UNIVERSITÄT
BERN

Software Developers' Information Needs

Bachelor Thesis

Jonas Richner

from

Gränichen, Switzerland

Faculty of Science
University of Bern

1 February 2019

Prof. Dr. Oscar Nierstrasz
Pooja Rani

Software Composition Group
Institute of Computer Science

University of Bern, Switzerland

Abstract

Nowadays software shapes most aspects of our daily life, and application developers are increasingly confronted with complex scenarios which demand collecting information from various sources. As developers face many challenges in seeking desired information, a substantial amount of research has been performed to understand software developers' information needs. However, little is known regarding researchers' use of methods and measures to quantify developers' information needs, and there has not been any effort yet to systematically select, review, and synthesize developers' information needs that are presented in the literature. Consequently, we investigate this domain by conducting a systematic literature review. Through our search strategy, we identified 60 relevant papers. We discovered that the most common way to quantify developers' information needs is to use surveys, and that some of the most valuable information that developers need to make progress considers the rationale (e.g., *why is this implemented this way?*), awareness (e.g., *what have my coworkers been doing?*), and the implications of a code change (e.g., *what might break?*). This thesis presents the review's findings, which we believe will help scientists studying developers' information needs by enabling them to use appropriate methods and measures for their research, as well as assist researchers in developing tools and practices that are better aligned with developers' needs.

Contents

1 Introduction

Technology has become an increasingly important part of our everyday life. The number of internet users worldwide has doubled over the last decade; today more than half of the world has access to the internet through computers or smartphones [75]. At the same time software applications have become more complex and massive systems like the Linux kernel grew even larger. The Linux operating system increased from 2.4 million lines of code in 2001 [77], to over 25.3 million in 2018 [61]. Nowadays even modern cars can contain more than 100 million lines of code, as features like collision avoidance, navigational systems, and automatic emergency braking contribute to this rapid increase in complexity [72].

Building and maintaining complex systems is a time-consuming and expensive process that requires the collaboration of many people. Successfully writing code requires developers to have in-depth knowledge in multiple fields, and coordinate with their team effectively. However, even when developers try their best, the software still ships late and buggy [66]. Due to the complexity of the work, developers need to exchange a variety of information to understand the system's behavior and its components [70]. Specifically, developers need information regarding the rationale of a piece of code [25], what coworkers have been doing [20], which changes will break code elsewhere [26], and who has experience with a component [6].

Hence, we can understand what contributes to these difficulties by identifying which information needs lead to exhausting and time-consuming searches. In order to find information, developers search various sources such as the documentation, the source code, online resources, or they interrupt coworkers who might have the knowledge that they seek [20] [56]. Developers often develop hypothesis about the code and ask questions to test their assumptions [70]. Gathering these questions, *i.e.*, the information needs, from various sources and understanding them in context can guide the development of tools and practices that help developers find the desired information more easily.

Scientists performed a substantial amount of research to obtain and understand these information needs. They study developers' searching behavior by observing them either in industry or with a predefined task in the lab, they use self-reporting in surveys, and collect qualitative data in interviews. Finally, researchers also mine various software development artifacts such as Stack Overflow and mailing lists to collect developers' questions.

Our goal is to find the most important questions that developers ask, *i.e.*, questions that use the most resources and have the biggest impact on the progress of a software project. For this task, we need to understand the methods and measures that researchers use to gather and quantify developers' questions. Thus we assess which methods researchers use to collect the questions *e.g.*, surveys, or observations, and which measures they use to evaluate their importance *e.g.*, by measuring the frequency, or the time taken to answer the question.

In order to answer these questions, we conducted a systematic literature review on software developers' information needs and investigate the following research questions:

- **RQ**$_1$: *How do researchers quantify software developers' information needs?* We reviewed relevant literature and collected the research methods and measures used to quantify the information needs, as well as the rationale behind them. Our results reveal that researchers quantify the information needs primarily using self-reporting in surveys and by studying the questions that are asked on Stack Overflow. Furthermore, we observed that there are only three observational studies that assess the importance of a question using a measure other than how frequently it was asked.

- **RQ$_2$**: *Which information needs are important for developers to make progress, and useful for the development of tools and practices?* We reviewed relevant literature and gathered information needs that researchers consider useful for the development of tools and practices. We then assessed which of those information needs are important for developers to make progress by synthesizing the results of the papers included in our study. We find three useful and important information needs that are supported by many studies, namely information about rationale (e.g., *why is this implemented this way?*), awareness (e.g., *what have my coworkers been doing?*), and the implications of a code change (e.g., *what might break?*).

We believe that our work will support researchers by providing an overview of the current research methods and sources used to collect developers' information needs. Moreover, we assist researchers in assessing the importance of information needs by comparing different measures such as how often a given type of information was sought, and how much time it took to obtain it. Finally, we provide a list of important categories of information needs that are supported by a wide range of studies. We believe that these needs can guide the work on tools and practices that make software development more productive.

The remainder of this thesis is structured as follows: chapter 2 describes the methodology we follow, chapter 3 investigates the quantification of information needs, and in chapter 4 we discuss the most important ones. We consider threats to validity in chapter 5, we discuss future work in chapter 6 before we conclude with chapter 7.

2 Methodology

We closely follow Keele's guidelines on conducting systematic literature reviews in software engineering. This approach makes it more likely that the results will be unbiased and reproducible. It consists of the following steps: i) We state our research questions, ii) perform a preliminary search, iii) define our inclusion and exclusion criteria for the primary studies, iv) state what data we extract, v) describe which data sources and search strategy we use, and finally, vi) state which studies were selected [69].

2.1 Preliminary Search

In accordance with the guidelines, we performed a preliminary search to develop a review protocol. This allowed us to refine our research questions, identify the search terms, decide which data we will extract from the studies, and develop detailed inclusion and exclusion criteria. We found that researchers most often study developers' information needs using interviews, observations and surveys or use Stack Overflow and mailing lists as data sources to extract developers' questions.

2.2 Inclusion and Exclusion Criteria

In order to focus on relevant primary studies, we define several criteria for inclusion and exclusion as shown in Table 2.1. The criteria lead to papers that focus either in whole or in part on the information needs of software developers and are relevant to our research questions. A study is selected if it satisfies all of the inclusion criteria, and excluded if it fulfills any of the exclusion criteria.

2.3 Data Extraction

In accordance with Keele's guidelines, we state which data we will extract from the primary studies. We will collect the data sources and methods that were used to study developers' information needs, the measures that were employed to quantify them, as well as other information such as the title of the paper and the primary author as shown in section A.1.

2.4 Data Sources and Search Strategy

To find appropriate studies we first searched for terms relevant to our research questions. However, we found that even when we used many different variations of search terms and different scholarly search engines such as *Google Scholar*, *IEEE Xplore*, and *ACM digital library*, we did not find all of the studies from our preliminary search. In the end we primarily used *Google Scholar* for searching and citation tracking, as it is considered an appropriate data source [64], and delivered the most comprehensive results in our study.

We adopted an iterative search strategy in which we first used terms like "software

Criterion	Rationale
Inclusion Criterion 1. A study that focuses in whole or in part on software developers' information needs.	We want to identify the most important information needs of software developers, thus we need studies that assess developers working directly with software (*e.g.*, not managers).
Inclusion Criterion 2. A study where the information needs are related to software development.	We want to find which type of information software developers need to make progress. Thus information needs should be related to the process of software development.
Inclusion Criterion 3. A study that is peer-reviewed.	A peer-reviewed study provides some level of quality.
Inclusion Criterion 4. A study that is written in English.	For feasibility reasons only papers written in English are included.
Inclusion Criterion 5. A study that includes empirical evidence.	We want studies with empirical evidence, not lessons learned or expert advice.
Exclusion Criterion 1. A study that only contains information needs from other studies and does not provide any new quantitative data on those information needs.	We do not want papers that only contain information needs from other studies, as this duplicate data can bias our results. However, if the publication provides new quantitative information on the results from other studies, then it is of interest.
Exclusion Criterion 2. A study where the only focus is to evaluate a tool.	We want studies that assess information needs and not just how many information needs a tool can answer.

Table 2.1: The inclusion and exclusion criteria for the primary studies

developer information needs” in *Google Scholar* (see section A.2 for the list of all search terms) and then continued with all of the cited and citing papers that we found. In more detail, we searched for seven search terms in the first iteration, with each search limited to the first 100 results, as we did not find any more relevant papers after this threshold. These results include duplicates because we searched for the terms sequentially. Then in each iteration, we investigated all the cited and citing studies of the discovered papers, until we did not find any new papers. Searching through the cited studies of the returned papers provides additional reliability as it is independent of the data source and search engine that is used.

In Table 2.2 we list the number of papers that we searched through, which contain duplicates because some papers have been cited or referenced more than once. More details can be found in section A.3, which provides the list of the new papers that we found after each iteration.

One can see from Table 2.2 that the number of papers found through citation and reference searching dropped rapidly after the first iteration. We searched through 2 174 studies in the second iteration, yet we only discovered three new papers. This decline in newly discovered papers is probably due to many studies referencing and citing each other; thus we believe it indicates that we found most of the relevant studies in our field of research.

Iteration	Results Searched	Cited Papers Searched	Citing Papers Searched	Papers Included
Initial Search	700	0	0	35
First Iteration	0	978	3 820	31
Second Iteration	0	840	1 334	3
Third Iteration	0	74	304	1
Fourth Iteration	0	28	66	0

Table 2.2: The number of results, cited and citing papers that we searched

When applying our inclusion and exclusion criteria we proceeded as follows: i) We first excluded papers based on the title. ii) When it was unclear from the title if the study should be included or excluded, we reviewed the abstract. iii) When it was still not clear if the study should be included or excluded, we considered the content of the paper.

iv) When in doubt, we always included the study and only excluded it later based on further examination.

2.5 Relevant Literature

This process yielded 70 papers, from which ten were afterwards excluded as they did not pass our inclusion and exclusion criteria, resulting in a total of 60 papers that are relevant for answering $\mathbf{RQ}_1$. The included publications are listed in Table A.5.

To answer $\mathbf{RQ}_2$, we define two additional exclusion criteria that select a subset of the papers in our study. First, we exclude papers that do not contain quantitative data or that explicitly state that they are qualitative instead of quantitative. Second, we exclude studies that are more than 15 years old, because tools and practices might have changed over time and we want to find the current most important information needs of developers. This leaves us with 43 studies that are listed in Table A.8.

As suggested by Keele's guidelines, we also maintain a list of excluded studies. This list does not contain all of the studies that we investigated in our systematic literature review. Instead, it consists of edge cases or papers that were excluded for other reasons that were not immediately evident from the title or abstract, such as papers that only contain duplicate data from other studies as mentioned in Table A.11.

3 Quantifying Information Needs

In this section, we provide an overview of the selected studies and analyze the various data sources and methods that researchers use to quantify developers' information needs.

3.1 Overview

The relevant studies date from 1987 to 2018, as shown in Figure 3.2. Our studies include 15 different originating countries, although the United States and Canada combined account for more than half of the studies. They span 35 different journals, conferences, workshops and symposiums, of which the seven most common ones are listed in Table 3.1.

In Figure 3.1, we show how the number of papers published in the most common conferences, workshops, and journals changed over time. While the *International*

Conference on Software Engineering published most studies before 2014, the papers are now distributed among many different conferences and journals, such as the *Empirical Software Engineering* journal and the conference on *Mining Software Repositories*.

Short name	Full name	# Studies
ICSE	International Conference on Software Engineering	11
MSR	Mining Software Repositories	5
FSE	International Symposium on Foundations of Software Engineering[1]	4
PLATEAU	Workshop on Evaluation and Usability of Programming Languages and Tools	3
CHASE	International Workshop on Cooperative and Human Aspects of Software Engineering	3
EMSE	Empirical Software Engineering	3
IWPC	International Workshop on Program Comprehension	3

Table 3.1: The number of studies published in the seven most common conferences, journals and workshops

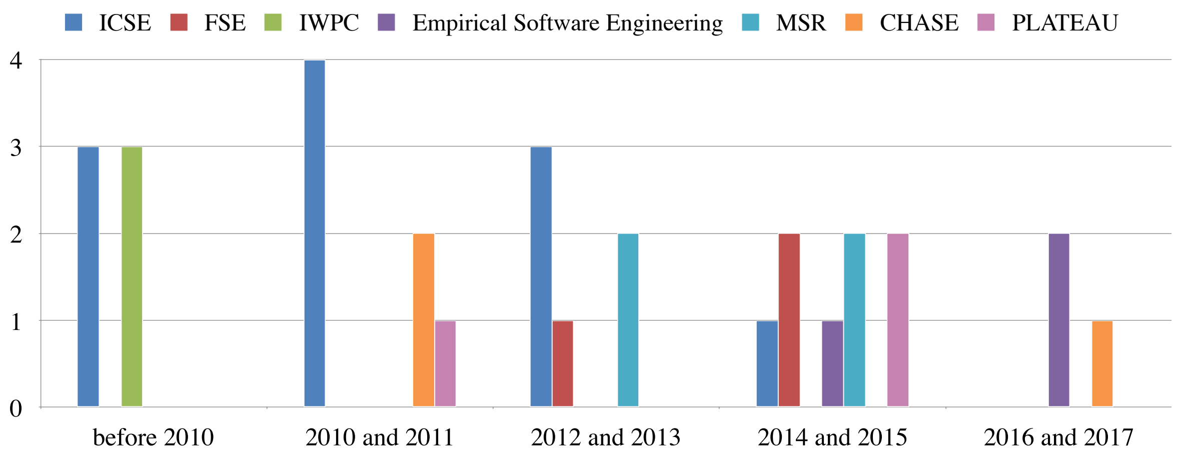

Figure 3.1: The number of studies published per year in the seven most common conferences, journals and workshops

[1]This symposium includes works of the ESEC/FSE joint conference as this is the new name of this conference series.

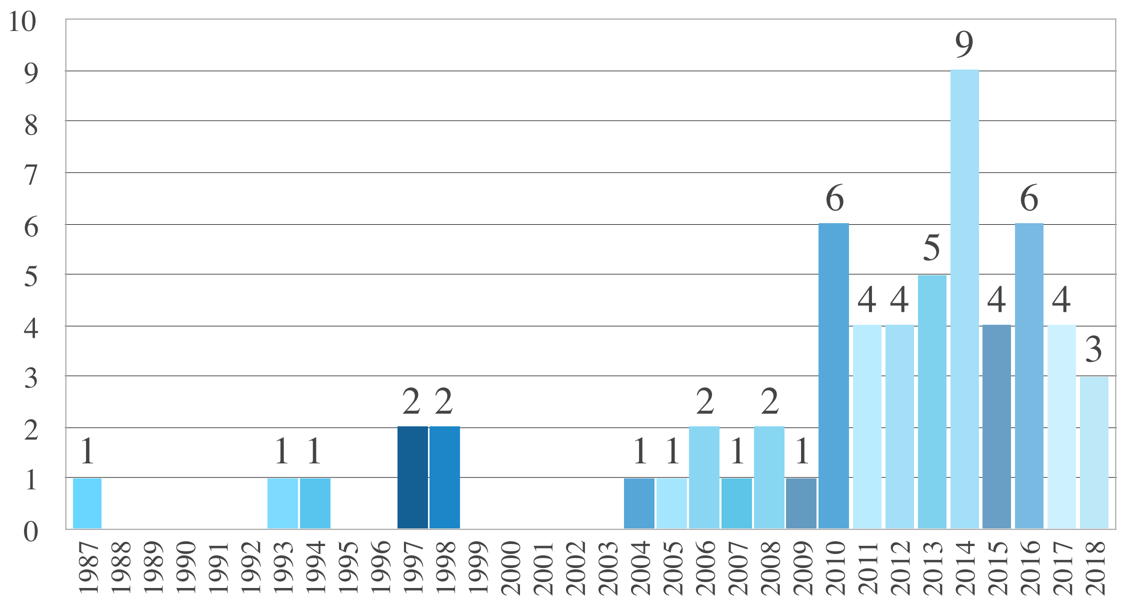

Figure 3.2: The number of studies published per year

We categorized the studies according to which data source they use. We classify 35 (58%) of the papers as *People-centric*. They refer to studies that recruit participants directly for observations, surveys, and interviews. The other 25 (42%) papers are *Technology-centric*, as they study an artifact of the development process such as Stack Overflow or mailing lists.

3.2 People-centric Studies

When researchers recruited participants directly, they used a variety of methods to obtain data, as shown in Figure 3.3. The categories are not mutually exclusive. Sixteen studies use more than one method, for example, an interview is often conducted after observing participants.

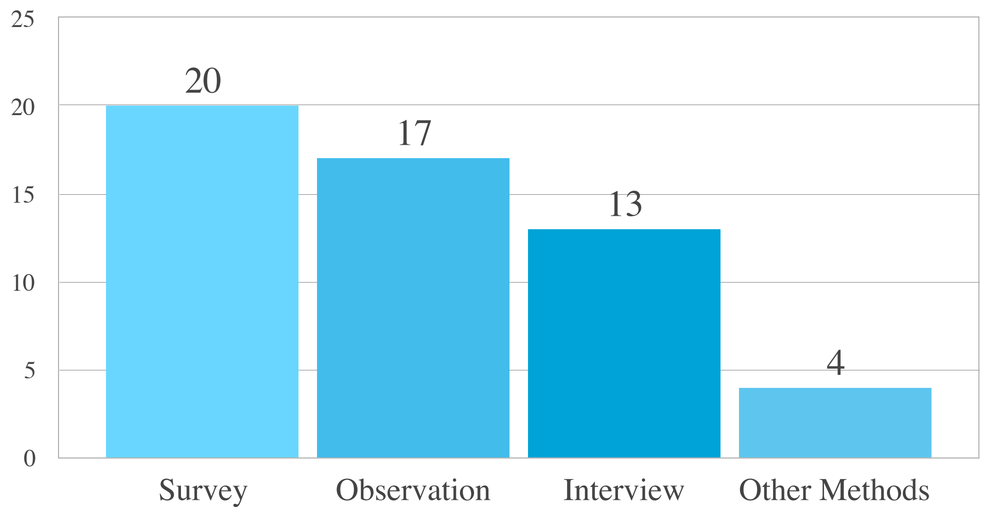

Figure 3.3: The methods used for studying the information needs of developers directly

3.2.1 Surveys

Surveys are a convenient way to collect and assess developers' questions. Compared to interviews and observational studies, they are inexpensive to carry out, even with many participants across different organizations and countries. We distinguish between surveys that are closed-form, where the participants rate information needs from predefined answer choices, and those that are open-ended, where participants can write down their own information needs. Studies that use both kinds of survey methods use the open-ended section to allow developers to provide additional information needs that the authors might have missed in the closed-form survey. The numbers of surveys of each type are shown in Table 3.2. The categories are mutually exclusive.

Survey Type	# Studies
closed-form	11
open-ended	5
closed-form and open-ended	4

Table 3.2: The number of closed-form and open-ended surveys

The studies assess the importance of an information need in several ways. Participants most often rate how frequently they need some information, how difficult it is to acquire, or how important it is for them to make progress.

Some ways of asking participants to rate their information needs might be able to combine these dimensions with a single measure, such as asking developers if a given information need represents a serious problem for them [26], or asking participants to pick information needs where they think that it would have a big positive impact on their workday if they had a tool available to address this need [6].

Open-ended surveys. The studies that use open-ended surveys usually report how often a given question was asked. For example, LaToza *et al.* ask their participants to provide *"hard-to-answer questions about code"* and then abstract those specific questions into more general questions. At the end, they report how many participants asked a specific type of question [25].

Closed-form surveys. From the 15 studies that use closed-form surveys, only five of them evaluate the importance of a question using multiple measures. Those that use multiple measures or methods for assessing the importance of a question often discover results that could not be found otherwise. For example:

- Tao *et al.* found that participants consider rationale the most important information need, but at the same time they rate it as easy to acquire. We believe that if the authors just would have asked participants to rate how important an information need is, they would have missed this fact [46].
- LaToza *et al.* discovered positive correlations between a question's difficulty and how often it is asked [24].
- Buse *et al.* discovered that predicting the future is rated as harder to acquire than understanding the past, but that participants rated predicting the future as less important. Again, if the participants would have only rated how difficult the information is to acquire, the results could have been misleading, as the information rated difficult to acquire was also considered the least important [11].
- Ko *et al.* found that the implications of a change are rated as difficult to acquire, but surprisingly in their observations they found that this information is easily

obtained. This reveals a mismatch between perceptions of the participants and reality [20].

3.2.2 Observations

Observational studies have been performed either in an equipped lab, or in industry. The number of observational studies that were conducted in the lab and the ones that were performed in industry are shown in Table 3.3. The categories are again mutually exclusive.

Observational Setting	# Studies
Lab	7
Field study in industry	6
Include both, a field study and a study in the lab	4

Table 3.3: The number of studies that were conducted in the lab and in the field

In contrast to surveys, observational studies are not subject to misremembering, misreporting, or inaccurate estimations by the participants. However, they can be influenced by the observers' biases and are limited to what can be observed. Ko *et al.* note that they could not observe some instances of information seeking because they were "either too subtle, like glancing at a coworker's instant messenger status, or invisible, like the use of memory to recall facts about the code" [20]. Observational studies can also be expensive to carry out. Kubelka *et al.* note that it took them one and a half days to analyze just 40 minutes of footage from their study [23]. The studies often use the think-aloud technique in combination with audio and video recordings.

From the 17 observational studies in our dataset, nine (53%) of them are qualitative. They occasionally provide information regarding the frequency of given information needs, but some authors explicitly advise the reader against using their data to quantify the information needs. The remaining eight (47%) studies are quantitative, and most of them are based on the frequency of which a given information which was sought. Surprisingly, only three of the eight quantitative studies use other measures to assess the importance of a question:

- Ko *et al.* measured the time it took developers to satisfy an information need and how frequently the information was sought. They also measured how often the search was deferred or abandoned, because some developers immediately gave up on searching for information that they knew only existed in the mind of an unavailable coworker [20].
- Duala-Ekoko *et al.* decided against using the amount of time taken to answer a question to measure the difficulty, because there is significant variability of how long it takes developers to answer a question, even within similar levels of experience. Interestingly, they observed that some actions indicated a lack of progress. For example, the researchers measured how often participants had to backtrack on the assigned actions when they realized that their search strategy led down the wrong path [12].
- LaToza *et al.* measured the time spent on answering a question. Additionally, whenever developers inserted defects, they looked at the questions that developers had asked that led them to information that they did not understand correctly. Thus, they measured which questions were associated with the most defects [24].

Classifying the statements of participants in observational studies can be challenging, as their utterances could include both implicit or explicit questions. In order to extract questions from the observations some researchers define a protocol. For example, the protocol of the publication “Questions developers ask while diagnosing potential security vulnerabilities with static analysis” from Smith *et al.* [45]:

A statement was coded as a question only if it met one of the following criteria:

- **The participant explicitly asks a question.**
  Example: Why aren’t they using PreparedStatements?
- **The participant makes a statement and explores the validity of that statement.**
  Example: It doesn’t seem to have shown what I was looking for. Oh, wait! It’s right above it...
- **The participant uses key words such as, “I assume,” “I guess,” or “I don’t**

know.”
Example: I don’t know that it’s a problem yet.

- **The participant clearly expresses uncertainty over a statement.**
  Example: Well, it’s private to this object, right?
- **The participant clearly expresses an information need by describing plans to acquire information.**
  Example: I would figure out where it is being called.

3.2.3 Interviews

Most interviews are follow-up interviews conducted after the observations or surveys to collect qualitative data. We found only one study where the focus of the interview was to collect questions that developers asked. There are no interviews that assess developers’ information needs quantitatively.

3.2.4 Other methods

Xia *et al.* installed a tool on participants’ computers that collected all of the queries that were submitted to search engines. Based on these results they could later analyze those queries to find how frequently a developer was looking for a specific type of information [56]. Sadowski *et al.* use a similar methodology with a company-internal code search tool [38]. Kim conducted a focus group study and recorded the conversation [19], while Müller *et al.* asked the participants to write a diary at the end of each day [30].

3.3 Technology-centric Studies

Twenty-five (42%) of the papers included in our study are *Technology-centric*, as they study an artifact of the development process such as Stack Overflow or mailing lists. The number of papers per data source included in our study can be seen in Figure 3.4.

The categories are not mutually exclusive, but only one study uses data from multiple sources.

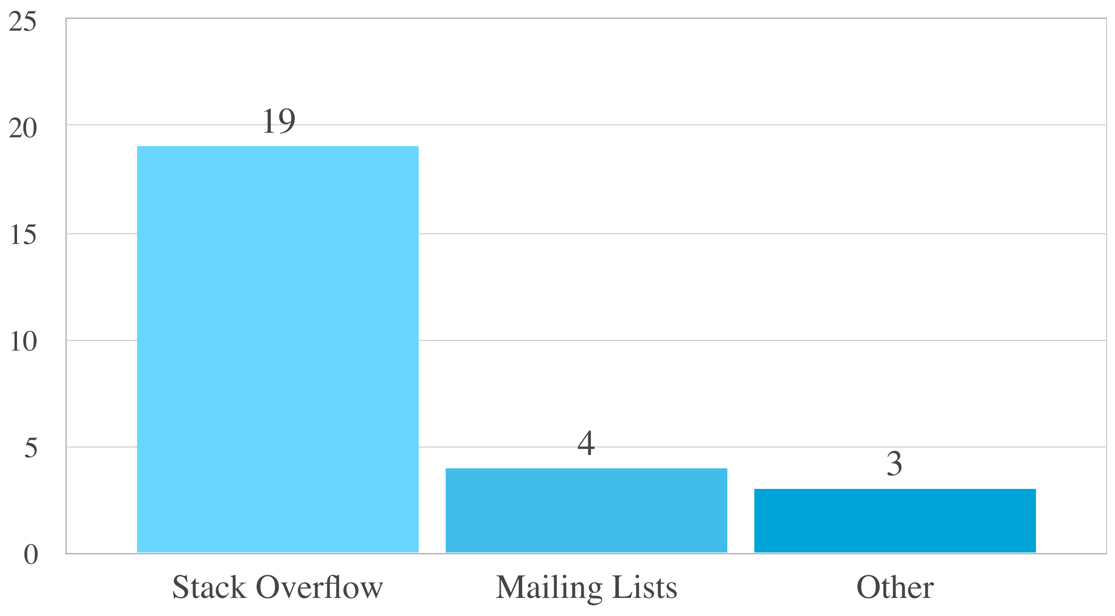

Figure 3.4: The number of papers per data source

3.3.1 Stack Overflow

Stack Overflow[2] is a Q&A website for programmers. Among many features, it provides users with a way to ask and answer questions, vote questions up or down, and edit them. Stack Overflow aims to be a wiki-type repository of knowledge that is easily searchable [63]. Most developers seeking answers do not ask questions on Stack Overflow, instead they use search engines that lead them to questions that already have answers [62].

Stack Overflow has some rules on what types of questions are allowed on their website, for example, they do not allow questions on opinion. There are also other factors that limit the types of questions that are asked. For example, what an employees' coworkers have been working on cannot be answered by strangers.

There are a variety of ways that researchers use to measure the importance of a question

[2]`https://stackoverflow.com`

on Stack Overflow. A few studies just count the *number of questions* in a given category. However, this does not take into account that some questions were viewed millions of times while others have almost no views. Thus some researchers also sum up the *views of the questions* in their categories. Other scientists believe that this is not a valid measure for the popularity of a question, because a question with many views may be prioritized by search engines, leading many developers to view that question although they may be looking for something else. As a result, they also consider the *number of answers*, the *number of comments*, the *number of users who added the question to their favorites list* as well as the *number of upvotes and downvotes* to measure the popularity of a question. Furthermore, to find the difficulty of a question, some researchers use metadata such as how long it took to get a response, or if the question has an accepted response.

3.3.2 Mailing lists

Software developers also use mailing lists as a communication channel for collaboration. However, compared to Stack Overflow, this data is less structured, since one e-mail can contain multiple questions or even discussions. For this reason, it requires more work to extract the data from e-mails than from Stack Overflow. Researchers measure the *number of questions* about a given topic, *if the questions were answered*, the *number of responses*, the *delay before the first response*, and the *average response time*.

3.3.3 Other sources

We also found two papers that gather developers' information needs from newsgroups and one paper that uses bug reports as a data source. They measure the *number of questions in a given category*, *how many questions get asked* and *how long it takes to get a response*.

3.4 Discussion of RQ$_1$

We investigated how researchers quantify software developers' information needs (**RQ**$_1$).

We found that most papers that directly study the information needs of developers use surveys, and most surveys assess the importance of a question with just a single measure. However, the importance of an information need depends on many factors, such as how hard it is to satisfy, or how frequently it is sought. We believe that it is not enough to just measure how frequently a given type of information is sought, as it might be easily obtainable or might not be important for the progress of the project. Some ways of asking participants to rate their information needs might be able to combine these dimensions with a single rating such as *"this represents a serious problem for me"*. We found that only five surveys measure the importance of a question using multiple dimensions, with many of them leading to non-obvious insights. However, it is unclear how reliable these measures are because they are subject to misremembering, inaccurate estimations, and misreporting. Given that surveys are the most common way to measure how important an information need is and that some researchers found discrepancies between self-reported statements and observations, we believe that it would be useful to know which measures are reliable and which measures might have problems with self-reporting.

Furthermore, psychological research suggests that rating the frequency of a given information need may be especially prone to bias. People overestimate the frequency of events that are easier to recall from memory, which can be influenced by factors such as how recent or emotionally charged the memory is, which can lead to systematic biases [76]. Other measures such as rating how difficult a given type of information is to acquire could also be affected by well-known biases such as how easy it is to come up with a typical example [71].

In our opinion it would be useful to know if there are differences in the reliability of self-reporting between different types of information needs. For example, finding which changes broke your code might be combined with more negative feelings leading to an oversized estimation of its difficulty and frequency. Understanding which measures and information needs are more prone to bias could lead to better techniques for assessing them.

3.5 Results

We discovered that the quantification of information needs is a crucial part of many studies, as the most important information needs guide the development of tools and practices. However, there has been no meta-study yet that analyses how researchers quantify information needs. In order to shed light on this topic, we studied how researchers quantify developers' information needs. We found that in 35 (58%) of the papers the researchers study developers' information needs directly, whereas in 25 (42%) papers they study an artifact of the development process such as mailing lists and Stack Overflow.

Of the papers that study information needs of developers directly, most of them acquire the data through surveys. Only 25% of the surveys use more than one measure to assess the importance of a question, such as how frequently it is asked or how difficult it is to answer. The surveys that use multiple measures often contain insights that would be impossible to find with only one measure. We also found that only three of the quantitative observational studies use a measure other than the frequency with which a given type of information was sought to assess the importance of an information need.

It might be worth doing a follow-up study to find the percentage of time developers spend searching for answers on Stack Overflow and mailing lists in relation to the time they spend on other information needs. If the amount of time spent and the mental drain from context switching turns out to be small in relation to other information needs, then research on tools that answer questions that can already be efficiently answered on Stack Overflow might not be optimally aligned with the needs of developers.

4 Important Information Needs

In this section, we first discuss the challenges that we faced when synthesizing the data and then describe the methodology that we used to find categories of information needs that are both useful for the development of tools and practices and have the most empirical support. Finally, we present our results by summarizing the most important information needs and the solutions that researchers propose.

4.1 Challenges of Synthesizing Data

Synthesizing data across studies is apparently difficult, as most systematic literature reviews in software engineering do not synthesize results [67]. We want to synthesize data from the papers included in our study to find which important information needs have the most empirical support. We discovered that this is very challenging. The research methods and data sources, as well as the quantitative evaluation of developers' questions, are diverse. Thus we dedicate this section to the difficulties that we faced when synthesizing results from the studies.

We first discuss issues with comparing the question categories of researchers, then we look at the difficulties of partitioning the questions, we highlight the problems of comparing different measures, and finally, we present the complications concerning the scope of the studies.

4.1.1 Categorization

There are many different ways to categorize developers' questions.

Sometimes researchers cluster the questions just for readability and to provide a structure to the paper. For example, Ko *et al.* categorized questions by the work category in which they arose, such as writing code, fixing bugs, or reasoning about design. This is not an issue because they also provide more fine-grained categories of developers' questions, such as *"Why was this code implemented this way?"* or *"What are the implications of this change?"*. They abstracted 21 information needs like these from 334 concrete questions that developers asked. For example, the question category *"What are the implications of this change?"* can come in many concrete forms, such as *"Does this change break any code elsewhere?"*, *"How does this change alter the program's dynamic behavior?"*, or *"Did this change miss any place that should also be changed at the same time?"* [20] [46].

Researchers need to abstract these concrete information needs into general categories of information needs to make sense of the multitude of data. However, for our needs of aggregating and synthesizing this data it is problematic when researchers choose to only report high-level categories of information needs, such as *how-to*, *discrepancy*, and *error* questions [47], or *user interface*, *web document* and *stack trace* questions [55], or *Domain concept descriptions*, and *location and uses of identifiers* question categories [52]. This can make it impossible to extract the fine-grained questions and relate findings across papers when the questions have been reduced to high-level categories.

Some researchers tried to compare their findings with other papers. For example, La-Toza *et al.* define reachability questions as *"a search across feasible paths through a* *program for target statements matching search criteria"*. They then relate their question category to the findings of two other papers, stating that they believe that 33% of the questions in the paper by Ko *et al.* [20] and 52% of questions that were asked in the paper of Sillito *et al.* [43] might be answerable by reachability questions [24]. This also illustrates that it is impossible to find an exhaustive list of categories, as researchers can constantly develop new categories that they deem useful.

Sharif *et al.* also spend significant effort on mapping their question categories to categories of other researchers, as pictured in Figure 4.1. However, the relationship between those question categories is rarely bijective.

Furthermore, categories with the same name can be different. For example, the design category of Ko *et al.* mainly focuses on why something was implemented a certain way, whereas questions on design identified by Sharif *et al.* focus on how to move from a high-level design to a specific implementation [20] [39]. The boundaries of the categories, such as which questions were included and excluded with their rationale for inclusion and exclusion, are rarely documented. Thus it is unclear how much categories with the same name differ between researchers.

Indeed, even Kubelka *et al.*, who replicated a previous study by Silito *et al.*, wonder how to compare their results with other studies [23]. They note that they found more questions than in the study that they replicated and wonder if this difference can be explained with their study protocol, as they might have adopted a more fine-grained approach. The detailed study protocol for observations is rarely documented and can lead researchers to find different questions. For example, Smith *et al.* find questions on self-reflection in their observations, such as *"Do I understand?"* or *"What should I do first"?*, or *"Have I seen this before?"*, whereas the other observational studies do not. We believe that this is probably due to the study protocol and not that they were the only ones that had participants who asked questions on self-reflection [45].

Another difficulty in assessing the importance of a category is that one can always construct categories that are more important by just including more questions in that category. For example, one can construct a category such as questions about understanding code, which will be evaluated as much more important than questions about inheritance. Indeed LaToza *et al.* find that the categories that were rated as the most frequently needed and

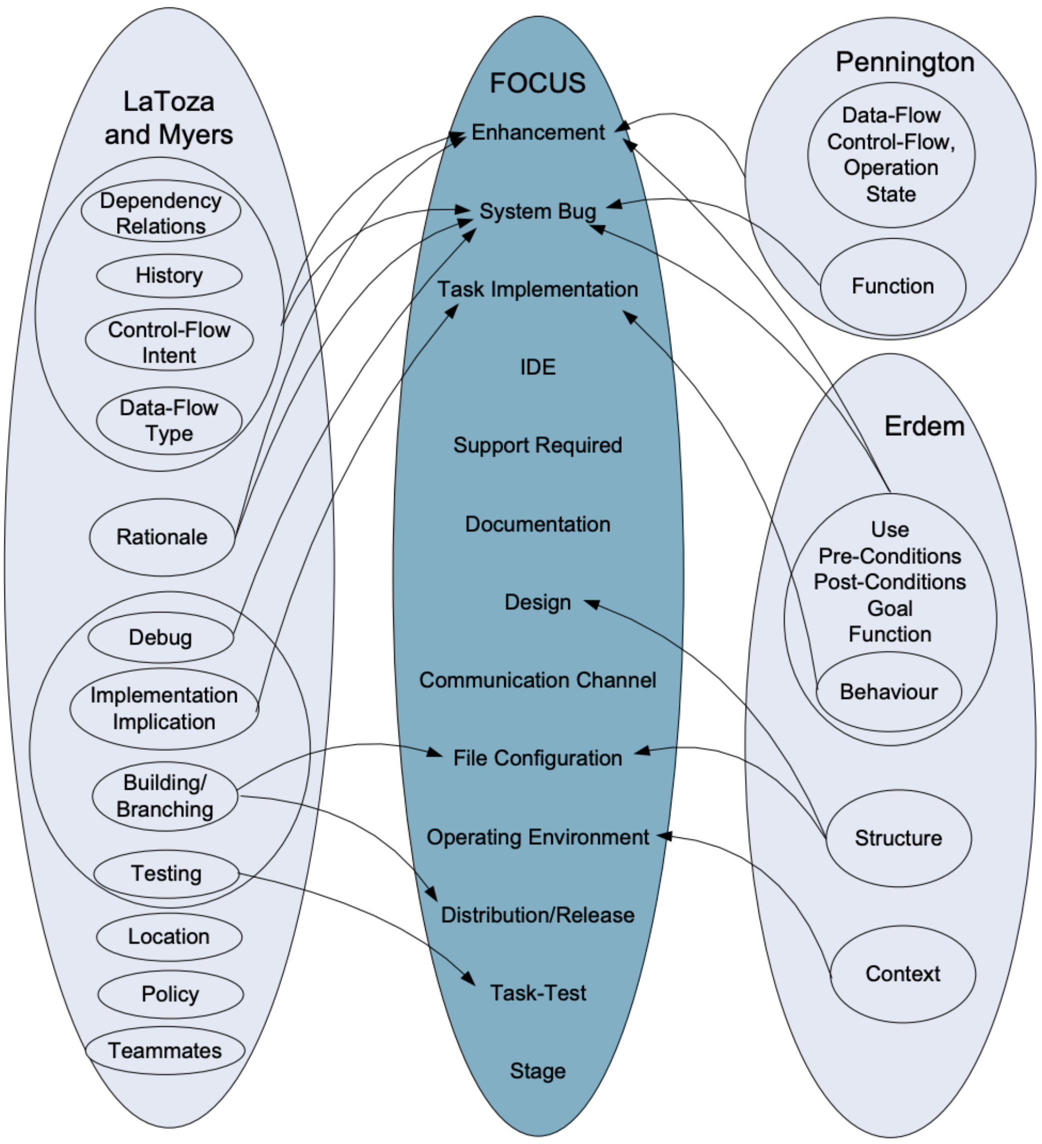

Figure 4.1: A figure from Sharif *et al.* [39] relating their categories with the categories of other researchers

difficult to answer were the most high-level categories [24]. Thus we also need to ask ourselves how useful a category is for the development of new tools and practices.

4.1.2 Partitioning of questions

Sillito *et al.* observe that answering a high-level question is often done by answering multiple low-level sub questions. For example, when searching for an answer to the question *"Which classes are relevant?"*, participants asked several low-level questions about finding certain classes, set breakpoints and ran the application to find out if *"this is the thing"*. Although one may observe developers searching for method names or classes, these questions can be in support of a more high-level question. Sillito *et al.* noted that they sometimes only later realized what the high-level question was that the developer was asking himself [43].

Some observational studies find mostly high-level questions, while others observe more low-level questions. Moreover, which level of granularity the observations focus on is often not explicitly stated in the research protocol. Finally, the high-level question that the developer is trying to answer might not always be obvious to the observer. This makes it hard to compare questions across papers, as some questions might be asked in support of other questions.

4.1.3 Comparing measures

Comparing different measures with each other is challenging. For example, LaToza *et al.* let their participants rate information needs according to the statement *"this is a serious problem for me"* on a 7-point rating scale [26]. In contrast, Begel *et al.* ask their participants to *"pick the tasks that are most important to you, and where if you had a new tool that could make this task easier, it would have a big positive impact on your work day"*. It remains unclear how these measures relate to each other.

Additionally, similar information needs are often stated differently. For example, LaToza *et al.* ask their participants to rate *"understanding who owns a piece of code"*, whereas Begel *et al.* ask their participants to rate *"finding out who owns some code or* *has ever worked on it in the past"*. Thus it is also unclear how differences in the wording of the information needs might influence their rating.

Therefore we believe that comparing only the absolute values is not the ultimate solution. A better way might be to compare the relative values of the information needs listed in a study. If it is one of the most highly rated information needs, then it would be important, and if it is at the bottom of the list, then it would be less important. However, some studies may only ask participants to rate information needs that are all very important, which makes the ones rated as the least important on the list still be very important.

4.1.4 Scope of the study

The studies are often scoped to a specific activity or task, such as which information developers need when diagnosing security vulnerabilities, reasoning about object structure, when understanding code changes, or during software maintenance. The scope of the study is also affected by which data sources researchers chose to study and which research methods they used. Thus if a study does not find a given information need to be important or does not find the information need at all, does that mean that the study does not support the finding that this information need is important? Or was it just out of the scope of the study? Sometimes this is obvious. For example, papers that study the questions on Stack Overflow will not find information needs about what coworkers have been doing. However, often it is not as clear that an information need is out of the scope of a study. For example, when a study is conducted in an artificial lab setting and does not find questions on *if an issue is important enough to fix*. Is this because the study was done in the lab, or does this indicate a lack of evidence that this is an important information need?

4.2 Information Needs

In this section, we first describe our methodology for answering $\mathbf{RQ}_2$, and then present our results by summarizing important categories of information needs. We also list contrary findings and outline the solutions proposed by the researchers.

4.2.1 Methodology

Since we want to find categories that are useful for the development of new tools and practices, we need a way to select categories from the wide range of categories proposed in the research papers. Our assumption is that researchers would not point out a category in the abstract or the conclusion of their paper, if the category is not useful. Thus we looked at the abstract and the conclusion of the collected papers to compile a list of categories to investigate.

We then go through this list and for each category proposed, we look through all of the research papers to see if there is enough support. We then list the papers that are in support of, or provide evidence against the importance of that category.

4.2.2 Results

We found three important categories of information needs that have substantial support across a variety of studies. We note that the findings of most of the 43 papers relevant to $\mathbf{RQ}_2$ neither support nor reject the importance of a given category of information needs, either because the scope of the study is too narrow (*e.g.*, closed-form surveys that only assess a fixed list of information needs), or because the authors do not discuss how their categories relate to other researchers' categories.

4.2.2.1 Rationale

Questions on rationale address why something was done in a certain way, *i.e.*, what was the reasoning behind a decision? Research shows that questions on rationale are moderately prevalent, but they are rated as important and hard to answer [20] [26] [25] [29] [19] [31] [6] [35].

Developers question whether a piece of code is a temporary workaround, how much thought went into it, and if it is implemented this way because of a deep understanding of the problem or because of a lack of knowledge [26] [25].

Ko *et al.* observed that developers who asked why the code was implemented a certain way often had to postpone their search, as the design knowledge only existed in coworkers' minds. Design knowledge is often in unsearchable places such as whiteboards and personal notebooks. In the rare case where this knowledge is documented, it is scattered across design documents, bug reports, personal notebooks, e-mail threads, and code comments. Developers do not search these sources because they think that they are inaccurate and outdated [20] [26].

> As one of the developers in the study states: *"Given that I'll be the one fixing the bugs, I need to make sure I know not what we are doing, but why we are doing it. We have these big long design meetings, and everybody states their ideas, and we come to a consensus, but what never gets written in the spec is why we decided on that. Keeping track of that is really hard."* [20]

Thus obtaining design knowledge comes at a high cost. Developers have to interrupt their coworkers, which decreases their productivity [74]. Moreover, whenever someone needs that same information again, they have to rediscover it by once more interrupting a coworker [26].

Contrary Findings. Tao *et al.* find that knowledge on rationale is the most important information need. However, contrary to most other research they find that this is the most easily acquired information. We conjecture that this is because they only study the information that developers need when understanding code changes, instead of studying the broader information needs that arise in other everyday tasks. Specifically, one participant in their study states that he can easily understand a change rationale from bugs, change list descriptions, review comments and other metadata, but that this data does not accompany the code as it changes over time [46].

Sharif *et al.* find fewer questions on rationale in mailing lists of open source projects than other researchers found in colocated teams. However, this might be because their categories do not match the rationale category of other researchers. They classified questions into various categories including *what* and *why* questions. They state that there are only very few questions in the *why* category, which the rationale category of other researchers maps onto, in their comparison. However, they also note that questions on rationale can be asked both in the form of *why* and *what* questions. Thus it remains unclear if their study finds fewer questions on rationale or if the categories differ too much to reasonably compare them [39].

Proposed Solutions. LaToza *et al.* suggest that providing hyperlinks in the code or using tools that can capture informal whiteboard or paper designs might reduce the cost of using design documents. They ask themselves if the design information that developers seek was even considered by the original developer, and if so, how readable informal notes would be to other developers [26].

Ko *et al.* suggest that recording all the design knowledge might not be cost-effective and result in wasted effort, as it might never be used or might become outdated before being read. Thus they propose that a demand-driven approach might be better. They also note that developers often need to know how trustworthy a source is and that they enjoy face-to-face interactions for exchanging design knowledge [20].

Begel *et al.* note that there are tools that could help. For example, "Deep Intellisense" [68] can present scattered historical artifacts related to a piece of code, such as bug reports, e-mails, check-ins, and specs in a more coherent way, which can help developers infer the rationale behind a piece of code [6].

4.2.2.2 Awareness

Questions about maintaining awareness of coworkers and artifacts are asked frequently. Some of this information may be easy to acquire but can come at a high cost of interrupting coworkers. Developers often want to know who has experience with some piece of code, what coworkers have been doing, who is working on the same file, or how the resources they depend on have changed [20] [39] [41] [19] [16] [25] [31].

The most important information needs are about people rather than artifacts. Developers want to find experts to talk to about a feature, API, or a product [6]. Coworkers are essential information sources that developers rely on for many information needs such as understanding the rationale behind a piece of code. Thus the need for maintaining awareness of coworkers is linked to other information needs that are frequently only in developers' minds [20]. People have meetings or drop by coworkers' offices to find out which issues coworkers face and what is blocking them [20], and often ask friends about who might have experience with a component [6].

Contrary Findings. Maalej *et al.* find that information about awareness is the least frequently sought type of information. They also show that developers in large organizations need significantly more awareness-related information than developers working for smaller companies [29].

Proposed Solutions. Ko *et al.* propose that agile methods such as *Scrum* could help with maintaining awareness. They also point to a few tools such as *FASTDash* [65] where developers can see who is viewing or working on which classes and methods among a variety of other features. Indeed, there are so many different awareness displays that there are even studies that compare their features [73].

Begel *et al.* propose a tool that connects developers with their artifacts, which makes it easier for developers to find relationships such as "who has experience with a particular piece of code" [6].

4.2.2.3 Implications of a change

Developers continuously add new features and fix bugs. As software evolves over time, they need to know the implications their changes have on other code, as well as which other changes affect their code. These questions are difficult to answer, and are rated as important [46] [25] [19] [20] [26] [41] [31] [6] [7].

Developers most often want to know if a change breaks code in another part of a project. They are also concerned whether a change has missed any place that should be changed at the same time, how the change affects the behavior of the program, and if it introduces any security concerns or timing issues [46] [25]. The main approaches to gain some certainty that a change does not break anything are testing and code review. However, writing tests as well as running them can be time-consuming and depends on the desired test coverage. Code review requires manual effort and is subject to human error. Moreover, developers would like to have high confidence in their changes [46].

Contrary Findings. Sharif *et al.* find only very few questions in open-source mailing lists about the implications of a change. They state that possible reasons for this could be that the dataset is largely taken from the initial stages of software evolution, or that it could be because of the difference in research methods used, or that it might indicate a different mindset of commercial and open source software developers [39].

Ko *et al.* observed developers acquiring information about the implications of a change easily, but in the same study the participants rated it difficult to acquire. This reveals a mismatch between the perception of developers and observations. As the other studies mostly use surveys to assess how difficult information about the implications of a change is to acquire, this might indicate that there is a problem with developers' perceptions of this issue [20].

Proposed Solutions. Kim notes that developers often have to go through large numbers of irrelevant changes that do not semantically affect their own changes. She suggests that tools could help developers filter out the irrelevant changes [19].

Understanding the implications of a code change that fixes multiple bugs or implements multiple features is especially hard. Other researchers such as Tao *et al.* tried to reduce the complexity of this problem by suggesting tools that can decompose a large change into many smaller ones that are aligned with a specific issue [46].

4.2.2.4 Other categories

In this section, we discuss a few other categories that might be activity or company specific and some that were impossible to investigate rigorously.

How to use an API. Some studies find this to be an important information need and some do not. We do not know the reason for this variance. Most often developers that are trying to use an API are looking for examples of how to use it and also often wonder if it provides a certain functionality or not.

Understanding how the customers typically use their applications. There are only two studies that assess this. It is also unclear how to measure the impact that this information need has on the progress of a software project.

Understanding the cause of a program state. The information needs regarding the cause of a program state may be specific to debugging. This is a big issue particularly in studies where developers are fixing bugs, whereas this does not seem to be that important when developers are engaged in other activities.

Understanding what the code is supposed to do. Some studies find this to be an important information need while others do not. We do not know the reason for this variance.

We list the categories that we excluded in Table A.15. Most of the excluded categories were too technology specific, did not have enough support from other studies, or were impossible to investigate.

5 Threats to Validity

Construct Validity. We assessed how important developers' information needs are by evaluating how important these needs are in primary research papers. Thus our results are affected by the quality of the primary studies.

We excluded studies that are older than 15 years for $\mathbf{RQ}_2$. However, information needs might have changed within this time period as developers embraced better tools and practices. Nevertheless, we believe that the information needs we found are still important today.

We also assume that useful and important categories of information needs are pointed out in the abstract or the conclusion of these papers, which is rather a simplistic heuristic.

Internal Validity. We used *Google Scholar* for the systematic search instead of using multiple search engines. We mitigated this threat by also searching for the papers that are referenced by the included papers as this is independent of the search engine.

We believe that the greatest threat to validity is that all of the steps in our systematic literature review were carried out by only one person. The guidelines that we followed assume that the systematic search would be done by a large group of researchers. This might bias our result and make them more prone to clerical errors.

Given the large number of results that we searched (8 147 results), many papers had to be excluded just based on the title or the abstract, without looking at the content more closely. This threat has been partially mitigated, because the matched papers that we evaluated contained many duplicates. However, this might have biased the results towards including more papers with many citations than ones that are not cited as often. We believe that although we might have missed a few papers, this would not have changed the results of $\mathbf{RQ}_1$ significantly. For $\mathbf{RQ}_2$, more papers included could lead to more categories found. However, the goal for $\mathbf{RQ}_2$ was not to find an exhaustive list of categories. The goal was to find important and useful categories, which we believe we found.

$\mathbf{RQ}_2$ is at higher risk of researcher bias, as synthesizing the data across papers is challenging. Additionally, the methodology used for finding important and useful categories of information needs was only fully defined after we answered $\mathbf{RQ}_1$. Thus it is subject to post hoc hypotheses.

External Validity. Many of the studies were conducted at larger companies and with developers experienced enough to be hired by these companies. Thus the information needs that we found might not generalize to novices or small projects.

6
Discussion and Future Work

Finding important information needs turned out to be much more challenging than finding how researchers quantify information needs. How data from different papers can be compared correctly is a question researchers struggle with. We do not know what a better way to combine this data might be. We believe that having multiple researchers synthesize this data independently of each other would improve the validity of our results.

During this project, we learned from the studies that the types of questions that developers ask are intertwined with the tools and strategies that they use. As noted in several papers, many developers seem to make ineffective use of these tools and use futile strategies. As an example, Roehm *et al.* observed a participant who wanted to find locations where a specific constant is used. Thus he changed the name of the constant and then inspected the compiler warnings. He was unaware of the feature that finds all references, although he had six years of experience using Eclipse [36]. During our search, we noticed that most papers propose new tools whereas only a few papers propose new practices or how developers could learn more effective strategies. Many papers note that developers made inefficient use of the tools and used ineffective strategies. If this is true, then it begs the question: Is research on new tools more beneficial at helping developers than research on new practices that enable them to choose, learn, and share more effective strategies?

7 Conclusion

We studied how researchers quantify developers' information needs (**RQ**$_1$) and which important categories of information needs have empirical support from a wide range of studies (**RQ**$_2$).

We found only three observational studies that assess the importance of a question using a measure other than how frequently it was asked. Most studies use self-reporting in surveys or an artifact of the development process such as Stack Overflow to study developers' information needs. Both might have problems in that we do not know how closely self-reporting matches with reality and we do not know if searching for answers on Stack Overflow impacts the productivity of developers significantly in relation to other information needs. We find that information about rationale (e.g., *why is this implemented this way?)*, awareness (e.g., *what have my coworkers been doing?)*, and the implications of a code change (e.g., *what might break?)* are some of the most important information types that developers need to make progress.

We described the state of the research on quantifying developers' information needs. We also discussed the challenges that we faced when synthesizing the most important information needs from different studies. We did not find a complete list of categories of information needs, nor is it possible to find a complete list, as researchers can constantly develop new categories of information needs that they deem useful. We believe that the categories that we found are both useful and important, as many studies support them. Furthermore, we encourage researchers to align the tools and practices used to record, communicate, and obtain information with developers' needs.

Thus we think that our systematic literature review provides useful insights for researchers studying developers' information needs, as well as for tool developers working on tools and practices to make software development more productive.

A

Anleitung zum wissenschaftlichen Arbeiten

In the appendix, we list the details on:

- Which data we extracted from the studies (section A.1)
- Our search terms and search parameters (section A.2)
- The papers that we found after each iteration of our systematic search (section A.3)
- The papers included in our study (Table A.5)
- The papers that we included after applying our additional exclusion criteria for **RQ**$_2$ (Table A.8)
- The list of excluded studies (Table A.11)
- The list of excluded categories (Table A.15)

A.1 Data extraction

We extract from the studies:

- **Study method.** Which method was used for studying developers' information needs, *e.g.*, did they use observations, surveys, interviews, or some other method?
- **Data source.** Which data source they used, *e.g.*, people, mailing lists, Stack Overflow or some other source?
- **If the study is quantitative.** Does the study include quantitative data or was the focus of the study qualitative?
- **How the study quantifies developers information needs.** *E.g.,* does the study measure how frequently a question was asked, how long it took to answer a question, or do they use some other measure?
- **Title.** The title of the paper.
- **Author.** The primary author of the paper.
- **Year published.** The year that the study was published.
- **Institution.** Which institution the primary author is affiliated with (*e.g.*, University of Bern).
- **Country.** The country of the institution (*e.g.*, Switzerland).
- **References.** The number of studies that the paper references.
- **Citations.** The number of studies that cite the paper. We used Google Scholar for citation tracking.
- **Venue.** *E.g.,* which conference, workshop, or journal was the paper published in (*e.g.*, ICSE).
- **Publication type.** *E.g.,* was the paper published in a conference, workshop or in a journal?
- **Survey type.** If the paper used surveys, did they use closed-form or open-ended surveys?

- **Study setting.** *E.g.,* was the study done in the lab or was it a field study in industry?

A.2 Search terms

We conducted the systematic search in December of 2018 using Google Scholar. We used the default search options, which did not limit our search to a specific time range and we sorted the results by relevance. We limited each search to the first 100 results, as we did not find any more relevant papers after this threshold. We used the following search terms:

Search term
software developer “information needs”
developer questions
program comprehension “information needs”
program comprehension questions
stack overflow questions
developer mailing list questions
developer information seeking mailing list

A.3 Systematic search

Initial Search

Title	Authors	References	Citations
Information Needs in Collocated Software Development Teams	Ko *et al.*	20	447
Information needs for software development analytics	Buse *et al.*	44	148
Information needs in bug reports: improving cooperation between developers and users	Breu *et al.*	28	178
A quantitative analysis of developer information needs in software ecosystems	Haenni *et al.*	15	20
Categorizing Developer Information Needs in Software Ecosystems	Haenni *et al.*	11	17
Using information fragments to answer the questions developers ask	Fritz *et al.*	16	153
Codebook: discovering and exploiting relationships in software repositories	Begel *et al.*	32	203
How do programmers ask and answer questions on the web?: Nier track	Treude *et al.*	12	266
How do professional developers comprehend software?	Roehm *et al.*	24	130
How Do Software Engineers Understand Code Changes? - An Exploratory Study in Industry	Tao *et al.*	59	102
Analyze this! 145 questions for data scientists in software engineering	Begel *et al.*	52	107
Developers ask reachability questions	LaToza *et al.*	22	126
An empirical study on developer interactions in Stack Overflow	Wang *et al.*	19	65
Maintaining mental models: a study of developer work habits	LaToza *et al.*	14	522
On the comprehension of program comprehension	Maleej *et al.*	68	92
Comprehension Processes During Large Scale Maintenance	Von Mayrhauser *et al.*	13	132

Table A.1: The list of studies that we found after each iteration of our systematic search. This table is continued on the next page

Title	Authors	References	Citations
Program Understanding Behavior During Adaptation of Large Scale Software	Von Mayrhauser *et al.*	23	38
Program understanding behaviour during enhancement of large-scale software	Von Mayrhauser *et al.*	21	90
Program Understanding Needs During Corrective Maintenance of Large Scale Software	Von Mayrhauser *et al.*	16	21
Modelling the Information-Seeking Behaviour of Programmers D An Empirical Approach	O'Brien *et al.*	32	22
Questions programmers ask during software evolution tasks	Sillito *et al.*	25	256
Why, when, and what: analyzing stack overflow questions by topic, type, and code	Allamanis *et al.*	5	68
What are developers talking about? An analysis of topics and trends in Stack Overflow	Barua *et al.*	55	234
What are mobile developers asking about? A large scale study using stack overflow	Rosen *et al.*	44	59
An Exploratory Analysis of Mobile Development Issues using Stack Overflow	Linares-Vasquez *et al.*	12	44
Mining Questions about Software Energy Consumption	Pinto *et al.*	35	106
Mining Questions Asked by Web Developers	Bajaj *et al.*	30	62
Using and Asking: APIs Used in the Android Market and Asked About in Stack Overflow	Kavaler *et al.*	22	29
What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts	Yang *et al.*	37	17
How the R Community Creates and Curates Knowledge: A Comparative Study of Stack Overflow and Mailing Lists	Zagalsky *et al.*	23	15
A Manual Categorization of Android App Development Issues on Stack Overflow	Beyer *et al.*	9	19
Which Non-functional Requirements do Developers Focus on?	Zou *et al.*	6	11
An empirically-based characterization and quantification of information seeking through mailing lists during Open Source developers' software evolution	Sharif *et al.*	92	9
Observation of Open Source Programmers' Information Seeking	Sharif *et al.*	6	4
Developing Schema for Open Source Programmers' Information-Seeking	Sharif *et al.*	36	8

Table A.2: This is a continuation of the table on the previous page

First Iteration

Title	Authors	References	Citations
Asking and answering questions during a programming change task	Sillito *et al.*	76	249
Hard-to-Answer Questions about Code	LaToza *et al.*	28	93
Asking and Answering Questions about Unfamiliar APIs: An Exploratory Study	Duala-Ekoko *et al.*	26	70
Questions developers ask while diagnosing potential security vulnerabilities with static analysis	Smith *et al.*	41	31
Questions about Object Structure during Coding Activities	Abi-Antoun *et al.*	27	13
An Exploratory Study of Awareness Interests about Software Modifications	Kim	19	11
Asking and Answering Questions during a Programming Change Task in Pharo Language	Kubelka *et al.*	9	9
Searching Across Paths	LaToza *et al.*	11	3
Stakeholders' Information Needs for Artifacts and their Dependencies in a Real World Context	Müller *et al.*	35	9
Reporting Usability Defects: Do Reporters Report What Software Developers Need?	Yusop *et al.*	27	14
What Questions Developers Ask During Software Evolution? An Academic Perspective	Novais *et al.*	13	3
Detecting API Usage Obstacles: A Study of iOS and Android Developer Questions	Wang *et al.*	17	40
Mining Testing Questions on Stack Overflow	Kochhar	16	2
Classifying Stack Overflow Posts on API Issues	Ahasanuzzama *et al.*	36	1
What do Developers want? An Advisor approach for Developer Priorities	Sharma *et al.*	7	1
What Help Do Developers Seek, When and How?	Li *et al.*	31	37
From Code Understanding Needs to reverse Engineering Tool Capabilities	Von Mayrhauser *et al.*	19	102
What Can Programmer Questions Tell Us About Frameworks?	Hou *et al.*	12	33
Characterizing Programmers' Information-Seeking during Software Evolution	Buckley *et al.*	30	8
Further Observation of Open Source Programmers' Information Seeking	Sharif *et al.*	34	4

Table A.3: This is a continuation of the table on the previous page

Title	Authors	References	Citations
A field study of how developers locate features in source code	Damevski *et al.*	22	16
A Study on the Most Popular Questions About Concurrent Programming	Pinto *et al.*	24	5
Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis	Zou *et al.*	65	6
Empirical Analysis of the Logging Questions on the Stack Overflow Website	Gujral *et al.*	24	1
What are Software Engineers asking about Android Testing on Stack Overflow?	Villanes *et al.*	27	2
What Questions Do Programmers Ask About Configuration as Code?	Rahman *et al.*	23	2
What Concerns Do Client Developers Have When Using Web APIs? An Empirical Study of Developer Forums and Stack Overflow	Venkatesh *et al.*	29	9
Open Source Programmers' Information Seeking During Software Maintenance	Sharif *et al.*	43	1
Archetypal Source Code Searches: A Survey of Software Developers and Maintainers	Sim *et al.*	14	122
What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow	Ahmed *et al.*	34	0
Cognitive Processes in Program Comprehension	Letovsky *et al.*	21	437

Second Iteration

Title	Authors	References	Citations
How Developers Search for Code: A Case Study	Sadowski *et al.*	31	50
What Makes APIs Hard to Learn? Answers from Developers	Robillard	9	239
What do developers search for on the web?	Xia *et al.*	34	18

Third Iteration

Title	Authors	References	Citations
Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions	Hou *et al.*	28	66

Table A.4: This is a continuation of the table on the previous page

A.4 List of papers

Title	Authors	Year	Reference
Empirical Analysis of the Logging Questions on the Stack Overflow Website	Gujral *et al.*	2018	[14]
What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow	Ahmed *et al.*	2018	[2]
What Questions Do Programmers Ask About Configuration as Code?	Rahman *et al.*	2018	[34]
What are Software Engineers asking about Android Testing on Stack Overflow?	Villanes *et al.*	2017	[49]
Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis	Zou *et al.*	2017	[60]
What do developers search for on the web?	Xia *et al.*	2017	[56]
What do Developers want? An Advisor approach for Developer Priorities	Sharma *et al.*	2017	[41]
Reporting Usability Defects: Do Reporters Report What Software Developers Need?	Yusop *et al.*	2016	[58]
What Concerns Do Client Developers Have When Using Web APIs? An Empirical Study of Developer Forums and Stack Overflow	Venkatesh *et al.*	2016	[48]
How the R Community Creates and Curates Knowledge: A Comparative Study of Stack Overflow and Mailing Lists	Zagalsky *et al.*	2016	[59]
What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts	Yang *et al.*	2016	[57]
Mining Testing Questions on Stack Overflow	Kochhar	2016	[21]
What are mobile developers asking about? A large scale study using stack overflow	Rosen *et al.*	2016	[37]
A Study on the Most Popular Questions About Concurrent Programming	Pinto *et al.*	2015	[33]
An empirically-based characterization and quantification of information seeking through mailing lists during Open Source developers' software evolution	Sharif *et al.*	2015	[39]
Questions developers ask while diagnosing potential security vulnerabilities with static analysis	Smith *et al.*	2015	[45]

Table A.5: The list of primary studies that we identified

Title	Authors	Year	Reference
How Developers Search for Code: A Case Study	Sadowski *et al.*	2015	[38]
A Manual Categorization of Android App Development Issues on Stack Overflow	Beyer *et al.*	2014	[8]
Mining Questions about Software Energy Consumption	Pinto *et al.*	2014	[32]
What Questions Developers Ask During Software Evolution? An Academic Perspective	Novais *et al.*	2014	[31]
Mining Questions Asked by Web Developers	Bajaj *et al.*	2014	[4]
What are developers talking about? An analysis of topics and trends in Stack Overflow	Barua *et al.*	2014	[5]
Asking and Answering Questions during a Programming Change Task in Pharo Language	Kubelka *et al.*	2014	[23]
A quantitative analysis of developer information needs in software ecosystems	Haenni *et al.*	2014	[16]
Analyze this! 145 questions for data scientists in software engineering	Begel *et al.*	2014	[7]
Why, when, and what: analyzing stack overflow questions by topic, type, and code	Allamanis *et al.*	2013	[3]
An empirical study on developer interactions in Stack Overflow	Wang *et al.*	2013	[55]
Stakeholders' Information Needs for Artifacts and their Dependencies in a Real World Context	Müller *et al.*	2013	[30]
Categorizing Developer Information Needs in Software Ecosystems	Haenni *et al.*	2013	[15]
An Exploratory Analysis of Mobile Development Issues using Stack Overflow	Linares-Vásquez *et al.*	2013	[28]
Asking and Answering Questions about Unfamiliar APIs: An Exploratory Study	Duala-Ekoko *et al.*	2012	[12]
How Do Software Engineers Understand Code Changes? - An Exploratory Study in Industry	Tao *et al.*	2012	[46]
On the comprehension of program comprehension	Maalej *et al.*	2012	[29]
How do professional developers comprehend software?	Roehm *et al.*	2012	[36]
Information needs for software development analytics	Buse *et al.*	2012	[11]
How do programmers ask and answer questions on the web?: Nier track	Treude *et al.*	2011	[47]
Open Source Programmers' Information Seeking During Software Maintenance	Sharif *et al.*	2011	[22]
An Exploratory Study of Awareness Interests about Software Modifications	Kim	2011	[19]
Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions	Hou *et al.*	2011	[17]
Using information fragments to answer the questions developers ask	Fritz *et al.*	2010	[13]

Table A.6: This is a continuation of the table on the previous page

Title	Authors	Year	Reference
Information needs in bug reports: improving cooperation between developers and users	Breu *et al.*	2010	[9]
Developers ask reachability questions	LaToza *et al.*	2010	[24]
Hard-to-Answer Questions about Code	LaToza *et al.*	2010	[25]
Questions about Object Structure during Coding Activities	Abi-Antoun *et al.*	2010	[1]
Codebook: discovering and exploiting relationships in software repositories	Begel *et al.*	2010	[6]
What Makes APIs Hard to Learn? Answers from Developers	Robillard	2009	[35]
Asking and answering questions during a programming change task	Sillito *et al.*	2008	[43]
Developing Schema for Open Source Programmers' Information-Seeking	Sharif *et al.*	2008	[40]
Information Needs in Collocated Software Development Teams	Ko *et al.*	2007	[20]
Questions programmers ask during software evolution tasks	Sillito *et al.*	2006	[42]
Maintaining mental models: a study of developer work habits	LaToza *et al.*	2006	[26]
What Can Programmer Questions Tell Us About Frameworks?	Hou *et al.*	2005	[18]
Characterizing Programmers' Information-Seeking during Software Evolution	Buckley *et al.*	2004	[10]
Archetypal Source Code Searches: A Survey of Software Developers and Maintainers	Sim *et al.*	1998	[44]
Program Understanding Behavior During Adaptation of Large Scale Software	Von Mayrhauser *et al.*	1998	[53]
Program Understanding Needs During Corrective Maintenance of Large Scale Software	Von Mayrhauser *et al.*	1997	[52]
Program understanding behaviour during enhancement of large-scale software	Von Mayrhauser *et al.*	1997	[54]
Comprehension Processes During Large Scale Maintenance	Von Mayrhauser *et al.*	1994	[51]
From Code Understanding Needs to reverse Engineering Tool Capabilities	Von Mayrhauser *et al.*	1993	[50]
Cognitive Processes in Program Comprehension	Letowsky	1987	[27]

Table A.7: This is a continuation of the table on the previous page

A.5 List of papers relevant to RQ$_2$

Title	Authors	Year	Reference
Empirical Analysis of the Logging Questions on the Stack Overflow Website	Gujral *et al.*	2018	[14]
What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow	Ahmed *et al.*	2018	[2]
What Questions Do Programmers Ask About Configuration as Code?	Rahman *et al.*	2018	[34]
What are Software Engineers asking about Android Testing on Stack Overflow?	Villanes *et al.*	2017	[49]
Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis	Zou *et al.*	2017	[60]
What do developers search for on the web?	Xia *et al.*	2017	[56]
What do Developers want? An Advisor approach for Developer Priorities	Sharma *et al.*	2017	[41]
Reporting Usability Defects: Do Reporters Report What Software Developers Need?	Yusop *et al.*	2016	[58]
What Concerns Do Client Developers Have When Using Web APIs? An Empirical Study of Developer Forums and Stack Overflow	Venkatesh *et al.*	2016	[48]
How the R Community Creates and Curates Knowledge: A Comparative Study of Stack Overflow and Mailing Lists	Zagalsky *et al.*	2016	[59]
What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts	Yang *et al.*	2016	[57]
Mining Testing Questions on Stack Overflow	Kochhar	2016	[21]
What are mobile developers asking about? A large scale study using stack overflow	Rosen *et al.*	2016	[37]
A Study on the Most Popular Questions About Concurrent Programming	Pinto *et al.*	2015	[33]
An empirically-based characterization and quantification of information seeking through mailing lists during Open Source developers' software evolution	Sharif *et al.*	2015	[39]
How Developers Search for Code: A Case Study	Sadowski *et al.*	2015	[38]

Table A.8: The list of primary studies that we identified after applying our additional inclusion and exclusion criteria of **RQ$_2$**. This table is continued on the next page

Title	Authors	Year	Reference
A Manual Categorization of Android App Development Issues on Stack Overflow	Beyer *et al.*	2014	[8]
Mining Questions about Software Energy Consumption	Pinto *et al.*	2014	[32]
What Questions Developers Ask During Software Evolution? An Academic Perspective	Novais *et al.*	2014	[31]
Mining Questions Asked by Web Developers	Bajaj *et al.*	2014	[4]
What are developers talking about? An analysis of topics and trends in Stack Overflow	Barua *et al.*	2014	[5]
A quantitative analysis of developer information needs in software ecosystems	Haenni *et al.*	2014	[16]
Analyze this! 145 questions for data scientists in software engineering	Begel *et al.*	2014	[7]
An empirical study on developer interactions in Stack Overflow	Wang *et al.*	2013	[55]
Stakeholders' Information Needs for Artifacts and their Dependencies in a Real World Context	Müller *et al.*	2013	[30]
An Exploratory Analysis of Mobile Development Issues using Stack Overflow	Linares-Vásquez *et al.*	2013	[28]
Asking and Answering Questions about Unfamiliar APIs: An Exploratory Study	Duala-Ekoko *et al.*	2012	[12]
How Do Software Engineers Understand Code Changes? - An Exploratory Study in Industry	Tao *et al.*	2012	[46]
On the comprehension of program comprehension	Maalej *et al.*	2012	[29]
Information needs for software development analytics	Buse *et al.*	2012	[11]
How do programmers ask and answer questions on the web?: Nier track	Treude *et al.*	2011	[47]
Open Source Programmers' Information Seeking During Software Maintenance	Sharif *et al.*	2011	[22]
An Exploratory Study of Awareness Interests about Software Modifications	Kim	2011	[19]
Information needs in bug reports: improving cooperation between developers and users	Breu *et al.*	2010	[9]
Developers ask reachability questions	LaToza *et al.*	2010	[24]
Hard-to-Answer Questions about Code	LaToza *et al.*	2010	[25]
Codebook: discovering and exploiting relationships in software repositories	Begel *et al.*	2010	[6]
What Makes APIs Hard to Learn? Answers from Developers	Robillard	2009	[35]

Table A.9: This is a continuation of the table on the previous page

Title	Authors	Year	Reference
Developing Schema for Open Source Programmers' Information-Seeking	Sharif *et al.*	2008	[40]
Information Needs in Collocated Software Development Teams	Ko *et al.*	2007	[20]
Maintaining mental models: a study of developer work habits	LaToza *et al.*	2006	[26]
What Can Programmer Questions Tell Us About Frame-works?	Hou *et al.*	2005	[18]
Characterizing Programmers' Information-Seeking during Software Evolution	Buckley *et al.*	2004	[10]

Table A.10: This is a continuation of the table on the previous page

A.6 Excluded studies

Title	Reason for exclusion
Modelling the Information-Seeking Behaviour of Programmers' An Empirical Approach	Focuses on information seeking behavior and not on information needs.
Searching Across Paths	The questions are taken from previous studies. This study just describes how developers translate many common questions into searches across paths.
Classifying Stack Overflow Posts on API Issues	Focuses on constructing a classifier rather than categorizing questions to find developers' information needs.
A field study of how developers locate features in source code	Focuses on how developers locate features in source code instead of the information needs of developers.
What Help Do Developers Seek, When and How?	Although the researchers state in the abstract that the study focuses on information needs, it does not. It instead focuses on which information sources developers use.
Which Non-functional Requirements Do Developers Focus On? An Empirical Study on Stack Overflow Using Topic Analysis	Contains duplicate data from another study by the same authors.
Further Observation of Open Source Programmers' Information Seeking	Contains duplicate data from another study by the same authors.
Observation of Open Source Programmers' Information Seeking	Contains duplicate data from another study by the same authors.
How effective developers investigate source code	Focuses on information seeking behaviour rather than what kind of information developers seek.
What Help Do Developers Seek, When and How?	Focuses on the help seeking behavior and not the information needs of the developers.
Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior	Proposes a new tool instead of focusing on developer' information needs.
Information Needs for Validating Evolving Software Systems: An Exploratory Study at Google	Focuses on reliability engineers instead of software developers.
How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool	Contains duplicate data from another study by the same authors.

Table A.11: The list of excluded papers. This table is continued on the next page

Title
Frequently Asked Questions in Bug Reports
Get Me Here: Using Verification Tools to Answer Developer Questions
Deep intellisense: a tool for rehydrating evaporated information
Replaying Past Changes in Multi-developer Projects
Program Comprehension as Fact Finding
Software Evolution Comprehension: Replay to the Rescue
Harnessing Stack Overflow for the IDE
Communication in Open Source Software Development Mailing Lists
Group Awareness in Distributed Software Development
Studying the Use of Developer IRC Meetings in Open Source Projects
An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks
Studying the use of developer IRC meetings in open source projects
Defining Open Source Software Project Success
A review of awareness in distributed collaborative software engineering
An Empirical Study of API Usability
Extracting and answering why and why not questions about Java program output
Programmer information needs after memory failure
Replaying Past Changes in Multi-developer Projects
Information Needs for Integration Decisions in the Release Process of Large-Scale Parallel Development
Debugging Revisited
Answering software evolution questions: An empirical evaluation
Open Source Programmers' Information Seeking
What Questions do Requirements Engineers Ask?
Information Needs for Integration Decisions in the Release Process of Large-Scale Parallel Development
Information Needs in Contemporary Code Review
Information Needs in Software Ecosystems Development
Finding Relevant Answers in Software Forums
Information Needs in Software Ecosystems Development
Information Needs for Integration Decisions in the Release Process of Large-Scale Parallel Development
Answering Conceptual Queries with Ferret
Navigating and Querying Code Without Getting Lost
What Questions do Requirements Engineers Ask?

Table A.12: This is a continuation of the table on the previous page

Title
Communicative Intention in Code Review Questions
An Efficient Approach for Providing Rationale of Method Change for Object Oriented Programming
What Do Developers Use the Crowd For?
A survey on mining stack overflow: question and answering (Q&A) community
Group Awareness in Distributed Software Development
On the Importance of Understanding the Strategies that Developers Use
Can you tell me if it smells?: A study on how developers discuss code smells and anti-patterns in Stack Overflow
The information-seeking practices of engineers: searching for documents as well as for people
Coordination in Large-Scale Software Development: Helpful and Unhelpful Behaviors
Ask the Crowd: Scaffolding Coordination and Knowledge Sharing in Microtask Programming
From Program Comprehension to Tool Requirements for an Industrial Environment
Foraging and Navigations, Fundamentally: Developers' Predictions of Value and Cost
Breakdowns and processes during the early activities of software design by professionals
Expert problem solving strategies for program comprehension
Stimulus structures and mental representations in expert comprehension of computer programs
Knowledge and Processes in the Comprehension of Computer Programs
Program Understanding - A Survey
Patterns of developers behaviour: A 1000-hour industrial study
On the Role of Program Comprehension in Embedded Systems
Empirically Refining a Model of Programmers' Information-Seeking Behavior during Software Maintenance
What do I need to know and where do I find it? - An Empirical Investigation of Information Needs in Enterprise Integration Projects
Managing Software Change Tasks: An Exploratory Study
Portfolio: Finding Relevant Functions and Their Usages
Querying source code with natural language
Working with Search Results
Understanding and Classifying the Quality of Technical Forum Questions
Classifying Stack Overflow Questions Based on Blooms Taxonomy
Software History under the Lens: A Study on Why and How Developers Examine It
Making Your Programming Questions Be Answered Quickly: A Content Oriented Study to Technical Q&A Forum
How Do Developers Discuss Rationale?
An Empirical Study of API Usability

Table A.13: This is a continuation of the table on the previous page

Title
Content Classification of Development Emails
Using Conditional Random Fields to Extract Contexts and Answers of Questions from Online Forums
Extracting Problem and Resolution Information from Online Discussion Forums
Finding Relevant Answers in Software Forums
A Survey on the Software Maintenance Process
Information needs of developers for program comprehension during software maintenance tasks
Concurrency at Microsoft: An exploratory survey
Identifying the information needs and sources of software practitioners
What information do software engineering practitioners need?
Evaluating Forum Discussions to Inform the Design of an API Critic
FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams

Table A.14: This is a continuation of the table on the previous page

A.7 Excluded categories

Category	Reason for exclusion
Why something is failing	We only found support for this in studies where developers are debugging.
How to fix a bug	If a given question is asked in support of fixing a bug cannot be extracted from the questions in most studies.
Understanding the cause of the problem	This is not assessed in most studies.
Exceptions/error messages	There is not much support from studies that do not use Stack Overflow that this is an important issue.
Unexpected behavior	This is hard to investigate. We cannot extract if a given question was asked because of unexpected behavior.
What the expected results are	This is not assessed in most studies.
Which suitable third-party libraries/services to use	We did not find enough support for this.
Questions about the tools and technology employed in the project	We did not find enough support for this.
Best industrial practices	We did not find enough support for this.
How-to questions	It is not clear from the studies where the boundaries of this category are, and thus it is impossible to investigate.
Why questions	It is not clear from the studies where the boundaries of this category are, and thus it is impossible to investigate.
Where questions	It is not clear from the studies where the boundaries of this category are, and thus it is impossible to investigate.
What questions	It is not clear from the studies where the boundaries of this category are, and thus it is impossible to investigate.

Table A.15: The list of excluded categories. This table is continued on the next page

Category	Reason for exclusion
The value of a fix	We did not find enough support for this.
Unknown terminologies	We did not find enough support for this.
Where code is located	We did not find enough support for this.
“Is it possible . . . ?” questions	It is not clear from the studies where the boundaries of this category are, and thus it is impossible to investigate.
“What is the problem. . . ?” questions	It is not clear from the studies where the boundaries of this category are, and thus it is impossible to investigate.
Reachability questions	LaToza *et al.* define reachability questions as *“a search across feasible paths through a program for target statements matching search criteria”*. However, it is impossible for us to investigate which high-level questions in other studies could be answered by asking several low-level reachability questions.

Table A.16: This is a continuation of the table on the previous page

Category
OOP
Classes
Java
Python
C#
C
User interface
Stack trace
Large code snippets
Web documents
Miscellaneous
Systems' implementations
HTML5
Javascript
Practical problems
Basic concepts of concurrent programming
Syntax errors
Provisioning instances
Assessing Puppet's feasibility to accomplish certain tasks
Installation
Security
Data separation
Template related questions
Compatibility issues
Crash reports
Database connection
General knowledge
App distribution
Mobile tools
User interface development
Usability problems
Reliability problems
Maintainability problems
Reviewing code
Correctness of concurrent programs
Performance of concurrent programs
Thread safety
Questions about database management systems
Public datasets to test newly developed algorithms
Database optimization solutions

Table A.17: This is a continuation of the table on the previous page

Systematic Literature Review Papers

[1] Marwan Abi-Antoun, Nariman Ammar, and Thomas LaToza. Questions about object structure during coding activities. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, pages 64–71. ACM, 2010.

[2] Syed Ahmed and Mehdi Bagherzadeh. What do concurrency developers ask about?: a large-scale study using Stack Overflow. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 30. ACM, 2018.

[3] Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing Stack Overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 53–56. IEEE Press, 2013.

[4] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 112–121. ACM, 2014.

[5] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[6] Andrew Begel, Yit Phang Khoo, and Thomas Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 1, pages 125–134. IEEE, 2010.

[7] Andrew Begel and Thomas Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, pages 12–23. ACM, 2014.

[8] Stefanie Beyer and Martin Pinzger. A manual categorization of Android app development issues on Stack Overflow. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 531–535. IEEE, 2014.

[9] Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310. ACM, 2010.

[10] Jim Buckley, Christopher Exton, and Judith Good. Characterizing programmers' information-seeking during software evolution. In *Software Technology and Engineering Practice, 2004. STEP 2004. The 12th International Workshop on*, pages 7–pp. IEEE, 2005.

[11] Raymond PL Buse and Thomas Zimmermann. Information needs for software development analytics. In *Proceedings of the 34th international conference on software engineering*, pages 987–996. IEEE Press, 2012.

[12] Ekwa Duala-Ekoko and Martin P Robillard. Asking and answering questions about unfamiliar APIs: An exploratory study. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 266–276. IEEE, 2012.

[13] Thomas Fritz and Gail C Murphy. Using information fragments to answer the questions developers ask. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 175–184. ACM, 2010.

[14] Harshit Gujral, Abhinav Sharma, Sangeeta Lal, Amanpreet Kaur, A Kumar, and Ashish Sureka. Empirical analysis of the logging questions on the Stack Overflow website. In *2018 Conference On Software Engineering & Data Sciences (CoSEDS)(in-press)*, 2018.

[15] Nicole Haenni, Mircea Lungu, Niko Schwarz, and Oscar Nierstrasz. Categorizing developer information needs in software ecosystems. In *Proceedings of the 2013 International Workshop on Ecosystem Architectures*, pages 1–5. ACM, 2013.

[16] Nicole Haenni, Mircea Lungu, Niko Schwarz, and Oscar Nierstrasz. A quantitative analysis of developer information needs in software ecosystems. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, page 12. ACM, 2014.

[17] Daqing Hou and Lin Li. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, pages 91–100. IEEE, 2011.

[18] Daqing Hou, Kenny Wong, and H James Hoover. What can programmer questions tell us about frameworks? In *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*, pages 87–96. IEEE, 2005.

[19] Miryung Kim. An exploratory study of awareness interests about software modifications. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 80–83. ACM, 2011.

[20] Andrew J Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th international conference on Software Engineering*, pages 344–353. IEEE Computer Society, 2007.

[21] Pavneet Singh Kochhar. Mining testing questions on Stack Overflow. In *Proceedings of the 5th International Workshop on Software Mining*, pages 32–38. ACM, 2016.

[22] Fakulti S Komputer and T Maklumat. Open source programmers' information seeking during software maintenance. *Journal of Computer Science*, 7(7):1060–1071, 2011.

[23] Juraj Kubelka, Alexandre Bergel, and Romain Robbes. Asking and answering questions during a programming change task in Pharo language. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 1–11. ACM, 2014.

[24] Thomas D LaToza and Brad A Myers. Developers ask reachability questions. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 185–194. ACM, 2010.

[25] Thomas D LaToza and Brad A Myers. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*, page 8. ACM, 2010.

[26] Thomas D LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*, pages 492–501. ACM, 2006.

[27] Stanley Letovsky. Cognitive processes in program comprehension. *Journal of Systems and software*, 7(4):325–339, 1987.

[28] Mario Linares-Vásquez, Bogdan Dit, and Denys Poshyvanyk. An exploratory analysis of mobile development issues using Stack Overflow. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 93–96. IEEE, 2013.

[29] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):31, 2014.

[30] Sebastian C Müller and Thomas Fritz. Stakeholders' information needs for artifacts and their dependencies in a real world context. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 290–299. IEEE, 2013.

[31] Renato Novais, Creidiane Brito, and Manoel Mendonça. What questions developers ask during software evolution? an academic perspective. In *2nd Workshop on Software Visualization, Evolution, and Maintenance (VEM 2014)*, pages 14–21, 2014.

[32] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31. ACM, 2014.

[33] Gustavo Pinto, Weslley Torres, and Fernando Castor. 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*, pages 39–46. ACM, 2015.

[34] Akond Rahman, Asif Partho, Patrick Morrison, and Laurie Williams. What questions do programmers ask about configuration as code? In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, pages 16–22. ACM, 2018.

[35] Martin P Robillard. What makes APIs hard to learn? answers from developers. *IEEE software*, 26(6):27–34, 2009.

[36] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, pages 255–265. IEEE Press, 2012.

[37] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using Stack Overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[38] Caitlin Sadowski, Kathryn T Stolee, and Sebastian Elbaum. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 191–201. ACM, 2015.

[39] Khaironi Y Sharif, Michael English, Nour Ali, Chris Exton, JJ Collins, and Jim Buckley. An empirically-based characterization and quantification of information seeking through mailing lists during open source developers' software evolution. *Information and Software Technology*, 57:77–94, 2015.

[40] Khaironi Yatim Sharif and Jim Buckley. Developing schema for open source programmers' information-seeking. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 1, pages 1–9. IEEE, 2008.

[41] Vibhu Saujanya Sharma, Rohit Mehra, and Vikrant Kaulgud. What do developers want?: an advisor approach for developer priorities. In *Proceedings of the 10th In-*

ternational Workshop on Cooperative and Human Aspects of Software Engineering, pages 78–81. IEEE Press, 2017.

[42] Jonathan Sillito, Gail C Murphy, and Kris De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 23–34. ACM, 2006.

[43] Jonathan Sillito, Gail C Murphy, and Kris De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.

[44] Susan Elliott Sim, Charles LA Clarke, and Richard C Holt. Archetypal source code searches: A survey of software developers and maintainers. In *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*, pages 180–187. IEEE, 1998.

[45] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 248–259. ACM, 2015.

[46] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. How do software engineers understand code changes?: an exploratory study in industry. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 51. ACM, 2012.

[47] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web?: Nier track. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 804–807. IEEE, 2011.

[48] Pradeep K Venkatesh, Shaohua Wang, Feng Zhang, Ying Zou, and A Hassan. What concerns do client developers have when using web APIs? an empirical study of developer forums and Stack Overflow. In *IEEE international conference on web services (ICWS)*, pages 131–138, 2016.

[49] Isabel K Villanes, Silvia M Ascate, Josias Gomes, and Arilo Claudio Dias-Neto. What are software engineers asking about Android testing on Stack Overflow? In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, pages 104–113. ACM, 2017.

[50] Anneliese von Mayrhauser and A Marie Vans. From code understanding needs to reverse engineering tool capabilities. In *Computer-Aided Software Engineering, 1993. CASE'93., Proceeding of the Sixth International Workshop on*, pages 230–239. IEEE, 1993.

[51] Anneliese von Mayrhauser and A Marie Vans. Comprehension processes during large scale maintenance. In *Proceedings of the 16th international conference on Software engineering*, pages 39–48. IEEE Computer Society Press, 1994.

[52] Anneliese von Mayrhauser and A Marie Vans. Program understanding needs during corrective maintenance of large scale software. In *Computer Software and Applications Conference, 1997. COMPSAC'97. Proceedings., The Twenty-First Annual International*, pages 630–637. IEEE, 1997.

[53] Anneliese Von Mayrhauser and A Marie Vans. Program understanding behavior during adaptation of large scale software. In *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*, pages 164–172. IEEE, 1998.

[54] Anneliese Von Mayrhauser, A Marie Vans, and Adele E Howe. Program understanding behaviour during enhancement of large-scale software. *Journal of Software Maintenance: Research and Practice*, 9(5):299–327, 1997.

[55] Shaowei Wang, David Lo, and Lingxiao Jiang. An empirical study on developer interactions in StackOverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1019–1024. ACM, 2013.

[56] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.

[57] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of Stack Overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.

[58] Nor Shahida Mohamad Yusop, John Grundy, and Rajesh Vasa. Reporting usability defects: do reporters report what software developers need? In *Proceedings of the 20th international conference on evaluation and assessment in software engineering*, page 38. ACM, 2016.

[59] Alexey Zagalsky, Carlos Gómez Teshima, Daniel M German, Margaret-Anne Storey, and Germán Poo-Caamaño. How the R community creates and curates knowledge: a comparative study of Stack Overflow and mailing lists. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 441–451. ACM, 2016.

[60] Jie Zou, Ling Xu, Mengning Yang, Xiaohong Zhang, and Dan Yang. Towards comprehending the non-functional requirements through developers' eyes: An exploration of Stack Overflow using topic analysis. *Information and Software Technology*, 84:19–32, 2017.

Other References

[61] Ausblick auf 4.16, umfang der Änderungen & changelog in: heise.de. `https://www.heise.de/ct/artikel/Die-Neuerungen-von-Linux-4-15-3900646.html?seite=9`. Accessed: 2019-01-31.

[62] A blog post on the official Stack Overflow blog with the title "Does Anyone Actually Visit Stack Overflow's Home Page?". `https://stackoverflow.blog/2017/03/09/anyone-actually-visit-stack-overflows-home-page/`. Accessed: 2019-01-09.

[63] Jeff Atwood. A blog post from the creator of Stack Overflow on his blog Coding Horror with the title "What does Stack Overflow want to be when it grows up?". `https://blog.codinghorror.com/what-does-stack-overflow-want-to-be-when-it-grows-up/`. Accessed: 2019-01-09.

[64] Nisa Bakkalbasi, Kathleen Bauer, Janis Glover, and Lei Wang. Three options for citation tracking: Google Scholar, Scopus and Web of Science. *Biomedical digital libraries*, 3(1):7, 2006.

[65] Jacob T Biehl, Mary Czerwinski, Greg Smith, and George G Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322. ACM, 2007.

[66] Frederick P Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E.* Pearson Education India, 1995.

[67] Daniela S Cruzes and Tore Dybå. Research synthesis in software engineering: A tertiary study. *Information and Software Technology*, 53(5):440–455, 2011.

[68] Reid Holmes and Andrew Begel. Deep Intellisense: a tool for rehydrating evaporated information. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 23–26. ACM, 2008.

[69] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[70] Andrew J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on software engineering*, 32(12):971–987, 2006.

[71] Melvin Manis, Jonathan Shedler, John Jonides, and Thomas E Nelson. Availability heuristic in judgments of set size and frequency of occurrence. *Journal of Personality and Social Psychology*, 65(3):448, 1993.

[72] Jürgen Mössinger. Software in automotive systems. *IEEE software*, 27(2), 2010.

[73] Inah Omoronyia, John Ferguson, Marc Roper, and Murray Wood. Using developer activity data to enhance awareness during collaborative software development. *Computer Supported Cooperative Work (CSCW)*, 18(5-6):509, 2009.

[74] Heider Sanchez, Romain Robbes, and Victor M Gonzalez. An empirical study of work fragmentation in software evolution tasks. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 251–260. IEEE, 2015.

[75] International Telecommunication Union Telecommunication Development Bureau. ICT facts and figures 2005, 2010, 2017. `https://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx/`. Accessed: 2019-01-31.

[76] Amos Tversky and Daniel Kahneman. Availability: A heuristic for judging frequency and probability. *Cognitive psychology*, 5(2):207–232, 1973.

[77] David A. Wheeler. More than a gigabuck: Estimating GNU/Linux's size. `https://dwheeler.com/sloc/redhat71-v1/redhat71sloc.html`. Accessed: 2019-01-31.