 issues were considered related to the selected systematic reviews. This number corresponds to 1.75% of the 26,687 most relevant issues on the five Stack Exchange communities. Among these 424 issues, systematic reviews can successfully cover 14.1% (60) of them. Based on a qualitative analysis, we identified 45 recurrent issues spread in many software engineering areas. The most demanded topic is related to agile software development, with 15 recurrent issues identified and 127 practitioners' issues as a whole.

Conclusions: An overall coverage rate of 14.1% reveals a good opportunity for conducting systematic reviews in software engineering to fill the gap of not covered issues. We also observed practitioners explicitly demanding for scientific empirical evidence, rich in context and oriented to specific target audiences. Finally, we also provided guidelines for researchers who want to conduct systematic reviews more connected with software engineering practice.

How Systematic Reviews Cover Practitioners' Issues: A Study on Stack Exchange Communities

Bruno Cartaxo[1], Gustavo Pinto[2], Fernando Kamei[3,4], Danilo Ribeiro[3], Fabio Q. B. da Silva[3], and Sérgio Soares[3]

[1]**Federal Institute of Pernambuco, Brazil**
[2]**Federal University of Pará, Brazil**
[3]**Federal University of Pernambuco, Brazil**
[4]**Federal Institute of Alagoas, Brazil**

Corresponding author:
Gustavo Pinto[1]

Email address: gpinto@ufpa.br

ABSTRACT

Context: One of the goals of Evidence-Based Software Engineering is to leverage evidence from research to practice. However, some studies suggest this goal has not being fully accomplished.
Objective: This paper proposes a strategy to assess how systematic reviews cover practitioners' issues in software engineering.
Method: We selected 24 systematic reviews identified by a comprehensive tertiary study. Using search strings of the selected systematic reviews, we queried most relevant practitioners' issues on five active Stack Exchange communities, a professional and high-quality Question & Answer platform. After examining more than 1,800 issues, we investigated how findings of the selected systematic reviews could help to solve (i.e. cover) practitioners' issues.
Results: After excluding false positives and duplicates, a total of 424 issues were considered related to the selected systematic reviews. This number corresponds to 1.75% of the 26,687 most relevant issues on the five Stack Exchange communities. Among these 424 issues, systematic reviews can successfully cover 14.1% (60) of them. Based on a qualitative analysis, we identified 45 recurrent issues spread in many software engineering areas. The most demanded topic is related to agile software development, with 15 recurrent issues identified and 127 practitioners' issues as a whole.
Conclusions: An overall coverage rate of 14.1% reveals a good opportunity for conducting systematic reviews in software engineering to fill the gap of not covered issues. We also observed practitioners explicitly demanding for scientific empirical evidence, rich in context and oriented to specific target audiences. Finally, we also provided guidelines for researchers who want to conduct systematic reviews more connected with software engineering practice.

INTRODUCTION

Evidence-Based Practice (EBP) aims at investigating the best available research evidence in a given domain of expertise and integrating them to practice (74). The medicine field was the first to introduce EBP, and due to its benefits, it was adopted in fields such as psychology (10), nursing (36), crime prevention (40), social work (103), and education (33). In 2004, Kitchenham *et al.* (66) acknowledged the importance of EBP and suggested that Software Engineering (SE) community ought to adopt it. According to Kitchenham *et al.* (66), EBSE's goal is:

"*to provide the means by which current best evidence from research can be* ***integrated with practical experience*** *and human values in the decision-making process regarding the development and maintenance of software*."

After more than a decade of contributions, Evidence-Based Software Engineering (EBSE) is now a solid research field, with new studies being conducted on a regular basis (30). However, despite its clear

evolution, there are studies suggesting its stated goal was not fully accomplished (44; 88; 30). Hassler *et al.* (44) identified the lack of connection with industry as the sixth top barrier to SRs, from a total of 37 barriers listed by EBSE researchers. Santos *et al.* (88) conducted a survey with 44 authors of 120 systematic reviews and only six of them affirmed they had a direct impact on industrial practice. Da Silva *et al.* (30) conducted a tertiary study that identified only 32 out of 120 systematic reviews providing guidelines to practitioners. These studies suggest the full potential of EBSE was not entirely unlocked.

A critical step for EBSE achieving its goal is to understand to what extent the current EBSE research in general, and systematic reviews (SRs) in particular, covers issues practitioners face (18). We believe it is important to focus on systematic reviews since there are results suggesting that evidence from SRs is one of the most appropriate kinds of knowledge to be transferred to practice, whereas individual studies can often lead to contradictory conclusions (67). Starting from this premise, the research question this paper aims to answer is:

RQ. How systematic reviews cover software engineering practitioners' issues?

Regarding the research question, the following definitions are crucial:

- By **Systematic Review** (SR) we mean any kind of secondary study, such as systematic mappings, meta-analyses, and the traditional systematic literature reviews (62).
- By **coverage** we mean when at least one SR finding offers knowledge that *helps to solve* a practical issue. Nevertheless, it is important to highlight we are **not** suggesting SRs should provide definitive evidence to solve practitioners' issues. Instead, as Booth *et al.* (20) discussed, we believe research evidence can help practitioners during decision-making, ultimately helping them to solve a practical issue.
- By **practitioner's issue** we mean, a question asked on one of the Stack Exchange communities related to SE. Generally speaking, questions in Stack Exchange are composed of a title — summarizing the question — and a body — introducing further details.

Stack Exchange is a platform with over 100 high-quality, professional Question & Answer (Q&A) communities. It covers topics as diverse as Mathematics, Home Improvement, Statistics, and English Language. Software development, which is a knowledge intensive activity that requires a constant learning process (87), is particularly well-supported. Stack Overflow is certainly the most well-known Stack Exchange community, which focuses on technical coding issues. However, there are many other communities focused on different areas of SE such as software testing, quality, reverse engineering, project management, and others. The following two snippets illustrate questions asked in these communities:

- *[...] How to facilitate communication and peer reviews on a distributed scrum team?"*
- *[...] Pair programming when driver and observer have different skill level and experience [... this] strategy still work [...] if they have a very different programming skill level? If one never experience in the problem domain while another have? Is it still OK if they have low programming skill level?"*

Q&A websites empowered software engineers to increase the pace of learning, allowing them to be more productive, more effective, and more fulfilled (70; 94). The SE community has long recognized the importance of these websites, and produced contributions related to both social aspects (*e.g.,* personality traits (14), and gender (98)) and technical aspects (*e.g.,* documentation (79) and debugging (27)) of software engineering.

In this context, to provide answers to this important but neglected question, we use a coverage method proposed in a previous study (25). This method consists of matching the findings of SRs with SE related questions posted on Q&A websites. Although the previous study focused on a small set of four SRs, in this study, we greatly expanded the scope to 24 SRs previously identified by the tertiary study of Da Silva et al. (30) (which is based on two other tertiary studies (64; 65)). This tertiaty study is the most recent and comprehensive tertiary study focusing on the broad area of SE. The selected SRs vary from several topics, such as agile methods (38), usability evaluation (52), and knowledge management (19). Although other tertiary studies were published in the last few years, they are not broad in scope, as we carefully discuss in the end of the paper.

This paper presents the following contributions:

- A **strategy** to analyze how systematic reviews cover practitioners' issues.
- An empirical **study** covering more than 1,800 issues asked in five active and popular Q&A communities.
- Practical **guidelines** on how to improve systematic reviews to increase coverage of practitioners' issues;
- A reusable **dataset** related to the analysis presented in this paper (available at `http://bit.ly/2lONdj2`).

METHOD

In this section, we present the steps required to conduct this research, as depicted in Figure 1. The numbers denote each step order. Looking to major activities (gray regions), we started by selecting systematic reviews. Then we extracted the search string of these SRs. In parallel, we selected Stack Exchange communities related to SE. After that, we used the extracted search strings to find practitioners' issues at the selected Stack Exchange communities. We then excluded false positives issues. Finally, we conducted a coverage analysis, matching each practitioner's issue with SRs evidence to calculate coverage rate, and we also identified recurrent issues.

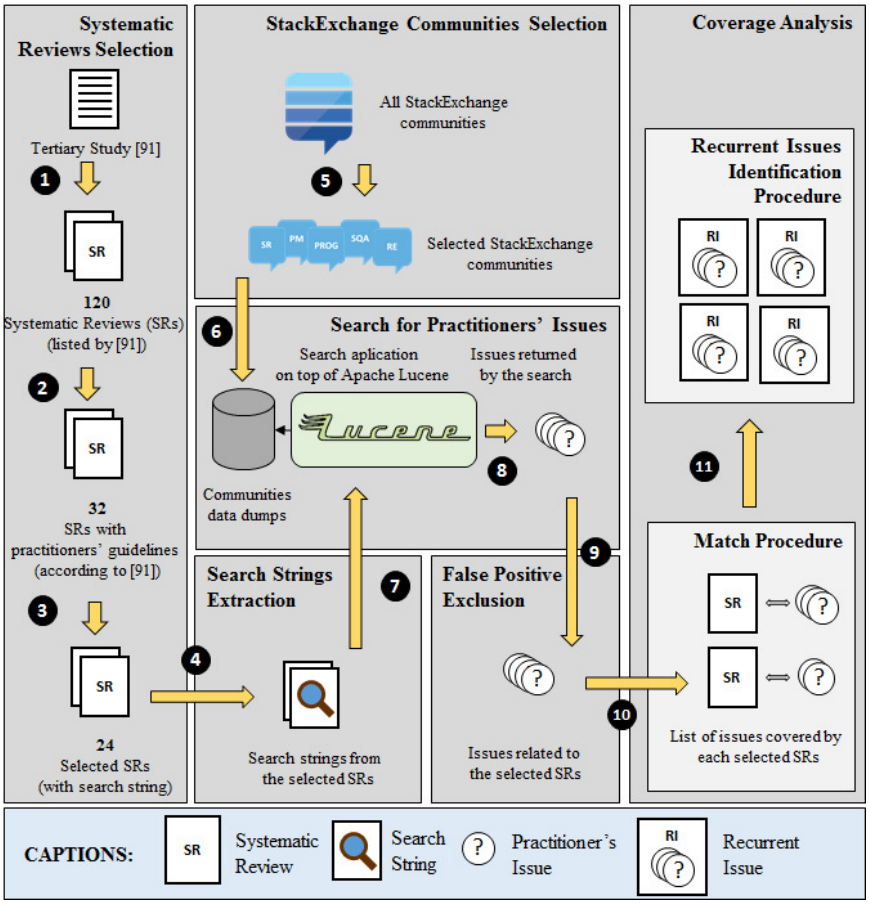

Figure 1. Research steps.

Systematic Reviews Selection

We relied on Da Silva et al.'s study (30), which is based on two other tertiary studies ((64; 65)), to select our initial group of systematic reviews. Details about why we did not use other recent tertiary studies are explained next. Our initial set is composed of 120 systematic reviews. However, we excluded SRs that:

- Do not present guidelines to practitioners, removing 88 SRs;
- Do not report their search strings, removing more 8 SRs.

Guidelines to practitioners are important because the lack of them might leave practitioners with no concrete actionable items, which should be avoided when the intention is providing evidence to practice (67). Otherwise, practitioners under time pressure have little chance to read and search immediate implications on often extensive SRs. Additionally, search strings are important for this study since we need them to search for practitioners' issues on Stack Exchange communities.

We classified each SR according to the 15 Software Engineering Areas defined in SWEBOK (21). It is necessary to define whether practitioners' issues are indeed related to the same SRs' SE area or are false positives. Table 1 presents the 24 selected systematic reviews and their respective SE areas.

Table 1. Selected systematic reviews.

CODE	TITLE	SE AREA
SR1 (63)	A systematic review of cross- vs. within- company cost estimation studies	Software Engineering Management
SR2 (86)	A systematic review of software maintainability prediction and metrics	Software Maintenance
SR3 (52)	A systematic review of usability evaluation in web development	Software Design
SR4 (102)	A systematic literature review to identify and classify software requirement errors	Software Requirements
SR5 (45)	Automated acceptance testing: A literature review and an industrial case study	Software Testing
SR6 (55)	Challenges and improvements in distributed software development: A systematic review	Software Engineering Management
SR7 (61)	Critical Barriers for Offshore Software Development Outsourcing Vendors: A Systematic Literature Review	Software Engineering Economics
SR8 (60)	Critical success factors for offshore software development outsourcing vendors: A systematic literature review	Software Engineering Economics
SR9 (75)	Definitions and approaches to model quality in model-based software development – a review of literature	Software Quality
SR10 (34)	Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review	Software Requirements
SR11 (38)	Empirical studies of agile software development: A systematic review	Software Engineering Models and Methods
SR12 (56)	Evidence-based guidelines for assessment of software development cost uncertainty	Software Engineering Management
SR13 (93)	Factors influencing software development productivity-state-of-the-art and industrial experiences	Software Engineering Economics
SR14 (57)	Forecasting of software development work effort: Evidence on expert judgement and formal models	Software Engineering Management
SR15 (49)	Harmfulness of code duplication: A structured review of the evidence	Software Construction
SR16 (19)	Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used	Software Engineering Professional Practice
SR17 (35)	Model-based testing approaches selection for software projects	Software Testing
SR18 (77)	On the generation of requirements specifications from software engineering models: A systematic literature review	Software Requirements
SR19 (69)	Risks and safeguards for the requirements engineering process in global software development	Software Engineering Management
SR20 (80)	Software process improvement in small and medium software enterprises: A systematic review	Software Engineering Process
SR21 (53)	Technology transfer decision support in requirements engineering research: a systematic review of rej	Software Requirements
SR22 (43)	The effectiveness of pair programming: A meta-analysis	Software Engineering Models and Methods
SR23 (58)	Towards a defect prevention based process improvement approach	Software Quality
SR24 (50)	Using scrum in global software development: A systematic literature review	Software Engineering Models and Methods

Search Strings Extraction

After selecting the 32 systematic reviews that presented guidelines to practitioners, we extracted their search strings to search for practitioners' issues on Stack Exchange communities. As previously mentioned, eight of those SRs were excluded because they do not present search strings, resulting in 24 selected SRs. We are aware that some SRs might employ manual search (*e.g.,* snowballing (104)), which does not need search strings. However, we found that seven out of the eight systematic reviews without search strings explicitly declared they used search engines. The remaining one did not clearly explain its search strategy.

Still, we found that nine out of the 24 selected SRs did not present their search strings properly. For instance, some studies present only a list of search terms but do not mention which logical operator (AND or OR) they used to connect each term. Others connected terms with ambiguous operators. For example, the string *"quality + model", "quality + model driven", "model driven + experience"* used "+" (plus) and "," (comma), which we considered as OR and AND, respectively. Another study used the operator "WITH", for instance, *(software AND ((cost OR effort OR productivity) WITH (factors OR indicators OR drivers OR measure)))*, which we considered as AND.

Stack Exchange Communities Selection

In this step, we selected Stack Exchange communities to search for practitioners' issues. We restricted our search to communities related to at least one of the 15 SWEBOK's SE areas (21). To establish the relationship, we compared the official communities' descriptions with each software engineering area.

For this study, we selected five Stack Exchange communities from more than 160 active communities[1]. Table 2 shows the relationship between the five selected Stack Exchange communities and the software engineering areas.

Table 2. Selected Stack Exchange communities.

COMMUNITY	COMMUNITY DESCRIPTION	SE AREAS
Programmers (PROG) (PRO)	Q&A for professional programmers who are interested in getting expert answers on conceptual questions about software development	Software Design Software Construction
Project Management (PM) (PMS)	Q&A for project managers	Software Management
Quality Assurance & Testing (SQA) (SQA)	Q&A for software quality control experts, automation engineers, and software testers	Software Testing Software Quality
Reverse Engineering (RE) (RES)	Q&A for researchers and developers who explore the principles of a system through analysis of its structure, function, and operation	Software Maintenance
Software Recommendations (SREC) (SRS)	Q&A for people seeking specific software recommendations	Software Tools

Stack Exchange maintains a staging zone, entitled *Area 51*[2], intended to receive requests to create new communities, as well as monitor ing a set of metrics to assess how well existing communities are. According to the website, communities with answers/question ratio above three are considered *good*, and above one are *okay but need improvement*. Table 3 shows some numerical data about the selected communities.

Table 3. Selected communities' numerical data.

COMMUNITY	# QUESTIONS	# ANSWERS	A/Q
PROG	35,560	128,199	3.6
PM	2,362	8,420	3.56
SQA	2,642	6,333	2.39
RE	1,745	2,751	1.57
SREC	4,434	4,894	1.1
TOTAL	46,743	150,597	3.22

As we can see in Table 3, three selected communities are classified as *good*, and the remaining ones are *okay but need improvement*. *Area 51* monitors other metrics, such as *number of visits per day* and *number of avid users*. For instance, RE and SREC are considered excellent in these metrics, with 199 avid users and 1,905 visits per day, and 394 avid users and 4,384 visits per day, respectively. We did not use the most well-known Stack Exchange community, Stack Overflow, because it focuses mainly on technical and coding issues, which is often out of the scope of SRs.

Search for Practitioners' Issues

We search for practitioners' issues in the selected Stack Exchange communities using search strings extracted from selected SRs. We implemented a search application on top of Apache Lucene[3], a highly scalable search library, and used the search strings as input to find issues on Stack Exchange data dump.

To select high-quality issues, we filtered them based on their score. In Stack Exchange, a user can "up-vote" an issue if s/he thinks it is relevant, or "down-vote" otherwise. The score is the resulting value of this voting process. The score is also commonly used as a metric for choosing relevant and high-quality

[1] `stackexchange.com/sites`
[2] `area51.stackexchange.com`
[3] https://lucene.apache.org/core

issues (*e.g.,* (96)). Considering the median score as the threshold, we selected only issues above or equal to their respective communities median score. We decided to adopt the median score instead of, for instance, the mean score because scores have no upper or lower limits. Thus, outliers affecting the mean are common. Table 4 shows descriptive statistics about the issues' scores for each selected communities. Hereafter, we will use the term **more relevant issues** for issues with score above or equal the median score of their communities.

Table 4. Issues' scores descriptive statistics.

COMMUNITY	MEDIAN	MEAN	S.D.	MAX	MIN
PROG	3	7.2	22.1	2,189	-11
PM	3	4.3	5.3	80	-5
SQA	1	2.5	4.1	72	-8
RE	2	3.6	5.2	76	-8
SREC	2	3.4	4.4	75	-4

As we can see in Table 4, the standard deviation (S.D.) is larger than the mean in all cases, which corroborates the decision of not choosing the mean as a measure for selecting practitioners' issues on Stack Exchange communities.

At the time of the data dump (August 18, 2015), the five selected Stack Exchange communities had a total of 46,743 issues. Among them, 26,687 issues have scores above the median of their communities. From these issues, 1,860 (7%) were found using the search strings of our 24 selected SRs. Table 5 depicts the number of returned issues for each selected SR, per Stack Exchange community.

Table 5. Number of returned issues for each selected SR, per Stack Exchange community.

SR	PROG	PM	RE	SREC	SQA	TOTAL
SR14 (57)	471	78	0	8	19	576
SR11 (38)	257	84	0	30	24	395
SR18 (77)	161	25	0	54	16	256
SR15 (49)	147	0	1	6	3	157
SR5 (45)	106	12	0	18	20	156
SR21 (53)	87	14	0	3	11	115
SR22 (43)	75	6	0	1	1	83
SR9 (75)	17	4	0	1	7	29
SR7 (61)	20	5	0	0	1	26
SR13 (93)	17	2	0	1	2	22
SR16 (19)	14	5	0	1	0	20
SR24 (50)	7	2	0	0	2	11
SR8 (60)	5	2	0	0	0	7
SR10 (34)	3	0	0	0	0	3
SR3 (52)	2	0	0	0	0	2
SR20 (80)	1	0	0	0	0	1
SR1 (63)	1	0	0	0	0	1
SR4 (102)	0	0	0	0	0	0
SR2 (86)	0	0	0	0	0	0
SR6 (55)	0	0	0	0	0	0
SR17 (35)	0	0	0	0	0	0
SR19 (69)	0	0	0	0	0	0
SR23 (58)	0	0	0	0	0	0
SR12 (56)	0	0	0	0	0	0
TOTAL	1,391	239	1	123	106	1,860

Some systematic reviews did not return any issue

Among the 24 selected systematic reviews, the search string of seven (29%) of them did not return any issue. We identified three reasons that might explain this fact:

- **The search string has too many key terms:** This makes the search too specific, making it difficult to find issues with all key terms in its title or body since they are connected with the restrictive AND operator. For instance, there is a systematic review with 13 terms connected by AND operators in its search string. SR2 and SR4 search strings have this problem.

- **The search string has key terms with no synonyms:** This kind of search string leaves no room for issues using different words. For instance, consider the following search string: *"uncertainty assessment" AND motivation*. This search string does not admit a synonym for any of its two terms, which prevent to find issues that use different words. SR12 and SR19 search strings have this problem.

- **The search string has key terms with composed synonyms only:** A composed synonym comprises more than one word, and the order of each word matters.To illustrate a composed synonym for a key term like "requirements" could be "software requirements". Thus, when all the synonyms are composed it forces the issues' title or body to have the same words of the composed synonym in the same order. For instance, lets take the following synonyms of a key term from a search string of a selected systematic review: *"model based test" OR "model based testing" OR "model driven test" OR "model driven testing" OR "specification based test" OR "specification based testing" OR "specification driven test" OR "specification driven testing" OR "use case based test" OR "use case based testing" OR "use case driven test" OR "use case driven testing" OR "UML based test" OR "UML based testing" OR "UML driven test" OR "UML driven testing" OR "requirement based test" OR "requirement based testing" OR "requirement driven test" OR "requirement driven testing" OR "finite state machine based test" OR "finite state machine based testing" OR "finite state machine driven test" OR "finite state machine driven testing"*. It has many synonyms, but they are all composed and very specific, reducing the probability to find an issue that has, at least, one of them. SR6, SR17, and SR23 search strings have this problem.

Stack Exchange communities characteristics

Based on the search, we observed the following characteristics about the selected Stack Exchange communities:

- **PROG**: This community returned the highest number of issues (74.7%). This was not surprising since this community is the one with higher number of issues posted by practitioners. Despite its name – Programmers – it is clearly focused on conceptual design and programming issues such as good practices, design patterns, and architectural trade-offs.

- **PM**: This community returned the second highest number of issues (12.8%) with the search strings of the selected SRs. Additionally, in this community, we could found many issues about diverse software engineering areas, beyond the scope of software project management, such as software requirements, software testing, and others.

- **SQA**: In this community issues are indeed focused on quality assurance and testing, and it is not common finding issues from other topics.

- **SREC**: This community returned few issues, even though it has the second highest number of issues reported by practitioners, as shown in Table 3. Additionally, we could observe that many of those few issues were, in fact, false positives. In the end,the majority of issues posted in this community are recommendations requests about software applications in general, such as applications to burn DVDs, or to remotely access a PC, to mention a few. Requests for tools to support software engineering practices, such as IDEs, test automation tools, or bug trackers are usually posted on the other communities focused on software engineering areas, such as in PROG, PM, SQA, and RE. Therefore, we believe this community should not be considered for further studies that want to investigate software engineering issues.

- **RE**: This community returned only one issue, and was the one with the lowest amount of issues returned by the search. On the other side, only one of the selected SRs ((86)) was considered as belonging to *Software Maintenance*, which is the broader area that comprises reverse engineering according to SWEBOK (21). Thus, we believe further investigations are important to understanding whether this community is a good choice to investigate SE issues.

False Positive Exclusion

Some studies reported high-rate of false positives when searching for issues in Stack Exchange communities (59; 81). For instance, in Pinto et al.'s work (81), the authors observed about 50% of the initially selected questions were, in fact, false positives. To remove them, we classified each practitioners' issues as *Related* or *Not Related* to the same SE area of the selected SRs, as defined in Table 1. That is why we classified each selected SRs based on the SWEBOK SE areas, as previously explained. To avoid misclassification, this procedure was conducted in pairs, followed by conflict resolution meetings. After classification, we analyzed *Not Related* issues to understand the reasons why they were returned by the search.

Our search returned a total of 1,860 issues using the search strings of the selected SRs. Table 6 shows the result of false positive exclusion. The seven systematic reviews which did not return any issues by the search – SR4, SR2, SR6, SR17, SR19, SR23, SR12 – were omitted from the table. We performed an agreement analysis using the Kappa statistic (101). The Kappa value was 0.85, which means an *Excellent Agreement* level according to the Kappa reference table (101).

Table 6. Number of issues *Related* and *Not Related* to the selected systematic reviews.

2*SR	RELATED		NOT RELATED	TOTAL
	#	%	#	#
SR11 (38)	217	54.9%	178	395
SR22 (43)	45	54.2%	38	83
SR18 (77)	41	16%	215	256
SR15 (49)	34	21.6%	123	157
SR21 (53)	31	26.9%	84	115
SR5 (45)	24	15.3%	132	156
SR14 (57)	23	4%	553	576
SR16 (19)	15	75%	5	20
SR7 (61)	14	53.8%	12	26
SR24 (50)	7	63.6%	4	11
SR13 (93)	6	27.2%	16	22
SR9 (75)	5	17.2%	24	29
SR8 (60)	5	71.4%	2	7
SR1 (63)	1	100%	0	1
SR10 (34)	0	0%	3	3
SR3 (52)	0	0%	2	2
SR20 (80)	0	0%	1	1
TOTAL	468	25.1%	1,392	1,860

As we can see, 1,392 issues were discarded due to being considered as *Not Related* to the selected SRs, *i.e.,* false positives. After discarding false positives, we ended up with 468 practitioners' issues. This set represents 1.75% of the 26,687 more relevant issues of the five Stack Exchange communities. This result might indicate a gap between topics explored with SRs and ones demanded by practitioners.

Moreover, ten out of the 17 SRs presented a high rate of false positives (less than 50% of *Related* issues). For instance, no issue was related to SR3, SR10, and SR20. We found a scenario that may explain this situation:

- **Systematic reviews using rather common terms in their search strings:** For instance, SR18 (77) uses terms such as "from", "documents", and "features" as part of the search string. These terms are likely to appear in other contexts beyond requirements specifications, which is the SR focus. As

a result, 215 (84%) out of the 256 issues, returned by the search, were classified as *Not Related* to this particular SR. A similar situation was observed in Kavalers' *et al.* study (59), where they looked for issues that reported Java classes with the term "security" in their name, and many issues were found because the term "security" is often mentioned in posts not related with the Security classes, rising the number of false positives.

Coverage Analysis

After excluding false positives, we conducted the coverage analysis based on qualitative techniques (91). The analysis is grouped into two parts: *match procedure* and *recurrent issues identification procedure*. To avoid bias, the entire coverage analysis was conducted by one researcher, and revised by another. With this analysis, we could identify which issue is covered, and which is not, mapping gaps between systematic reviews and issues asked by practitioners. Figure 2 depicts the entire coverage analysis.

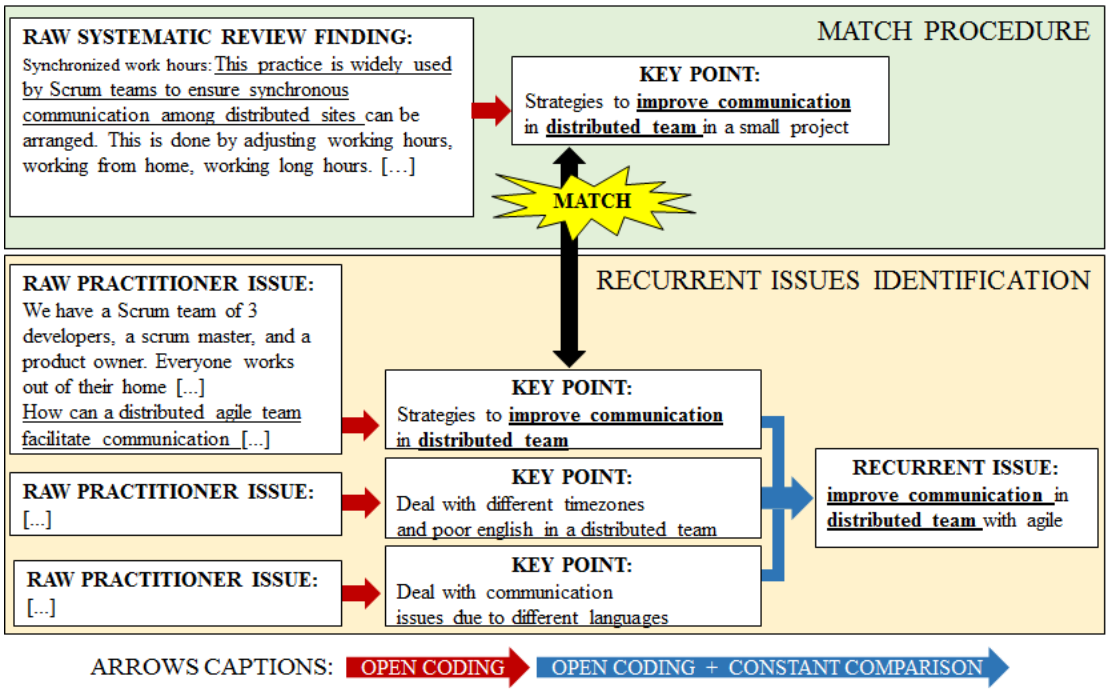

Figure 2. Coverage Analysis Procedure

Match Procedure

In the match procedure, we analyze issue per issue comparing them to the findings of the systematic review it is related to. Steps comprising this procedure are:

1. We extracted findings of each SR and applied open coding techniques (91) to define their *Key Points*;
2. We analyzed practitioners' issues related to each SR and applied open coding techniques (91) to defined their *Key Points*;
3. We matched the Key Point of each issue against the SR findings' Key Points, establishing whether an issue is covered or not.

The *Key Points* are codes that summarize, in few words, both practitioners' issues from Stack Exchange and findings from systematic reviews. To illustrate the match procedure, Table 7 shows some examples of issues considered as covered by findings of SRs their are related to. The issue's title starts with symbol "💬", and its body starts with symbol "▪▪▪".

Another important step is that after the match procedure when we exclude duplicated issues. Duplication occurs when more than one SR is related to the same issue. We decided to exclude duplicated issues

Table 7. Examples of issues covered by findings of systematic reviews.

PRACTITIONER'S ISSUE	SR'S FINDING COVERING THE ISSUE
2* **When does pair programming work? When to avoid it?** ▪▪▪ Rather than slavishly pair program all the time, we use pair programming selectively on our team. [We] think it works best in [some] circumstances [...] When to use pair program and why? When to avoid pair programming? Why?	3* *"If you do not know the seniority or skill levels of your programmers, but do have a feeling for task complexity, then employ pair programming either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important."* SR22 (43)
2* **What makes Agile software development so appealing?** ▪▪▪ Agile software development is becoming a pretty fun buzzword these days [...] what are the biggest reasons for choosing to do Agile development [...]	8* *"Most studies reported that agile development practices are easy to adopt and work well. Benefits were reported in the following areas: customer collaboration, work processes for handling defects, learning in pair programming, thinking ahead for management, focusing on current work for engineers, and estimation [...] benefits in projects that use agile methods because changes are incorporated more easily and business value is demonstrated more efficiently [...]"* SR11 (56)
3* **How to facilitate communication and peer reviews on a distributed scrum team?** ▪▪▪ We have a Scrum team of 3 developers, a scrum master, and a product owner. Everyone works out of their home [...] How can a distributed agile team facilitate communication [...]	6* *"Our SLR has found that Scrum teams use various practices or strategies to reduce these challenging factors to support the use of Scrum practices in globally distributed projects. This review has identified and categorized these practices as follows: Synchronous communication [...] Team Collaboration [...] Communication bandwidth [...] Tool Support [...] Team management [...] Office space [...]"* SR24 (50)

after the match procedure to guarantee that if any SR related to that issue presents a finding helping to solve it, the issue would be considered covered. From the 468 related issues, there were 24 duplicated. Thus, **we ended up with 424 practitioners' issues**.

Recurrent Issues Identification Procedure

At the end of the *Match Procedure* we calculate the overall coverage rate. However, due to the high number of issues, it would be hard to draw meaningful insights beyond the quantitative of covered issues and the overall coverage rate. That is why we also conduct the *Recurrent Issues Identification Procedure* aiming to aggregate issues that report the same problem and provide a manageable list of recurrent issues practitioners face in practice. Steps comprising this procedure are:

1. We identified recurrent issues applying open coding and constant comparison techniques (91);
2. We classified each recurrent issue according to SWEBOK (21) SE areas.

A **recurrent issue** groups two or more issues about the same problem. The aggregation of issues as recurrent issues enables us to identify common problems in practice and also manage the coverage analysis. An example of a recurrent issue we identified is shown in Figure 2.

RESULTS

This section presents the results of this research.

Overall Coverage

A total of 60 practitioners' issues (14.1%) are covered by the selected SRs, revealing a good opportunity for conducting systematic reviews in software engineering to fill the gap of not covered issues.

We identified two reasons that explain why the SRs do not cover practitioners' issues:

- **Issues reporting scenarios systematic reviews do not fully cover**: Majority of not covered issues fit under this situation, when SR is related to the same SE area of a practitioner's issue, but there are no findings helping to solve it. To illustrate, a user asked *"if pair programming works in case of pairs with different programming skill levels."* This issue is related to SR22 (43) that investigates pair programming effectiveness. However, the issue was considered not covered because the SR investigated only pairs with the same skill level. This could be avoided if the SR had analyzed pairs with different skill levels. Such analysis would, certainly, be limited primary evidence availability.

- **Issues using approaches available/popular after the systematic review was conducted**: For instance, a user asked *"Is BDD actually writable by non-programmers? "*. Although related to agile methods – a topic addressed by some selected SRs – BDD was introduced only in 2012, slightly after selected SRs publication. So, there is no chance the selected SRs would cover that kind of issue.

Nine practitioners' issues particularly took our attention. They reveal practitioners are demanding scientific evidence, which might indicate "bridges" need to be built to transfer knowledge from empirical studies to SE practice (15; 23; 26; 54). The following two issues are given as examples:

> 💬 **Are there any studies of cross-functional teams vs. domain-based teams (*e.g.*, project-based vs. software/mechanics/etc)?**
> ▪▪▪ I work in an organization which creates many integrated systems products - i.e. it is complete products with mechanical/system/electronics/software being designed and manufactured. At the moment most teams are organized around projects in a cross functional way. The advantage of an organization like this is that people who are working closely together for a common goal are close. The disadvantages come from the isolation of engineers from their peers. Typically a project is assigned only one software engineer. This means that the projects have a high truck factor, minimal knowledge sharing and best practices, and technical development is limited. So my question is: **are there any studies comparing the cost/benefits of these two approaches?**

> 💬 **Should there be more scientific study of the effectiveness of various hyped-up ideas in software development?**
> ▪▪▪ Everyone seems to implicitly assume that the free market of ideas will eventually converge on the "right" solutions in software development. We don't assume that in medicine - we recognise that scientific experiments are needed there - so why should we assume it in software development? I am not arguing for regulation of programmers. It is far too early to even talk about that. Before healthcare could be effectively regulated, there was a need for scientific experiments to establish which treatments worked and which didn't. Software engineering doesn't even have this scientific evidence base to back up touted methodologies such as Scrum or Agile, or programming paradigms such as functional programming or MDA. [...] **The question is, why is this scientific evidence base (for all intents and purposes) nonexistent?**

Software Engineering Areas Coverage

Table 8 shows how practitioners' issues are covered by the selected SRs separated by each of the 15 software engineering areas according to SWEBOK (21). None of the SE areas presented a coverage rate above 50%.

The SE area that we found more issues is **Software Engineering Models and Methods**, with 127 issues in total. Three SRs present findings in that area, they are: SR11 (38) , SR22 (43), and SR24 (50) as shown in Table 1. It is the SE area with the highest number of covered issues, 17 in total (13.3%). All the issues from that SE area are related to agile methods/practices. The three most recurrent issues are: Applicability of agile in specific project context; Mixing agile with traditional methods/practices;

Table 8. Issues coverage per each SWEBOK (21) Software Engineering area.

2*SOFTWARE ENGINEERING AREA	COVERED		TOTAL
	#	%	#
Software Engineering Models and Methods	17	13.3%	127
Software Engineering Management	3	4.6%	64
Software Engineering Professional Practice	12	24%	46
Software Requirements	6	15.7%	38
Software Testing	8	26.6%	30
Software Engineering Process	4	20%	20
Software Construction	2	12.5%	16
Software Engineering Economics	4	36.3%	11
Software Maintenance	2	22.2%	9
Software Design	0	0%	9
Software Configuration Management	0	0%	0
Software Quality	0	0%	0
Computing Foundations	0	0%	0
Mathematical Foundations	0	0%	0
Engineering Foundations	0	0%	0
DEFINITION	2	3.7%	54

and Benefits of agile methods/practices in general. More details about recurrent issues classified under Software Engineering Models and Methods area are shown next.

The second SE area we found more issues is **Software Engineering Management**, with 64 issues in total, less than half of what was found in Software Engineering Models and Methods. It is also the SE area with highest number of related SRs: five in total. They are: SR1 (63), SR6 (55), SR12 (56), SR14 (57), and SR19 (69). However, only three practitioners' issues were covered, resulting in the lowest coverage rate among all SE areas, 4.6%. The three most recurrent issues of this SE area were: Strategies to cost and effort estimation in specific contexts; Tools to support project management with specific features; and Strategies and metrics to measure team productivity. More details about recurrent issues classified under Software Engineering Management area are shown aftwerwards.

The third SE area we found more issues is **Software Engineering Professional Practice**, with 46 issues in total. Just one selected SRs is related to this area, SR16 (19). Eleven practitioners' issues were considered as covered in this area, which corresponds to 24% coverage rate. The three most recurrent issues of this SE area were: Strategies to deal with knowledge management in a team; Difficulties dealing with customer in specific contexts; Improving communication in a distributed team. More details about the recurrent issues classified under Software Engineering Professional Practice area are shown in Section afterwards.

Issues asking for information about simple concepts like *"In pair programming, what is each role named, and why?"*, were classified as **DEFINITION** since these kind of issues are better covered by the basic literature, rather than research evidence provided by SRs.

Recurrent Issues Coverage

In this section, we present the coverage of each recurrent issue we identified organized per each SE area. The five most recurrent issues — the ones that aggregate highest number of practitioners' issues — are:

1. **Applicability of agile in specific project context** (19 issues): Aggregates issues about whether a specific agile method/practice is applicable in a specific practical context.

2. **Introducing and adapting a software development process in specific context** (14 issues): Aggregates issues about how to introduce or adapt a software process to a specific practical context.

3. **Strategies to cost and effort estimation in specific contexts** (13 issues): Aggregates issues about strategies to estimate cost and effort in a specific practical context.

4. **Mixing agile with traditional methods/practices** (10 issues): Aggregates issues about how to deal with environments where agile and traditional methods/practices need to live together.

5. **Tools to support software testing with specific features** (10 issues): Aggregates issues about tools to support software testing with specific features.

We now discuss each recurrent issue per SE area. For each SE area, we group recurrent issues, we quote at least one issue, and we discuss why that issue was covered or not covered. If the issue was not covered, we provide discussions on how SRs could evolve to cover the issue. The *Miscellaneous* category aggregates issues that do not fit in any of the identified recurrent issues, although belonging to a SE area.

Software Requirements (6 recurrent issues)

Table 9 shows recurrent issues related to software requirement area. None of the recurrent issues presented a coverage higher than 50%.

Table 9. Coverage of software requirements recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Tools to manage requirements with specific features	Software Requirements Tools. Chapter 1, Section 8	3	37%	8
Approaches to manage constant requirements change	ChangeManagement. Chapter 1, Section 7.2	0	0%	6
Select requirements elicitation techniques in specific contexts	ElicitationTechniques. Chapter 1, Section 3.2	1	20%	5
Strategies to prioritize requirements in specific contexts	RequirementsClassification. Chapter 1, Section 4.1	0	0%	3
Defining requirements attributes in specific contexts	RequirementsAttributes. Chapter 1, Section 7.3	0	0%	3
User stories to specify non-functional requirements	ElicitationTechniques. Chapter 1, Section 3.2	0	0%	2
Miscellaneous	—	2	18%	11

The most recurrent issue is the need of **Tools to manage requirements with specific features**, with eight issues grouped under this classification. SR18 (77) and SR21 (53) provided evidence about tools to manage software requirements that could cover three out of eight practitioners' issues which corresponds to a coverage rate of 37%. To illustrate, one example of issue follows:

> **What FOSS solutions are available to manage software requirements?**
> In the company where I work, we are starting to plan to be compliant to the software development life cycle. We already have, wiki, vcs system, bug tracking system, and a continuous integration system. The next step we want to have is to start to manage, in a structured way, software requirements. [...] **We are trying to search and we hope we can find and use a FOSS software to manage all this things.** We have about 30 people, and don't have a budget for commercial software [...] **Required features:** Software requirements divided in a structured configurable way; Versioning of the requirements (history, diff, etc, like source code); Interdependency of requirements (child of, parent of, related to); Rule Based Access Control for data handling;[...]

To mitigate this issue, we believe systematic reviews aimed at identifying tools, comparing their features, or assessing their effectiveness, for software requirements engineering practice might play an important role. One example is the study conducted by Marshall *et al.* (73) that identified, analyzed, and compared tools based on their features. However, the study analyzed tools to support systematic reviews in SE, not software requirements.

The second most recurrent issue is the need of **Approaches to manage constant requirements change**, with six issues classified. Unfortunately, none SRs were able to provide useful information helping to solve the issues. To illustrate, following there is one of those issues:

> **How do you deal with the costs of too-rapid change?**
> Like most modern developers I value Agile principals like customer collaboration and responding to change, but what happens when a product-owner (or whoever determines requirements and priorities) changes requirements and priorities too often? Like several times a day? [...] **Is there some [...] in-depth study, metaphor, or quote that can help me reduce the amount of wasted effort or at least explain the costs of this chaotic behavior?**

It is interesting to note that this second most recurrent issue may challenge one of the building blocks of agile software development, which is *"Responding to change over following a plan"* (agi). We noticed that ideal balance between embracing changes and proper planning seems not to be fully accomplished in practice. Systematic reviews aimed at identifying and analyzing strategies to manage requirements change, comparing their pros and cons, might improve practitioners' confidence facing such kind of scenario.

The next three most recurrent issues present a peculiar characteristic. They report particular problems but focuses on the necessity of information that fits in their specific contexts, they are: Recommendations on how to **Select requirements elicitation techniques under specific contexts**; **Strategies to prioritize requirements in specific contexts**; and recommendations on **Defining requirements attributes in specific contexts**. Those three recurrent issues reinforce many claims about importance of highly contextualized evidence to provide useful information to practitioners (39). A systematic review identified mechanisms to characterize context of primary studies in software engineering (24). However, as far as we know there are no guidelines to support context characterization of evidence for a systematic review.

Software Design (2 recurrent issues)

We identified two recurrent issues related to software design. However, none were covered by the selected SRs, as can be seen in Table 10.

Table 10. Coverage of software design recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Benefits of a service layer, when compared to libraries	Other Methods. Chapter 2, Section 7.6	0	0%	4
Strategies to structure user interface design	User Interface Design. Chapter 2, Section 4	0	0%	2
Miscellaneous	—	0	0%	3

The most recurrent issue of software design area is related to **Benefits of a service layer, when compared to libraries**. None selected SRs provide evidence that could cover this kind of issue. To illustrate, following there is one of the four issues under this classification:

> **How essential is it to make a service layer?**
> I started building an app in 3 layers (DAL, BL, UI), it mainly handles CRM, some sales reports and inventory. A colleague told me that I must move to service layer pattern, that developers came to service pattern from their experience and it is the better approach to design most applications. He said it would be much easier to maintain the application in the future that way. Personally, I get the feeling that **it's just making things more complex and I couldn't see much of a benefit from it that would justify that**[...]

The plethora of different service layers, as well as the rich set of software libraries found in any high-level programming language makes such investigation challenging. A systematic review aimed at studying this particular scenario might present evidence for practitioners facing this kind of situations. However, the existence of such systematic review is limited by the presence of primary studies related to this topic (*e.g.,* (78)).

Software Construction (2 recurrent issues)

We found two recurrent issues related to the software construction area, as shown in Table 11.

Table 11. Coverage of software construction recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Object modeling techniques	ConstructionDesign. Chapter 3, Section 3.1	0	0%	7
Code duplication avoidance	ConstructionforReuse. Chapter 3, Section 3.5	2	40%	5
Miscellaneous	-	0	0%	4

The only recurrent issue covered refers to **code duplication avoidance** with a 40% coverage rate. To illustrate, following there is one issue under that classification:

> **When is 'cloning', rather than reusing, a module acceptable design solution?**
> ▪▪▪ For this question, I'll give an example module to facilitate the discussion, Let's say the module is a calculation engine, It currently servers its purpose for its current audience. The requirement is to clone the same engine but with some tweaking for an entirely new audience. Given that, these are Considerations/Factors that will affect the design solution: [...] However, **I am still conflicted, since: It will inherently be a copy-paste solution; Duplicate code;**[...] Is the compromise acceptable in this situation, given the user expectations highlighted above? And follow up question, is there something I can add to the solution that will address the conflicting issues [...]

Code clones have a long history in software engineering research, with traditional studies dating from the 1990s (*e.g.,* (13)). SR15 (49) is an example of a systematic review that investigates code clones, providing evidence and guidance for practitioners. In particular, it builds a model to demonstrate under which circumstances code duplication harm system quality. It also provides strategies to mitigate each of these situations. Since this recurrent issue covers a wide spectrum of code duplication, this might explain this recurrent issue coverage.

Software Testing Coverage (1 recurrent issue)

Among the 30 practitioners' issues regarding software testing, we identified only one recurrent issue, which is **Tools to support software testing with specific features**. Ten out of the 30 issues were classified in this recurrent issue, although only 3 of them are covered by the selected systematic reviews, as can be observed on Table 12. SR5 (45) offers evidence about FiteNesse (fit) and other tools related to automated acceptance testing, covering all the three issues. One who wants to offer evidence aiming to fill that gap can adopt that same strategy we employed that suggests systematic reviews comparing tools. Following there is one issue classified under this recurrent issue.

Table 12. Coverage of software testing recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Tools to support software testing with specific features	Software Testing Tools. Chapter 4, Section 6	3	30%	10
Miscellaneous	-	5	25%	20

> **Fitnesse vs Robot**
> ▪▪▪ **We are choosing what system to start using in our company**. it should be used for both backend (REST API, some DB checks) and UI testing; it should use a simple language so even non-programmers/tester can understand the test cases (Product Owners should be able to see whether all acceptance criteria are covered);it should support integration with Jenkins; it should support versioning of test cases so that for a particular product version we also can check out relevant test cases; right now we use TestRail (test case management SW) [...]

Software Maintenance Coverage (2 recurrent issues)

We identified two recurrent issues under the area of software maintenance. Together they group nine practitioners' issues as shown in Table 13.

Table 13. Coverage of software maintenance recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Developers demanding refactor the entire legacy system	Reengineering. Chapter 5, Section 4.2	2	40%	5
Strategies to perform refactoring	Reengineering. Chapter 5, Section 4.2	0	0%	4

The most recurrent issue is the one **Developers are demanding refactor the entire legacy system**, with five issues under this classification. This shows maintenance activities are still problematic due to either bad software design or construction or to a culture among software developers that prefer to spend effort reinventing the wheel instead of understanding and evolving a legacy system (51). Following there is one issue that illustrates this situation:

> **Reengineering the project from scratch**
> I am currently working on a project that has been in development for the last few years used throughout the organization but the way the project has been coded the maintainability of it is completely shot. Reading the code presents with pages and pages of Anti-Patterns and trying to identify the path of a business workflow takes on occasion days. At this point I would probably classify the software in its current state as "Working by accident" rather then as intended. So I am looking for some wisdom as to the following: **At what point would you consider simply dumping the project into an abandonware pile and starting from scratch?** [...]

Evidence that supports decision-making during software design and construction could help software developers to design systems with higher maintainability. For cases where developers want to reinvent the wheel, proper training can help to reduce the impetus to re-implement a system from scratch (51).

Software Engineering Management Coverage (6 recurrent issues)

Software engineering management is the second area with more issues related to it, with a total of 64 issues. We could identify six recurrent issues among them. However, four are not covered at all, and two present a coverage rate below 50%, as shown in Table 14.

Table 14. Coverage of software engineering management recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Strategies to cost and effort estimation in specific contexts	Effort, Schedule, and Cost Estimation. Chapter 7, Section 2.3	1	7%	13
Tools to support project management with specific features	Software Engineering Management Tools. Chapter 7, Section 7	0	0%	6
Strategies and metrics to measure team productivity	Reviewing and Evaluating Performance. Chapter 7, Section 4.2	1	25%	4
Strategies to manage distributed teams	Software Engineering Management. Chapter 7	0	0%	4
Tasks that do not fit in one sprint	Effort, Schedule, and Cost Estimation. Chapter 7, Section 2.3	0	0%	4
Strategies to negotiate project scope	DeterminationandNegotiation of Requirements. Chapter 7, Section 1.1	0	0%	3
Miscellaneous	-	1	3.3%	30

The most recurrent issue is practitioners asking for **Strategies to cost and effort estimation in specific contexts**. Only one out of 13 issues is covered. SR12 (56) and SR14 (57) are related to cost and effort estimation. However, SR12 did not return any issue from the search, and SR14 is focused on the comparison of expert judgment versus formal models to estimate effort in general. On the other side, issues are context specific. An example of one issue classified under this recurrent issue is:

> **How does a team (new to product and domain) estimate user stories of a ten year old product?**
> I am the scrum master for one of the products in a software development company. Our team, including me, operates from India. However my product owner is in USA. We are working on the feature development for this product that exists for ten years now. **Our team in India started six months ago, with no product nor domain knowledge on it.** [...]

A systematic review identifying and comparing estimation techniques could support practitioners facing problems like the one we showed in the example. Additionally, it is important to note that evidence of such techniques effectiveness needs to be contextualized, so practitioners can check if they fit in their working environment. In the example we have shown, only techniques to deal with situations of low knowledge about software domain matter.

The second most recurrent issue is when practitioners ask for recommendations of **Tools to support project management with specific features**. None selected SRs could cover the six issues under this classification. In previous sections, we already discussed approaches providing evidence when recommendations about tools are demanded.

The third most recurrent issue is when practitioners ask for **Strategies and metrics to measure team productivity**. SR13 (93) could cover one of the four issues under this classification. To illustrate, one issue grouped under this recurrent issue is:

> **How to measure team productivity?**
> The upper management at our company has laid out a goal for our software team to be 15% more productive over the next year. Measuring productivity in a software development environment is very subjective, but we are still required to come up with a set of metrics. **What sorts of data can we capture that would measure our team's productivity?**

A systematic review identifying software team productivity metrics could provide an interesting overview to practitioners facing that kind of problem. One good example in a different topic Saraiva et al.'s mapping study (90) that identified metrics to measure how software maintainability is affected by aspect-oriented programming. Traditional systematic reviews and meta-analyses can also provide rich evidence about the effectiveness and applicability of each metric.

Software Engineering Process Coverage (1 recurrent issue)

We identified just one recurrent issue among the 20 issues related to software engineering process. The issues coverage is presented in Table 15.

Table 15. Coverage of software engineering process recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Introducing and adapting a software development process in specific context	SoftwareProcessAdaptation. Chapter 8, Section 2.3	3	21%	14
Miscellaneous	-	1	16%	6

The only recurrent issue identified is practitioners asking for ways to support them **Introducing and adapting a software development process in specific context**, with 14 issues under this classification. Just three issues are covered. SR11 (38) SR16 (19), and SR24 (50) are the ones that presented evidence that could help to solve the three issues considered as covered. Following there is one example of issue:

> **Introducing Scrum in a distributed team**
> We would like to start using scrum [...] Until know we used a "home-grown" methodology, but we would like to switch to something more defined and mature. Scrum would be a great choice in my opinion and also the management supports us to go agile. **Where should we start this transition? Is there some guide or best practices for this transition?**

Systematic reviews identifying processes as well as best practices during introduction or adaptation of a process/method could provide useful evidence for issues likes the one we mentioned, specially when they are highly contextualized.

Software Engineering Models and Methods Coverage (15 recurrent issues)

This area is the one with more issues, 127 in total. We could identify 15 recurrent issues, all of them are related to agile software development. Table 16 shows those issues coverage.

The most recurrent issue is about **Applicability of agile in specific project context**. Only one out of the 19 issues under that classification was covered by SR11 (38). Some issues are highly tied to their project context, and that is why contextualized evidence is important to assess whether its applicable in real environments. To illustrate, following there is one of the issues under that classification, which asks for evidence about the applicability of agile methods in a context of firmware/embedded project:

> **How to adopt agile methodology for developing firmware/embedded-systems-software?**
> [...] how to apply agile methods in large complex embedded system software (100+ engineers). Firmware development has some unique characteristics that make it difficult to do agile (ie. Hardware is not available until late in the dev cycle; Once product is released, can't easily update firmware; etc...) The norm in this kind of development is thick documentation and grueling peer reviews. You can't get a simple code fix like renaming a variable without 2-3 signatures. (I exaggerate a little but this is typical. Additionally, a lot of people do take shortcuts and the Project Managers even approve them especially in the face of hard market deadlines.) **I would like to hear any tips or guidelines on how to adopt agile methodology for firmware development projects.**

Table 16. Coverage of software engineering models and methods recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Applicability of agile in specific project context	Agile Methods. Chapter 9, Section 4.4	1	5%	19
Mixing agile with traditional methods/practices	Agile Methods. Chapter 9, Section 4.4	0	0%	10
Benefits of agile methods/practices in general	Agile Methods. Chapter 9, Section 4.4	3	33%	9
Pair programming to transfer knowledge	Agile Methods. Chapter 9, Section 4.4	0	0%	9
Pair programming with distributed pairs	Agile Methods. Chapter 9, Section 4.4	1	12%	8
Impact of low detail level or absence of documentation in agile	Agile Methods. Chapter 9, Section 4.4	0	0%	6
Benefits of agile methods/practices from a specific perspective	Agile Methods. Chapter 9, Section 4.4	2	40%	5
Low customer collaboration in agile	Agile Methods. Chapter 9, Section 4.4	1	25%	4
Pair programming hindering concentration	Agile Methods. Chapter 9, Section 4.4	0	0%	4
Benefits of agile methods/practices in specific contexts	Agile Methods. Chapter 9, Section 4.4	0	0%	4
Mixing multiple agile methods/practices	Agile Methods. Chapter 9, Section 4.4	0	0%	4
Tools for agile methods/practices	Agile Methods. Chapter 9, Section 4.4	0	0%	4
Negative impact of agile in software design	Agile Methods. Chapter 9, Section 4.4	3	100%	3
Ad-hoc software development as agile	Agile Methods. Chapter 9, Section 4.4	0	0%	3
Pair programming as replacement to code reviews	Agile Methods. Chapter 9, Section 4.4	0	0%	2
Miscellaneous	-	5	15%	33

Studies aggregating and synthesizing evidence from cases studies, orthographies, and action researches could provide interesting information for practitioners who want to decide which agile methods/practices fit in their contexts.

The second most recurrent issue is practitioners asking for strategies to deal with **Mixing agile with traditional methods/practices**. None of the ten issues under this classification are covered by the selected SRs. To illustrate this situation, one of the issues under this classification follows:

> 💬 **How to synchronize an agile software team with a waterfall hardware team?**
> ▪▪▪ Our team is composed of both software and hardware engineers. The software team uses Scrum project management while the hardware team uses waterfall. The priority of our software requirements change quite frequently, so staying Agile makes sense for us. The priority of our hardware requirements are rather static and slow-moving, so again sticking with waterfall makes sense for us. **The tricky part is the integration of hardware and software. Are there any methodologies for deterministically synchronizing these two contrasting project management styles?**

In some cases there is an impression that once agile is adopted, every stakeholder and process operate through agile philosophy. However as we can see, many practitioners face situations where their team is agile, but not their company as a whole. Or even when the team is agile, but not their customer. Such situations provoke many disarrangements during software development life-cycle. In a survey in Microsoft, Begel *et al.* (16) identified that agile is not adopted simultaneously by all teams. They reported one situation engineers are more worried about how agile teams coordinate dependencies and deliverables with non-agile teams. However, more evidence is demanded in this topic, specially identifying strategies to deal with such kind of scenarios.

Three recurrent issues were identified around agile methods/practices benefits. The most recurrent is when practitioners ask for information about **Benefits of agile methods/practices *in general***, which can be illustrated by the following issue:

> 💬 **What makes Agile software development so appealing?**
> ▪▪▪ Agile software development is becoming a pretty fun buzzword these days. [...] Whether it is crystal, agile methods, dsdm, rup, xp, scrum, fdd, tdd, you name it. [...] **what are the biggest reasons for choosing to do Agile development**[...]

For those cases, SR11 (38) provides useful evidence since it focus on agile benefits in general, chiefly aggregating qualitative evidence.

Another recurrent issue is when practitioners ask for information about **Benefits of agile methods/practices from a specific perspective**. Following there is one issue to exemplify:

> **What are the monetary benefits of going agile?**
> ▪▪▪ Why go agile? This is the first question that comes to my mind when I think of going agile. What are the **possible financial benefits** one can achieve from going agile?[...]

This situation corroborates with the idea that empirical evidence should comprise not only data about the effectiveness of an intervention, but also useful information for target audience, in this case, cost-effectiveness (8). We also identified other issues that report the agile benefits information from other specific perspectives, beyond monetary, such as, benefits from the perspective of developers, managers, testers, customers, and others.

The third and last recurrent issue on agile benefits is when practitioners demand for information about **Benefits of agile methods/practices in specific contexts**, which again shows the importance of contextualized evidence.

Four recurrent issues about pair programming were identified. The first most recurrent issue in that matter is the one that practitioners ask about the applicability of **Pair programming to transfer knowledge** from more skilled developers to less skilled ones. None of the nine issues classified under this recurrent issue are covered by the selected SRs. SLR22 (43) evaluated the impact of pair programming in many dimensions, but only considered pairs with same experience level, failing to perceive practical importance of evaluating pair programming with pairs with different levels of experience and domain knowledge. To illustrate, one issue under this classification is:

> **Pair programming when driver and observer have different skill level and experience**
> ▪▪▪ I just wonder [if] the strategy still work in the case. For example if they have a very different programming skill level. if one never experience in the problem domain while another have. Is it still OK if they have low programming skill level?

The second most recurrent issue about pair programming is practitioners demanding support to have **Pair programming with distributed pairs**. Just one out of eight issues is covered. SLR22 (43) was the SR that provided evidence to cover the issue. To illustrate, an issue classified under this recurrent issue is:

> **Any suggestions for pair programming with external resource?**
> ▪▪▪ [...] I was considering hiring a developer [...] to assist me. Ideally, we would be a collaborative team [...] Has anyone attempted this? [...]

Systematic reviews identifying and comparing tools to support distributed pair programming could provide interesting information to practitioners.

Another recurrent issue we identified about pair programming is practitioners reporting problems with **Pair programming hindering concentration**. None of the four issues under this classification are covered. To illustrate, an issue classified under this recurrent issue is:

> **How can my team reconcile flow and pair-programming?**
> ▪▪▪ [...] Flow is a mental state attained by creativity workers (engineers, writers, programmers, etc.) which is often described as a state of immersion in which time seems to pass unknowingly and creative work flows from the mind [...] Pair programming, advocates a two person team which functions as an single organic, programming entity to accomplish a single goal. [...] are these ideas reconcilable?

A primary study has reported that concentration decreases in longer pair programming sessions due to exhaustiveness of pair dynamics (97). However, more evidence is needed about this topic to draw more accurate conclusions.

Another recurrent issue that deserves mention is practitioners affirming to experience **Ad-hoc software development as agile**. None of the three issues under this classification could be covered by the selected SRs. To illustrate, one issue classified under this recurrent issue is:

> **Is the agile approach too much of a convenient excuse for cowboys**
> I believe that an agile approach is best for projects where the requirements are fuzzy and a lot of interaction is required to help shape the end user's ideas. However... **In my professional work, I keep ending up at companies where an "agile" approach is used as an excuse as to why no effort was put into an up front design; when the requirements are well understood**.

Some empirical studies have observed situations like that (46; 22). However, SRs are demanded to investigate that kind of scenario, as well as to identify best practices to avoid it.

Another recurrent issue we identified shows practitioners asking about the **Impact of low detail level or absence of documentation in agile** and how to deal with that situation. None of the six issues under this classification are covered. To illustrate, there is the following issue:

> **Disillusioned with agile; how do you prepare for life after release 1.1?**
> My company is going full steam with the agile process, with multiple agile projects in work. [...] establish ideal documentation effort, the team was quickly disbanded [...] the next sprint leaves little documentation, little vision, and poor records of that design decisions were made [...]

This issue possibly challenges basic principles of agile software development, like *"working software over comprehensive documentation"* (agi). This does not mean we should go back to over-documentation times and high costs associated with it. However, it is noticeable an ideal balance between documentation and working software is, sometimes, far to be accomplished in practice. Systematic reviews identifying current strategies to establish ideal documentation, their pros and cons under specific contexts, and an evaluation of those strategies, could provide evidence to practitioners' facing that scenario.

Another recurrent issue challenging a basic principle of agile methods/practices emerges when practitioners report problems due to **Low customer collaboration**. The agile manifesto (agi) says that *"Customer collaboration over contract negotiation"* is one of the most important values. However, situations in practice may hinder agile adoption due to difficulties collaborating with customers. Some studies also observed such kind of situations (47), but there is a need of more evidence around this topic. Just one out of four issues under this classification are covered, this time by SR11 (38).

Software Engineering Professional Practice Coverage (7 recurrent issues)

This area is the third with more issues related to, with a total of 47. We could identify seven recurrent issues. Three of them present a coverage rate equal or above 50%, two present a coverage rate below 50%, and two are not covered at all, as can be seen in Table 17.

Table 17. Coverage of software engineering professional practice recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Strategies to deal with knowledge management in a team	Team and Group Communication. Chapter 11, Section 3.3	4	50%	8
Difficulties dealing with customer in specific contexts	Interacting with Stakeholders. Chapter 11, Section 2.4	1	20%	5
Improving communication in a distributed team	Team and Group Communication. Chapter 11, Section 3.3	4	100%	4
Difficulties dealing with team members in specific contexts	Interacting with Stakeholders. Chapter 11, Section 2.4	1	33%	3
Team rotation	Dynamics of Working in Teams/Groups. Chapter 11, Section 2.1	0	0%	4
Ideal workplace layout	Group Dynamics and Psychology. Chapter 11, Section 2	0	0%	3
Tools to support knowledge management with specific features	Team and Group Communication. Chapter 11, Section 3.3	1	50%	2
Miscellaneous	-	1	5%	17

The most recurrent issue is when practitioners ask for **Strategies to deal with knowledge management in a team**. Four out of eight issues under this classification are covered. SR16 (19) is responsible for offering knowledge that helps to solve those issues. This happens because SR16 investigates concepts, findings, and methods to manage knowledge in SE. To illustrate this recurrent issue, following there is an issue:

> **How to motivate team for knowledge sharing sessions**
> I work in a team with wide range of expertise and experience. I have been trying to introduce weekly knowledge sharing sessions. Sessions of 30-60 min length where everybody gets a chance to present something and talk about it [...] However, the team is not motivated towards this, either the attendance is too low or none. How to get a team work towards such an idea?

The second most recurrent issue is practitioners reporting to face **Difficulties dealing with customer in specific contexts**. Only one out of five issues under this classification is covered. To illustrate that situation there is the following issue:

> **Does anyone have experience with a difficult customer?**
> We have a reoccurring conflict with one of our larger and strategically important customers [...] I'm looking for hints on how to control the development-process (and our own economy) and still provide the customer with a product that gives the company most value-for-money [...]

To deal with stakeholders, it is demanded a particular set of skills that sometimes are even more important than traditional technical skills in SE world (32). Yaman *et al.* (105) conducted an interesting systematic review about benefits, challenges, methods, and tools to support customer involvement in a continuous deployment environment. A similar approach could be adopted and an SR with recommendations and best practices about customer relationship could provide useful information for practitioners facing that kind of problem.

Improving communication in a distributed team also seems to concern practitioners since it is the third most recurrent issue under Software Engineering Professional Practice area. This recurrent issue is particularly well-supported since all four issues under this classification are covered by SR24 (50).

Another recurrent issue is practitioners reporting **Difficulties dealing with team members in specific contexts**. As with issues related to difficulties dealing with customers, it is important to explore the human aspects of that kind of environment and strategies to deal with it since team conflicts might hinder team performance (37;).

Another recurrent issue is when practitioners ask for information about **Team rotation**. None of the four issues under this classification are covered. A primary study was published recently building a team rotation theory grounded in a qualitative case study (89). However, more evidence is needed to support the theory, as well as to aggregate enough empirical evidence to conduct a systematic review on this topic.

Finally, some practitioners ask for information about **Ideal workplace layout** for a software development team. None of the three issues classified under this recurrent issue are covered by the selected SRs. There are some primary studies under this topic, for example, Sykes' study (92) proposing workplace layout strategies to reduce level of interruption in a software development team. However, more evidence is demanded in this topic, specially addressing strategies to define workplace layout aiming at different types of goals beyond reducing interruptions.

Software Engineering Economics Practice Coverage (3 recurrent issues)

All three recurrent issues we identified under this SE area are related to outsourcing/offshore software development. Table 18 shows how the selected systematic reviews covered those issues.

Table 18. Coverage of software engineering economics recurrent issues.

2*RECURRENT ISSUE	2*SWEBOK SECTION	COVERED		TOTAL
		#	%	#
Strategies to introduce outsourcing/offshoring in specific contexts	Offshoring and Outsourcing. Chapter 12, Section 5.4	1	20%	5
Characteristcs of a good outsourcer/offshorer	Offshoring and Outsourcing. Chapter 12, Section 5.4	1	33%	3
Improving communication with the outsourcer/offshorer	Offshoring and Outsourcing. Chapter 12, Section 5.4	1	50%	2
Miscellaneous	-	1	100%	1

The most recurrent issue is when practitioners ask for **Strategies to introduce outsourcing/offshoring in specific contexts**. One of the five issues under this classification is covered. For instance, the following issue:

> **How to outsource the UI of a dynamic Web application?**
> [...] The project is nearly completed, but I want to enhance the look and feel. This will include better graphics and some extra behaviour. I want to outsource this task [...] I don't have experience with outsourcing and don't know how to incorporate an outsider to the project [...]

SR7 (61) and SR8 (60) provide useful evidence and could cover one issue. However, more evidence is demanded for specific contexts. For instance, the one in the example demands information about the possibility to outsource user interface web development. Strategies and recommendations may be different if the intention is to outsource software testing, or other parts/components of a software system.

The second most recurrent issue is when practitioners ask demand **Characteristics of a good outsourcer/offshorer**. Again SR7 (61) and SR8 (60) provided useful evidence and could cover one issue. However, the same problem concerning context limited the level of coverage of that kind of issues. To illustrate this recurrent issue, following there is an issue:

> **What to look for in an outsourced partner**
> [...] We might need some more development help and are looking at an Indian company which comes recommended by someone we know (although they are not very technical). I'll be having an informal chat with them, and thought I'd see if people here had some wisdom regarding **what to look for** and good questions to ask. [...]

The third recurrent issue is the one where practitioners ask for guidance for **Improving communication with the outsourcer/offshorer**. SR7 (61) and SR8 (60) provided useful evidence and could cover one issue. However, the same problem concerning context limited the level of coverage of that kind of issues. To illustrate, there is the following issue classified under this recurrent issue:

> **Communicating requirements to offshore teams**
> Just to give a context, there is an offshore team in India for a client in San Francisco. The offshore team is about 9 developers and 4 QA, with one project manager. I am doing onsite coordination for this team from the client location [...] They obviously fail to deliver sprint after sprint. What would you do to get these things right? How much of adequate is adequate clarity in requirement? [...]

DISCUSSION

In this section, we summarize our findings, discuss how systematic reviews can be further improved with the use of Stack Exchange, and debate the need for tertiary studies.

Revisiting Findings

There are healthy Stack Exchange communities beyond Stack Overflow, focusing on non-technical issues and ready to be explored. This enhances the possibilities to discover which are the issues practitioners are facing and to explore them through research aiming to provide empirical evidence connected to software engineering practice demands. Those Stack Exchange communities approach a wide variety of software engineering areas, even though not all of them yet. For researchers who want to explore practitioners' issues in software engineering through Stack Exchange, we recommend the following communities: *Programmers* (PROG) (PRO), *Project Management* (PM) (PMS), and *Software Quality Assurance & Testing* (SQA) (SQA). The *Reverse Engineering* (RE) (RES) community needs further investigation to understand whether it is a good source or not. On the other hand, we do not recommend *Software Recommendations* (SREC) (SRS) community, since it is focused on tools in general, not tools to support software engineering practice.

Many problems were observed with systematic reviews search strings. Some SRs present malformed search strings, or even do not report them at all. This hinders replicability, one important characteristics of systematic methods proposed by EBSE community. Part of the low quantity of practitioners' issues is related to poorly defined systematic reviews' search strings. Seven out of 24 SRs search strings did not return any issue. This occurred due to search strings with few key terms, key terms with no synonyms, or key terms with composed synonyms only. Additionally, one main reason for the excess of false positives is also poorly defined search strings. For instance, search strings using rather common

terms. Despite systematic reviews' search strings were not originally defined to search for practitioners' issues in Q&A platforms like Stack Exchange, such poorly defined search strings can lead to problems even using them with their original purpose, which is to find primary studies in search engines. On Da Silva et al.'s tertiary study (30), which is based on two other tertiary studies ((64; 65)), each of the 120 systematic reviews was evaluated on their quality. The evaluation was based on a questionnaire with four questions, and one of those questions was "Are the review's inclusion and exclusion criteria described and appropriate?". We believe a similar question should be added to their quality questionnaire: *"If the review uses automatic search, are the review's search string described and appropriate?"*. This would increase quality of SRs, specially their potential to be fully replicated.

Only 1.75% of most relevant practitioners' issues are related to the selected systematic reviews. This shows SRs are still far from touch the whole spectrum of topics discussed by practitioners. It is important to reinforce that related issues are not the same of covered issues. One related issue needs to belong to the same SE area of the SR, whereas a covered issue needs one SR with at least one finding that helps to solve the issue.

Only 14.1% of the 424 practitioners' issues related to the selected SRs are covered. This suggests the selected SRs are also facing a difficult time to provide evidence covering specific issues faced by practitioners. We presented some guidelines to support researchers wanting to conduct SRs more connected with practitioners' issues.

Practitioners' issues related to nine out of the 15 SWEBOK SE areas are covered in some extent by the selected SRs. None SE areas presented a coverage rate above 50%. The two SE areas with higher coverage rate are Software Engineering Economics and Software Testing with 36.3% and 26.6% respectively. None issues related to Software Design are covered by the selected SRs. Additionally, the selected SRs' search strings could not find any issue related to the following SE areas: Software Configuration Management, Software Quality, Computing Foundations, Mathematical Foundations, and Engineering Foundations.

We identified 45 recurrent issues distributed in many SE areas. The three most recurrent issues are: Applicability of agile in specific project context (19 issues); Introducing and adapting a software development process in specific context (14 issues); and Strategies to cost and effort estimation in specific contexts (13 issues).

There are practitioners explicitly asking for scientific empirical evidence in Stack Exchange communities. This shows there is interest in empirical evidence from practitioners side, as also observed in (26).

Many practitioners' issues ask for recommendations of tools with specific features. Systematic reviews identifying tools, comparing their features, and aggregating evidence about their effectiveness could help to cover that gap. One example of such study is the one conducted by Marshall *et al.* (73) that identified analyzed and compared tools to support SRs in SE based on their features. Another approach with direct implications for tool builders is to identify features demanded for SE tools based on issues posted in Stack Exchange communities. An example of this approach can be found in Pinto and Kamei's study (83), which investigated practitioners' issues in Stack Exchange communities to identify the most demanded features for refactoring tools.

Practitioners demand contextualized information. This can be observed by looking at recurrent issues like: Benefits of agile methods/practices from a specific perspective; Applicability of agile in specific project context; and Effort estimation in agile in specific project context. This supports many claims about importance of rich and contextualized evidence (24; 39; 8).

Practitioners demand target oriented information. For instance, we identified the following recurrent issue: Benefits of agile methods/practices from a specific perspective. This situation corroborates the idea that empirical evidence should comprise not only data about the effectiveness of an intervention but also useful information for the target audience (8). For instance, cost-effectiveness. We also identified other issues that report the need of information from other specific perspectives beyond monetary, such as, from the perspective of developers, managers, testers, customers, and others.

We identified 15 recurrent issues related to agile software development comprising 127 practitioners' issues. This is almost one-third of the issues we have found in this study, which means agile is still an important topic in practice. Two recurrent issues reveal practitioners are facing problems that challenge some of the basic agile principles, which can be observed when practitioners report *Low customer collaboration* or when they acknowledge not desirable *Impacts of low detail level or absence of*

documentation in agile. Another revealing recurrent issue is practitioners affirming to have experienced *Ad-hoc software development as agile*, corroborating some evidences and claims that in some situations agile is used as an excuse for absence of software process (46; 22). Additionally, we identified practitioners are facing problems *Mixing agile with traditional methods/practices*, and demanding evidence about applicability of *Pair programming to transfer knowledge*, among many other recurrent issues.

Guidelines for Conducting Systematic Reviews Considering Practitioners' Issues

After applying the proposed coverage analysis, we observed that some studies have room for improvement and if researchers adopt few guidelines, those SRs might address a wide range of practitioners' issues. We also would like to reinforce that not all SRs need to approach practical issues. We recognize there are plenty of systematic reviews exploring abstract and methodological aspects that do not necessarily interest practitioners and are still important to the development of EBSE. For this kind of SRs, there are no advantages of pursuing our guidelines. Following are our recommendations:

- **Test systematic review's search string on Stack Exchange at early stages of study planning:** We believe one who wants to conduct a systematic review with useful findings to practitioners can adopt a similar research strategy we presented here: assess on early stages – protocol definition – how its research relates to the practitioners' issues on Stack Exchange. The insights found at Stack Exchange can be included in systematic review's scope of investigation during its planning phase.
- **Test systematic review's search string on Stack Exchange when planning to update an already existent systematic review:** Querying Stack Exchange may reveal candidate updates of the original research questions and opportunities to cover issues demanded in practice.
- **Mitigate problems that prevent SR to well-cover practitioners' issues:** During this research we identified some key problems that hinder practitioners' issues coverage as well as their causes. They are either related to poorly defined search strings or to a research scope that does not consider practitioners' issues. In Table 19 we defined actions as suggestions to mitigate those problems during an SR planning phase. The table should be read as following: To avoid <problem> caused by <problem cause>, I should <mitigation action>. For example, ***To avoid*** *excess of false positives* ***caused by*** *systematic reviews using rather common terms in their search strings,* ***I should*** *avoid general key terms on the search string unless they are connected by an AND operator with specific key terms.*
- **Select high-quality primary studies:** The proposed coverage analysis is based on systematic reviews. However, the quality of these reviews is subject to the quality of the primary studies selected. If the primary studies found are all of rather poor quality, then it will be difficult to place great confidence in outcomes. Therefore, researchers should place additional care when deriving inclusion/exclusion criterion.

Table 19. Mitigating actions to avoid problems that prevent coverage of practitioners' issues.

	PROBLEM	PROBLEM CAUSE	MITIGATING ACTION
3*	**No practitioners' issues returned by the search**	Search string with too much key terms	Reduce the amount of key terms, since they are connected by the AND operator, that is restrictive
		Search string with no synonyms	Use synonyms for each key term to increase the possibility to find related issues
		Search string with key term with composed synonyms only	Mix composed synonyms with not composed ones, since the former is more unlikely to happen in the exact order on the issues
1*	**Excess of False positives**	Systematic reviews using rather common terms in their search strings	Avoid general key terms on search strings unless they are connected by an AND operator with specific key terms.
2*	**Not Covered Issues**	Issues reporting scenarios systematic reviews do not fully cover	If possible, include the scenario on the systematic review's scope, since it can enriches the study
		Issues using approaches available/popular after the systematic review was conducted	This should not occur when one assess Stack Exchange questions during early stages of systematic review planning. But if it occurs, this suggests maybe the area of research is outdated

The Need of Tertiary Studies

This research study is primarily based on Da Silva et al.'s tertiary study (30). Da Silva et al.'s (30) is an extension of two other tertiary studies. The original tertiary study found 20 unique studies reporting SRs

published between 1st January 2004 and 30th June 2008 (64). The first extension found 33 additional unique studies published between 1st January 2004 and 30th June 2008 (65). The study of Da Silva *et al.* (30) added 67 new systematic reviews, comprehending a total of 120 systematic reviews. These systematic reviews range from 1st January 2004 to 31th December 2009. However, with the steady stream of new systematic reviews conducted on a regular basis, it is expected that this tertiary study does not cover the new advances made in recent years. Therefore, we believe there is an urgent need for, at least, an update on this tertiary study since, to the best of our knowledge, Da Silva's study is the most comprehensive and up-to-date tertiary study mapping systematic reviews in software engineering as a whole.

Some tertiary studies were published after 2011, but none focus on SRs in software engineering as a whole. They either investigate methodological aspects of SRs or explore a narrower specific SE topic. Examples of the former are: a tertiary study conducted by Cruzes and Dybå (28) that identified which syntheses methods have been used in SRs; a tertiary study by Da Silva *et al.* (31) that critically appraised SRs from their research questions perspective; a tertiary study of Ali and Petersen (9) that investigated strategies used to selected primary studies in SRs; and a tertiary study conducted by Zhou et al. (106) that investigated how SRs assess quality of the primary studies they include. Examples of the latter - tertiary studies that explore a narrower specific topic of SE - are: Marques et al.'s tertiary study (72) on distributed software development; Santos et al.'s tertiary study (29) also on distributed software development; Verner et al.'s tertiary study (100) on global software development; Bano et al.'s tertiary study (11) on software requirements; Goulão et al.'s tertiary study (42) on model-driven engineering; Garousi et al.'s tertiary study (41) on software testing; and the tertiary study of Hoda et al. (48) on agile software development.

The Lack of Studies Targeting Other Stack Exchange Communities

Our study is based on five Stack Exchange communities. However, as we shall, even though there is a plethora of studies that rely on Stack Overflow, the most popular and largest Stack Exchange community, there are few studies built upon any of the remaining Stack Exchange communities. This lack of studies is not related to a lack of opportunities. On the contrary, we believe there are ample benefits for exploring these different communities, because:

1. They host a diverse set of practitioners (*e.g.,* while Unix and Linux users can be found at the Unix community[4]), database administrators can be found at the DBA community[5]).

2. They employ the same gamification mechanism Stack Overflow uses for guaranteeing quality of both questions and answers. Still, Area51 staging zone[6] monitors how well these communities are (regarding activity, followers, and percentage of answered questions).

It is clear that Stack Overflow is *the facto* community to go when exploring issues intrinsic related to coding activities. However, since software development is much more than just coding, when non-coding or community specific issues are relevant, these different Stack Exchange communities can play an important role.

LIMITATIONS

As any empirical study, this one has it is particular limitations. Here we acknowledge the ones we identified.

- The selected SRs came from a tertiary study published in 2011. This is the most up-to-date tertiary study we could find. Thus, issues coverage might be pessimistic since newer SRs may be able to cover more issues.

- The scope is limited to how systematic reviews cover practitioners' issues since they can provide more consolidated and mature evidence to practice than primary isolated studies (67). Though, not covered issues can occur due to either absence of SRs or absence of primary studies. The former can be mitigated conducting a systematic review providing evidence that helps to solve the issue. The

[4] http://unix.stackexchange.com/
[5] http://dba.stackexchange.com/
[6] http://area51.stackexchange.com/

latter demands an effort of the research community as whole since if there are no primary studies on a specific issue, there is no chance to exist SRs covering that issue. To determine why there are few or no SRs covering specific practitioners' issues is out of the scope of this study, as well as to investigate the coverage of practitioners' issues by primary studies. Thus, it is possible there are primary studies that could provide evidence to cover some practitioners' issues we identified, but this is beyond of our scope of investigation.

- Practitioners' issues returned by Stack Exchange searches are sensible to SRs search strings. Poor search strings might lead to poor results. On the other side, identification of SRs with poor search strings is a finding itself. Researchers might test their search strings in Stack Exchange to understand how they are connected with practice.

- The method we proposed cannot be fully automated. The only phase that was automated was the selection of related questions. A search engine based on Apache Lucene was built to automate this process. The remaining phases were conducted manually. To mitigate classification bias, we conducted it in pairs, with conflict resolution meetings.

- One might argue that instead of using five different Stack Exchange communities, we should favor Stack Overflow, the most popular Stack Exchange community. However, Stack Overflow is focused on programming and, therefore, it demands questions to have a specific, concrete technical answer (*e.g.,* the best Stack Overflow questions present a code snippet (76)). Such questions rarely fit in topics investigated in the SE research literature.

- Our results cannot be extended to other Q&A communities (*e.g.,* Yahoo Answers, Quora, Experts Exchange), neither all Stack Exchange communities. To mitigate the risk of not taking into account relevant Stack Exchange communities, we classified all communities according to all software engineering areas listed by SWEBOK (21) and selected only those related to at least one SE area (Table 2).

- Systematic reviews may be useful for practitioners even if they do not provide guidelines. For instance, practitioners might get acquainted with an emerging topic or new results. However, we argue that the lack of guidelines represent a serious limitation on these studies, since their absence might leave the reader with no clear answer (*e.g.,* should I use pair programming in my context?). That is why we excluded SRs that do not present guidelines to practitioners.

- Finally, we selected Stack Exchange's issues based on their score. This approach might favor old questions since it takes a time to a question have a high score. Although there are several other ways to select issues in Stack Exchange communities, score is a common property used in software engineering studies for filtering out relevant issues, avoiding low-quality ones (96).

RELATED WORK

In this section, we describe the studies overlapping with the scope of our work.

Empirical Studies on Stack Exchange communities

Software engineering community has recognized the importance of Stack Exchange, with significant efforts placed on understanding practitioners' needs and challenges they face. Most studies about Stack Exchange focus on understanding the dynamics of one particular community: Stack Overflow (71; 95; 12)). Notable exceptions are the work of Posnett *et al.* (84), which focus on serverfault[7], the second largest Stack Exchange community, and the study of Vasilescu *et al.* (99), which used CrossValidated, a community for statisticians, data analysts, data miners, and data visualization experts[8]. In contrast, there are some studies focusing on understanding what problems practitioners face, for instance, software methodologies (81), techniques (82), and tools (85) in these communities.

Pinto *et al.* (82) selected the most popular questions about concurrency created on Stack Overflow. They observed the majority of questions are asking for guidance on theoretical concepts (*e.g.,* what is the difference between a thread and a process?). Despite concurrency textbooks have discussed these

[7] http://serverfault.com/
[8] http://stats.stackexchange.com/

problems in length, the authors suggested more effort should be placed on improving the documentation of concurrency APIs and frameworks. In another study, Pinto *et al.* (81) manually investigated 300+ questions about software energy consumption. They found that even though practitioners are interested in this subject, they lack property tool support. Reboucas *et al.* (85) studied the benefits and drawbacks of being an early adopter of Swift, a programming language that is bound to be widely adopted. They analyzed around 60,000 questions about Swift using a topic modeling technique and observed that Swift developers have problems with basic language constructs and the toolset. However, none of these studies are interested in understanding if systematic reviews can be used to answer questions raised by practitioners. The closest work to this research is Garousi et al.'s (41), but it proposes Stack Exchange as evidence source, as gray literature for SRs, not as a source to analyze how SRs cover practitioners' issues. Our work is unique in the sense we take advantage of the Q&A communities rich databases to empower researchers wanting to assess how their SRs are aligned with practitioners' issues.

Relevance of Software Engineering Research to Practitioners:

Since software engineering is an applied discipline, there are some efforts reported aiming to connect research to practitioners. In 2013, Begel *et al.* (17) released a technical report presenting 145 questions that 203 Microsoft software engineers would like to ask data scientists to investigate about software engineering. Additionally, they asked for a different set of Microsoft software engineers to rank the importance of each of the 145 questions. Another study with Microsoft software engineers was conducted in 2015 by Lo *et al.* (68). In this study, the goal was to understand how practitioners perceive software engineering research relevance. They summarized 571 papers from five years of ICSE, ESEC/FSE and FSE conferences and asked practitioners to rate them according to their relevance. They received ratings from 512 practitioners, and their results suggest that practitioners are positive towards studies done by the software engineering research community since 71% of all ratings were essential or worthwhile. We believe in some ways our research is complementary with those two we mentioned. This research is focused only on systematic reviews to bring insights to EBSE community, while the others are focused on software engineer research as a whole. Their study is limited to Microsoft engineers, while this research is conducted based on practitioners data from potentially any company and country in the world since our data source is the Stack Exchange communities. On the other hand, they could collect demographic data about practitioners, and we could not since Stack Exchange does not provide a native chat or private messaging system allowing to contact questions' authors.

CONCLUDING REMARKS AND FUTURE WORK

In this paper, we conducted a study to measure how well systematic reviews cover practitioners' issues. To do so, we selected a set of SRs identified in a tertiary study (30). For each study, we extracted its search string, and use it to select questions in Stack Exchange communities, a popular Q&A platform. We analyzed more than 1,800 practitioners' issues and 424 issues were considered related to the selected SRs. From that set, 60 (14.1%) were considered covered by the selected SRs' findings. Among the 424 issues, we could identify 45 recurrent issues distributed in many SE areas.

There are practitioners explicitly asking for scientific empirical evidence in Stack Exchange communities, which shows interest in empirical evidence from the practitioners' side. Many practitioners' issues ask for recommendations of tools with specific features. We provided some guidance for researchers who want to conduct SRs aiming to fill that gap. We also observed practitioners demanding contextualized information, which shows the importance to produce and report highly contextualized evidence. Practitioners also demand target oriented information. Thus, just measuring the effectiveness of an intervention is not enough to satisfy specific audiences. Evidence about cost-effectiveness and the impact of an intervention on aspects that concern stakeholders with specific perspective such as developers, managers, and customers are demanded. Agile was the topic with more practitioners' issues related to. We could identify 15 recurrent issues related to agile software development, comprising 127 practitioners' issues.

As any empirical study, this has limitations. Some of the limitations include: (1) the selection of SRs based on a tertiary study published in 2011, (2) a qualitative-based method, which can be error-prone, and (3) the use of a handful number of Q&A communities. These limitations were properly discussed and mitigate throughout this study.

As future work, we plan to leverage machine learning techniques for automatically extracting SRs' findings. Ultimately, this approach can be implemented as a third-party service, so that researchers can

upload their SRs, and the service would automatically query and return the issues found in selected Stack Exchange communities. This might reduce the burden researchers have when evaluating their search strings. Another opportunity for future work is to replicate our approach in software engineering sub-fields. This is further motivated by the recent introduction of niche-specific tertiary studies. We leave this for future work.

REFERENCES

[fit] Fitnesse - the fully integrated standalone wiki and acceptance testing framework. http://www.fitnesse.org. Acessed in: Nov. 4, 2016.

[agi] Manifesto for agile software development. http://agilemanifesto.org. Acessed in: Feb. 1, 2017.

[PRO] Stackexchange programmers community. http://programmers.stackexchange.com. Acessed in: Nov. 4, 2016.

[PMS] Stackexchange project management community. http://pm.stackexchange.com. Acessed in: Nov. 4, 2016.

[RES] Stackexchange reverse engineering community. http://reverseengineering.stackexchange.com. Acessed in: Nov. 4, 2016.

[SQA] Stackexchange software quality & testing community. http://sqa.stackexchange.com. Acessed in: Nov. 4, 2016.

[SRS] Stackexchange software recommendations community. http://softwarerecs.stackexchange.com. Acessed in: Nov. 4, 2016.

[8] Ali, N. b. (2016). Is effectiveness sufficient to choose an intervention?: Considering resource use in empirical software engineering. In *10th ESEM*, pages 54:1–54:6.

[9] Ali, N. B. and Petersen, K. (2014). Evaluating strategies for study selection in systematic literature studies. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 45:1–45:4, New York, NY, USA. ACM.

[10] Anderson, N. B. (2006). Evidence-based practice in psychology. *American Psychologist*, 61(4):271–285.

[11] Bano, M., Zowghi, D., and Ikram, N. (2014). Systematic reviews in requirements engineering: A tertiary study. In *2014 IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 9–16.

[12] Barua, A., Thomas, S. W., and Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654.

[13] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L. (1998). Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 368–.

[14] Bazelli, B., Hindle, A., and Stroulia, E. (2013). On the personality traits of stackoverflow users. In *2013 IEEE International Conference on Software Maintenance*, pages 460–463.

[15] Beecham, S., O'Leary, P., Baker, S., Richardson, I., and Noll, J. (2014). Making software engineering research relevant. *Computer*, 47(4):80–83.

[16] Begel, A. and Nagappan, N. (2007). Usage and perceptions of agile software development in an industrial context: An exploratory study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 255–264.

[17] Begel, A. and Zimmermann, T. (2014). Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 12–23, New York, NY, USA. ACM.

[18] Bellman, L., Webster, J., and Jeanes, A. (2011). Knowledge transfer and the integration of research, policy and practice for patient benefit. *Journal of Research in Nursing*, 16(3):254–270.

[19] Bjørnson, F. O. and Dingsøyr, T. (2008). Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50(11):1055 – 1068.

[20] Booth, W. C., Colomb, G. G., and Williams, J. M. (2003). *The craft of research*. University of Chicago press.

[21] Bourque, P. and Fairley, R. E., editors (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition.

[22] Cao, L. and Ramesh, B. (2007). Agile software development: Ad hoc practices or sound principles? *IT Professional*, 9(2):41–47.

[23] Cartaxo, B. (2016). Integrating evidence from systematic reviews with software engineering practice through evidence briefings. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE '16, pages 6:1–6:4, New York, NY, USA. ACM.

[24] Cartaxo, B., Almeida, A., Barreiros, E., Saraiva, J., Ferreira, W., and Soares, S. (2015). Mechanisms to characterize context of empirical studies in software engineering. In *Experimental Software Engineering Latin American Workshop (ESELAW 2015)*, pages 1–14.

[25] Cartaxo, B., Pinto, G., Ribeiro, D., Kamei, F., Santos, R. E. S., da Silva, F. Q. B., and Soares, S. (2017). Using q&a websites as a method for assessing systematic reviews. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 238–242.

[26] Cartaxo, B., Pinto, G., Vieira, E., and Soares, S. (2016). Evidence briefings: Towards a medium to transfer knowledge from systematic reviews to practitioners. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, pages 57:1–57:10, New York, NY, USA. ACM.

[27] Chen, F. and Kim, S. (2015). Crowd debugging. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 320–332.

[28] Cruzes, D. S. and Dybå, T. (2011). Research synthesis in software engineering: A tertiary study. *Information and Software Technology*, 53(5):440 – 455. Special Section on Best Papers from {XP2010}.

[29] d. Santos, A. C. C., d. F. Junior, I. H., d. Moura, H. P., and Marczak, S. (2012). A systematic tertiary study of communication in distributed software development projects. In *2012 IEEE Seventh International Conference on Global Software Engineering*, pages 182–182.

[30] da Silva, F. Q., Santos, A. L., Soares, S., França, A. C. C., Monteiro, C. V., and Maciel, F. F. (2011). Six years of systematic literature reviews in software engineering: An updated tertiary study. *Information and Software Technology*, 53(9):899 – 913. Studying work practices in Global Software Engineering.

[31] da Silva, F. Q. B., Santos, A. L. M., Soares, S. C. B., França, A. C. C., and Monteiro, C. V. F. (2010). A critical appraisal of systematic reviews in software engineering from the perspective of the research questions asked in the reviews. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 33:1–33:4, New York, NY, USA. ACM.

[32] Damian, D. and Borici, A. (2012). Teamwork, coordination and customer relationship management skills: As important as technical skills in preparing our se graduates. In *2012 First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex)*, pages 37–40.

[33] Davies, P. (1999). What is evidence-based education? *British journal of educational studies*, 47(2):108–121.

[34] Davis, A., Dieste, O., Hickey, A., Juristo, N., and Moreno, A. M. (2006). Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 176–185, Washington, DC, USA. IEEE Computer Society.

[35] Dias-Neto, A. C. and Travassos, G. H. (2009). Model-based testing approaches selection for software projects. *Information and Software Technology*, 51(11):1487 – 1504. Third {IEEE} International Workshop on Automation of Software Test (AST 2008)Eighth International Conference on Quality Software (QSIC 2008).

[36] DiCenso, A., Cullum, N., and Ciliska, D. (1998). Implementing evidence-based nursing: some misconceptions. *Evidence Based Nursing*, 1(2):38–39.

[37] Dubinsky, Y., Ravid, S., Rafaeli, A., and Bar-Nahor, R. (2011). Governance mechanisms in global development environments. In *2011 IEEE Sixth International Conference on Global Software Engineering*, pages 6–14.

[38] Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9–10):833 – 859.

[39] Dybå, T., Sjøberg, D. I., and Cruzes, D. S. (2012). What works for whom, where, when, and why?: On the role of context in empirical software engineering. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 19–28, New York,

NY, USA. ACM.

[40] Farrington, D. P., MacKenzie, D. L., Sherman, L. W., Welsh, B. C., et al. (2003). *Evidence-based crime prevention*. Routledge.

[41] Garousi, V., Felderer, M., and Mäntylä, M. V. (2016). The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature. EASE '16, pages 26:1–26:6, New York, NY, USA. ACM.

[42] Goulão, M., Amaral, V., and Mernik, M. (2016). Quality in model-driven engineering: a tertiary study. *Software Quality Journal*, 24(3):601–633.

[43] Hannay, J. E., Dybå, T., Arisholm, E., and Sjøberg, D. I. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110 – 1122. Special Section: Software Engineering for Secure SystemsSoftware Engineering for Secure Systems.

[44] Hassler, E., Carver, J. C., Kraft, N. A., and Hale, D. (2014). Outcomes of a community workshop to identify and rank barriers to the systematic literature review process. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 31:1–31:10, New York, NY, USA. ACM.

[45] Haugset, B. and Hanssen, G. K. (2008). Automated acceptance testing: A literature review and an industrial case study. In *Agile 2008 Conference*, pages 27–38.

[46] Hilkka, M.-R., Tuure, T., and Rossi, M. (2005). Is extreme programming just old wine in new bottles: A comparison of two cases. *Journal of Database Management*, 16(4):41.

[47] Hoda, R., Noble, J., and Marshall, S. (2011). The impact of inadequate customer collaboration on self-organizing agile teams. *Information and Software Technology*, 53(5):521 – 534. Special Section on Best Papers from {XP2010}.

[48] Hoda, R., Salleh, N., Grundy, J., and Tee, H. M. (2017). Systematic literature reviews in agile software development: A tertiary study. *Information and Software Technology*, pages –.

[49] Hordijk, W., Ponisio, M. L., and Wieringa, R. (2009). Harmfulness of code duplication: A structured review of the evidence. In *Proceedings of the 13th International Conference on Evaluation and Assessment in Software Engineering*, EASE'09, pages 88–97, Swinton, UK, UK. British Computer Society.

[50] Hossain, E., Babar, M. A., and y. Paik, H. (2009). Using scrum in global software development: A systematic literature review. In *2009 Fourth IEEE International Conference on Global Software Engineering*, pages 175–184.

[51] Hunold, S., Krellner, B., Rauber, T., Reichel, T., and Rünger, G. (2009). Pattern-based refactoring of legacy software systems. In *International Conference on Enterprise Information Systems*, pages 78–89. Springer.

[52] Insfran, E. and Fernandez, A. (2008). *A Systematic Review of Usability Evaluation in Web Development*, pages 81–91. Springer Berlin Heidelberg, Berlin, Heidelberg.

[53] Ivarsson, M. and Gorschek, T. (2009). Technology transfer decision support in requirements engineering research: a systematic review of rej. *Requirements Engineering*, 14(3):155–175.

[54] Jedlitschka, A., Juristo, N., and Rombach, D. (2014). Reporting experiments to satisfy professionals' information needs. *Empirical Softw. Engg.*, 19(6):1921–1955.

[55] Jiménez, M., Piattini, M., and Vizcaíno, A. (2009). Challenges and improvements in distributed software development: A systematic review. *Adv. Soft. Eng.*, 2009:3:1–3:16.

[56] Jorgensen, M. (2005). Evidence-based guidelines for assessment of software development cost uncertainty. *IEEE Transactions on Software Engineering*, 31(11):942–954.

[57] Jørgensen, M. (2007). Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting*, 23(3):449 – 462.

[58] Kalinowski, M., Travassos, G. H., and Card, D. N. (2008). Towards a defect prevention based process improvement approach. In *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pages 199–206.

[59] Kavaler, D., Posnett, D., Gibler, C., Chen, H., Devanbu, P., and Filkov, V. (2013). Using and asking: Apis used in the android market and asked about in stackoverflow. In *Proceedings of the 5th International Conference on Social Informatics - Volume 8238*, SocInfo 2013, pages 405–418, New York, NY, USA. Springer-Verlag New York, Inc.

[60] Khan, S., Niazi, M., and Ahmad, R. (2009a). Critical success factors for offshore software development outsourcing vendors: A systematic literature review. In *ICGSE*.

[61] Khan, S. U., Niazi, M., and Ahmad, R. (2009b). Critical barriers for offshore software development outsourcing vendors: A systematic literature review. In *2009 16th Asia-Pacific Software Engineering Conference*, pages 79–86.

[62] Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, EBSE Technical Report EBSE-2007-01.

[63] Kitchenham, B., Mendes, E., and Travassos, G. H. (2006). A systematic review of cross- vs. within-company cost estimation studies. In *Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering*, EASE'06, pages 81–90, Swinton, UK, UK. British Computer Society.

[64] Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering - a systematic literature review. *Inf. Softw. Technol.*, 51(1):7–15.

[65] Kitchenham, B., Pretorius, R., Budgen, D., Pearl Brereton, O., Turner, M., Niazi, M., and Linkman, S. (2010). Systematic literature reviews in software engineering - a tertiary study. *Inf. Softw. Technol.*, 52(8):792–805.

[66] Kitchenham, B. A., Dyba, T., and Jorgensen, M. (2004). Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 273–281, Washington, DC, USA. IEEE Computer Society.

[67] Lavis, J. N., Robertson, D., Woodside, J. M., McLeod, C. B., and Abelson, J. (2003). How can research organizations more effectively transfer research knowledge to decision makers? *Milbank quarterly*, 81(2):221–248.

[68] Lo, D., Nagappan, N., and Zimmermann, T. (2015). How practitioners perceive the relevance of software engineering research. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 415–425, New York, NY, USA. ACM.

[69] Lopez, A., Nicolas, J., and Toval, A. (2009). Risks and safeguards for the requirements engineering process in global software development. In *Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering*, ICGSE '09, pages 394–399, Washington, DC, USA. IEEE Computer Society.

[70] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. (2011a). Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2857–2866, New York, NY, USA. ACM.

[71] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. (2011b). Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2857–2866, New York, NY, USA. ACM.

[72] Marques, A. B., Rodrigues, R., and Conte, T. (2012). Systematic literature reviews in distributed software development: A tertiary study. In *2012 IEEE Seventh International Conference on Global Software Engineering*, pages 134–143.

[73] Marshall, C. et al. (2014). Tools to support systematic reviews in software engineering: A feature analysis. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 13:1–13:10, New York, NY, USA. ACM.

[74] McKibbon, K. (1998). Evidence-based practice. *Bulletin of the Medical Library Association*, 86(3):396.

[75] Mohagheghi, P., Dehlen, V., and Neple, T. (2009). Definitions and approaches to model quality in model-based software development - a review of literature. *Inf. Softw. Technol.*, 51(12):1646–1669.

[76] Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. (2012). What makes a good code example?: A study of programming q amp;a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34.

[77] Nicolás, J. and Toval, A. (2009). On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology*, 51(9):1291 – 1307.

[78] Papazoglou, M. P. (2003). Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pages 3–12.

[79] Parnin, C. and Treude, C. (2011). Measuring api documentation on the web. In *Proceedings of the 2Nd International Workshop on Web 2.0 for Software Engineering*, Web2SE '11, pages 25–30.

[80] Pino, F. J., García, F., and Piattini, M. (2008). Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Journal*, 16(2):237–261.

[81] Pinto, G., Castor, F., and Liu, Y. D. (2014). Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA. ACM.

[82] Pinto, G., Torres, W., and Castor, F. (2015). A study on the most popular questions about concurrent programming. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2015, pages 39–46, New York, NY, USA. ACM.

[83] Pinto, G. H. and Kamei, F. (2013). What programmers say about refactoring tools?: An empirical investigation of stack overflow. In *Proceedings of the 2013 ACM Workshop on Workshop on Refactoring Tools*, WRT '13, pages 33–36, New York, NY, USA. ACM.

[84] Posnett, D., Warburg, E., Devanbu, P., and Filkov, V. (2012). Mining stack exchange: Expertise is evident from initial contributions. In *2012 International Conference on Social Informatics*, pages 199–204.

[85] Rebouças, M., Pinto, G., Ebert, F., Torres, W., Serebrenik, A., and Castor, F. (2016). An empirical study on the usage of the swift programming language. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 634–638.

[86] Riaz, M., Mendes, E., and Tempero, E. (2009). A systematic review of software maintainability prediction and metrics. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pages 367–377, Washington, DC, USA. IEEE Computer Society.

[87] Robillard, P. N. (1999). The role of knowledge in software development. *Commun. ACM*, 42(1):87–92.

[88] Santos, R. E. S. and d. Silva, F. Q. B. (2013). Motivation to perform systematic reviews and their impact on software engineering practice. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 292–295.

[89] Santos, R. E. S., da Silva, F. Q. B., de Magalhães, C. V. C., and Monteiro, C. V. F. (2016). Building a theory of job rotation in software engineering from an instrumental case study. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 971–981, New York, NY, USA. ACM.

[90] Saraiva, J., Barreiros, E., Almeida, A., Lima, F., Alencar, A., Lima, G., Soares, S., and Castor, F. (2012). Aspect-oriented software maintenance metrics: A systematic mapping study. In *16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012)*, pages 253–262.

[91] Stol, K.-J., Ralph, P., and Fitzgerald, B. (2016). Grounded theory in software engineering research: A critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 120–131, New York, NY, USA. ACM.

[92] Sykes, E. R. (2011). Interruptions in the workplace: A case study to reduce their effects. *International Journal of Information Management*, 31(4):385 – 394.

[93] Trendowicz, A. and Münch, J. (2009). Chapter 6 factors influencing software development productivity—state-of-the-art and industrial experiences. volume 77 of *Advances in Computers*, pages 185 – 241. Elsevier.

[94] Treude, C., Barzilay, O., and Storey, M.-A. (2011a). How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 804–807, New York, NY, USA. ACM.

[95] Treude, C., Barzilay, O., and Storey, M.-A. (2011b). How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 804–807, New York, NY, USA. ACM.

[96] Treude, C., Prolo, C. A., and Filho, F. F. (2015). Challenges in analyzing software documentation in portuguese. In *2015 29th Brazilian Symposium on Software Engineering*, pages 179–184.

[97] Vanhanen, J. and Lassenius, C. L. (2007). Perceived effects of pair programming in an industrial context. In *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*, pages 211–218.

[98] Vasilescu, B., Capiluppi, A., and Serebrenik, A. (2012). Gender, representation and online participation: A quantitative study of stackoverflow. In *2012 International Conference on Social Informatics*,

pages 332–338.

[99] Vasilescu, B., Serebrenik, A., Devanbu, P., and Filkov, V. (2014). How social q&a sites are changing knowledge sharing in open source software communities. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '14, pages 342–354.

[100] Verner, J. M., Brereton, O. P., Kitchenham, B. A., Turner, M., and Niazi, M. (2014). Risks and risk mitigation in global software development: A tertiary study. *Inf. Softw. Technol.*, 56(1):54–78.

[101] Viera, A. J., Garrett, J. M., et al. (2005). Understanding interobserver agreement: the kappa statistic. *Fam Med*, 37(5):360–363.

[102] Walia, G. S. and Carver, J. C. (2009). A systematic literature review to identify and classify software requirement errors. *Information and Software Technology*, 51(7):1087 – 1109. Special Section: Software Engineering for Secure SystemsSoftware Engineering for Secure Systems.

[103] Webb, S. A. (2001). Some considerations on the validity of evidence-based practice in social work. *British journal of social work*, 31(1):57–79.

[104] Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 38:1–38:10, New York, NY, USA. ACM.

[105] Yaman, S. G., Sauvola, T., Riungu-Kalliosaari, L., Hokkanen, L., Kuvaja, P., Oivo, M., and Männistö, T. (2016). *Customer Involvement in Continuous Deployment: A Systematic Literature Review*, pages 249–265. Springer International Publishing, Cham.

[106] Zhou, Y., Zhang, H., Huang, X., Yang, S., Babar, M. A., and Tang, H. (2015). Quality assessment of systematic reviews in software engineering: A tertiary study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, pages 14:1–14:14, New York, NY, USA. ACM.