Decomposing the Rationale of Code Commits: The Software Developer's Perspective

Khadijah Al Safwan
khsaf@vt.edu
Virginia Tech
Blacksburg, VA, USA

Francisco Servant
fservant@vt.edu
Virginia Tech
Blacksburg, VA, USA

ABSTRACT

Communicating the rationale behind decisions is essential for the success of software engineering projects. In particular, understanding the rationale of code commits is an important and often difficult task. We posit that part of such difficulty lies in rationale often being treated as a single piece of information. In this paper, we set to discover the breakdown of components in which developers decompose the rationale of code commits in the context of software maintenance, and to understand their experience with it and with its individual components. For this goal, we apply a mixed-methods approach, interviewing 20 software developers to ask them how they decompose rationale, and surveying an additional 24 developers to understand their experiences needing, finding, and recording those components. We found that developers decompose the rationale of code commits into 15 components, each of which is differently needed, found, and recorded. These components are: *goal*, *need*, *benefits*, *constraints*, *alternatives*, *selected alternative*, *dependencies*, *committer*, *time*, *location*, *modifications*, *explanation of modifications*, *validation*, *maturity stage*, and *side effects*. Our findings provide multiple implications. Educators can now disseminate the multiple dimensions and importance of the rationale of code commits. For practitioners, our decomposition of rationale defines a "common vocabulary" to use when discussing rationale of code commits, which we expect to strengthen the quality of their rationale sharing and documentation process. For researchers, our findings enable techniques for automatically assessing, improving, and generating rationale of code commits to specifically target the components that developers need.

CCS CONCEPTS

• **Software and its engineering → Software evolution**; **Software version control**; **Maintaining software**; **Documentation**.

KEYWORDS

Software Changes Rationale; Software Evolution and Maintenance

ACM Reference Format:
Khadijah Al Safwan and Francisco Servant. 2019. Decomposing the Rationale of Code Commits: The Software Developer's Perspective. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338906.3338979

1 INTRODUCTION

Software development is driven by decisions from various stakeholders, following some *rationale*, at every stage of the software development life-cycle [9]. Given the complexity of software and software development teams, the effective management of rationale for software development decisions is expected to play an important role in the success of software projects [14].

In the context of code changes, *i.e., code commits*, rationale is a major information need. Many research studies support the importance of understanding the rationale of code commits. It is the most common [11] and important [50] information need to understand from code history, and very frequently sought during code review [15, 39]. Unfortunately, it can also be quite difficult to find an answer for it [35, 50].

We posit that a fundamental step to support developers in managing the rationale of code commits is to discover the specific pieces of information that compose it. Thus, the main goal of this paper is to discover the breakdown of components in which developers decompose the rationale of code commits — in the context of software maintenance. Existing studies of software developers informally define the rationale of code commits as answering the question: *"why was this code implemented this way?" e.g.,* [11, 35]. However, this informal question could easily be interpreted by software developers in many different ways, potentially as disparate as: *"what is the purpose of this code?"* [30]; *"why where [these changes] introduced?"* [16]; or *"why was it done this way?"* [35] — all of which request different answers. Since software developers mentioned all these different interpretations when asked about rationale in past studies, we formed our intuition that it could be decomposed into multiple components, each addressing different aspects of the question.

Efforts to study rationale in depth have been carried out in the context of design, decomposing it into various more-specific components [48]: *e.g., design constraints*, *assumptions*, or *certainty of design*. In the context of software maintenance, Burge et al. prescriptively propose some questions that may answer rationale [9]. We, instead, take a *descriptive* approach, *i.e.,* we aim to discover *how developers decompose* the rationale of code commits — as opposed to conceptually and rigorously decomposing the concept. We employ this practical approach because we ultimately aim to

assist developers in finding and recording *what they mean* by the rationale of code commits.

We used a mixed methods approach in our study. First, we discovered how developers decompose the rationale of code commits by interviewing 20 software developers. Then, we ran a survey to ask an additional 24 developers about their experiences with (needing, finding, and recording) the rationale of code commits — both generally and for individual components. Our intention with these questions was to find areas of improvement in their practices.

We found that software developers decompose the rationale of code commits into 15 separate components that they could seek when searching for rationale: *goal*, *need*, *benefits*, *constraints*, *alternatives*, *selected alternative*, *dependencies*, *committer*, *time*, *location*, *modifications*, *explanation of modifications*, *validation*, *maturity stage*, and *side effects*. Some of these reported components were not previously mentioned in studies of rationale in other contexts, *e.g.,* [48], and were instead specific to the context of software maintenance — *e.g., committer* and *time*. Understanding which components developers seek in rationale is an important problem, since most developers reported seeking it multiple times a week or more often, and spending more than 20 minutes to find it in hard cases.

Our study also revealed areas of improvement in developers' practices regarding the rationale of code commits. For example, developers most struggled to find *side effects* and *alternatives*, and they need to find them on average multiple times per month and per year, respectively. Additionally, developers least often record: *alternatives*, *selected alternative*, *constraints*, and *maturity stage*, even if they need to find them on average multiple times per year (*alternatives*) and per month (remaining ones).

Our findings have multiple implications for practitioners. Our decomposition of the rationale of code commits provides: (1) a *common language* to use when discussing it, which practitioners can use to (2) assess and (3) strengthen the quality of their rationale sharing and documentation processes. While we do not expect practitioners to document all components in all situations, they now have an extensive list of components to judge which ones are relevant for each situation. Our findings also facilitate multiple lines of research. Given our decomposition of rationale, techniques could now be produced to automatically: (1) assess the quality of rationale documentation, and (2) recommend or (3) generate missing components.

This paper provides the following contributions:

- A detailed model of the components into which software developers decompose the rationale of code commits.
- A study of the experiences of software developers needing, finding, and recording the rationale of code commits in general.
- A study of the experiences of software developers needing, finding, and recording the individual components of the rationale of code commits.

2 RELATED WORK

Existing work supports the importance of rationale management throughout the software development life-cycle, *e.g.,* [9, 14]. Thus, multiple approaches and systems have been proposed to integrate rationale management in the process of requirements engineering, software design and architecture, *e.g.,* [18, 34]. However, most existing research efforts have focused on design rationale, which has been described from multiple perspectives *e.g.,* [41, 48] to understand the pieces of information that could express design rationale.

In this paper, we focus on rationale for software maintenance, particularly for individual code commits. Most research in this context empirically found that developers strongly need it, but, to the extent of our knowledge, no existing work studied the components of rationale that developers need during software maintenance.

Rationale of software requirements. Multiple models were proposed to extend requirements models to encourage the capture of rationale within them, *e.g.,* [4, 28]. In addition to these models, tools have also been proposed to manage rationale of software requirements *e.g.,* [3, 23, 34].

Rationale of software design and architecture. Many schemes have been proposed to capture the design and architecture rationale. The schemes can be divided into two categories: decision-centric *e.g.,* [36] and usage-centric approaches [9]. The decision-centric approaches *e.g.,* [41, 48] focus on capturing the rationale as a decision-making process utilizing Toulmin's model of argumentation [51] and Rittel's Issue-Based Information System (IBIS) [32]. The usage-centric approaches focus on capturing rationale without representing the decision-making process, [18, 48, 52**?**].

The usage-centric approaches "recognize that organizing rationale around decisions is not the best way to elicit and characterize some of the rationale needed for making appropriate design decisions" [9]. Jarczyk et al. provided a survey of the systems developed to support design rationale, all of which were based on Toulmin's model and/or IBIS [24]. Design rational has also been supported by multiple tools, *e.g.,* [8, 20], which can be useful to detect inconsistencies, omissions, and conflicts [49].

Rationale in software evolution and maintenance. Burge et al. prescriptively enumerate a few questions that may answer rationale in software maintenance [9]. We, in turn, provide a *descriptive* model of rationale in the context of software maintenance, from the perspective of what developers need to find when they seek it.

Studies involving software history and developers' information needs in the last decade [9, 11, 15, 16, 30, 35, 38, 39, 43, 50] establish a strong demand for rationale. As such, some work has focused on capturing rationale from software artifacts like IRC discussions [2], or user reviews [33]. Our work is motivated by these empirical studies that highlight the importance of finding rationale of code commits. The most closely related study to ours is Tao et al.'s [50]. They found that the most important information need for understanding code commits is rationale, which is sometimes easy, sometimes difficult. Our study validates Tao et al.'s results, since our participants reported similar ratings of importance and difficulty for finding the rationale of code changes. This also shows that we studied a similar population of developers.

Our study extends Tao et al.'s by finding: the individual pieces of information that compose rationale; the experiences of developers needing, finding and recording those individual pieces; and recommendations to improve their documentation. In particular, this finer level of granularity enables us to provide a possible explanation for one of the main phenomena observed by Tao et al.: that rationale is

Table 1: Preliminary model of rationale of code commits

Component	Component Expressed as a Question	Literature References
Goal	What do you want to achieve?	[9, 30]
Need	Why do you need to achieve that?	[16, 46]
Location	What artifacts were changed?	[9]
Modifications	What specific changes were performed in the artifacts?	[30, 46]
Alternatives	What other alternatives did you have?	[35]
Selected alternative	Why did you make those specific changes and not others?	[30, 35]
Validation	How do those specific changes achieve the goal?	[9, 35]
Benefits	What is the benefit of what you want to achieve?	[9, 30]
Costs	What risks could come from these changes?	[9, 30]

easy to find when it is *well documented*. We posit that rationale is deemed well documented when it contains the specific components that the developer is seeking at that moment.

Software comprehension. Researchers developed techniques to summarize and document code changes, [10], clarify the rationale behind code changes [12, 37], or answer "what" and "why" questions about code changes [7, 25, 26, 29, 42]. Other researchers mined software repositories to characterize commits [1, 19, 22**?**], and others studied the impact and risk of changes [27, 44, 53].

3 RESEARCH METHOD

We used a mixed-methods approach in this study. We first discovered the individual components into which developers decompose the rationale of code commits using semi-structured interviews. Then, we studied the experiences that developers have needing, finding and recording rationale and its components using the same interviews and a survey. This mixed-methods strategy allowed us to decompose rationale through rich one-on-one conversations with developers, while also reaching more participants for our questions about needing, finding, and recording it. Mixed-methods have been successfully employed by many other studies of software developers, *e.g.,* [11, 21, 45, 50]. Our study answers four research questions:

RQ1: What is the experience of developers needing, finding, and recording the rationale of code commits? First, we investigated this research question to understand the effort that developers dedicate to seek and document the rationale of code commits. Tao et al.'s study found that finding the rationale of code commits is very important, and it is easy or hard to find depending on how well-documented it is [50]. We extend their study by asking developers five additional questions in three different contexts: needing, finding and recording rationale.

RQ2: Which components do software developers decompose the rationale of code commits into? Next, we asked developers about the specific components into which they would decompose the rationale of code commits — the pieces of information that they believe would compose a high-quality, detailed description. We aim to discover an extensive set of components that developers could be looking for. This model will inform developers wanting to improve their documentation of rationale of code commits — whether they aim to document it fully or just more thoroughly.

RQ3: What is the experience of developers needing, finding, and recording the individual components of the rationale of code commits? Next, we studied how developers need, find, and record different components differently. This will now enable developers to improve their documentation of rationale in an effort-efficient manner, *e.g.,* by focusing on documenting those components that are most needed or most hard-to-find.

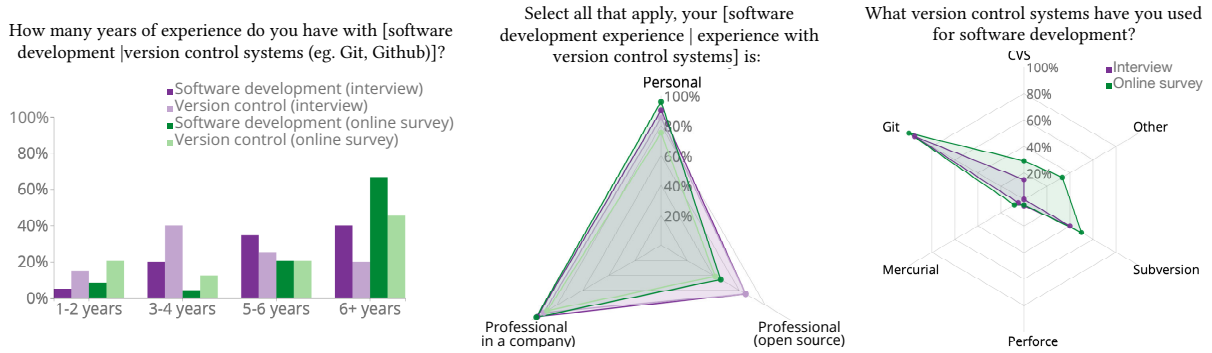
RQ4: Would comparing the experience of developers needing, finding, and recording the individual components of the rationale of code commits with each other reveal areas for improvement? Finally, we performed a cross-dimensional study (*i.e.,* comparing need vs. finding vs. recording components) to investigate areas for improvement in current practices of recording and retrieving rationale of code commits. Identifying gaps, *e.g.,* between needed and recorded components, will provide valuable recommendations for developers wanting to improve their documentation of rationale of code commits.

3.1 Developer Interviews

Interview Design. We designed and refined our interview script through five pilot sessions. Our interview consisted of two main parts. The first part focused on finding the perspective of developers of the components that form the rationale of code commits (RQ2). The second part aimed to understand the experiences of developers needing, finding, and recording it (RQ1) and its components (RQ3). We study RQ4 by comparing participants' responses in different dimensions.

Decomposing the Rationale of Code Commits. We started our interviews by giving our participants the definition of *rationale of code* that is most common in the research literature, *i.e.,* the answer to *"why is the code this way?"* [11, 35]. We did this to make sure that all participants had a uniform definition of the concept that we were going to discuss. Next, we asked them to describe real situations in which they investigated a code commit to understand its rationale. We took this step to stimulate their memories and set them in the right context. After that, we asked participants to decompose the rationale of code commits into components. We asked this question after they had been thinking about their own experiences searching for it, with the intention of maximizing the number of components that they would report. Then, we showed them a *preliminary model* (see Table 1) of components of the rationale of code commits that we created by studying the research literature — including components to which researchers have referred as *rationale* [9, 16, 30, 35, 46]. We used this preliminary model as a probe to prime our participants and get them in the right frame of reference. We asked participants to critique and extend the preliminary model — taking their previous decomposition into consideration — to the extent that they believed necessary to build a *final model* of all the components of the rationale of code commits. For any component that was added by participants, we asked them to describe it with a name, question, and example answer. We presented the same preliminary model to all participants — *i.e.,* we did not show the modified models to other interview participants.

Using the preliminary model as a probe served multiple purposes: it clarified the scope of our study, it allowed developers to discuss an extensive set of components, and it allowed us to reach saturation of answers much faster (interviewing fewer participants)



(a) Experience (Years) **(b) Experience (Type)** **(c) Version control systems**

Figure 1: Demographics of our interview and survey participants

than if we had only relied on our participants' experiences and decompositions — since situations in which many components are needed simultaneously may be rare, or because people's memory is generally unreliable.

However, by using a preliminary model, we had the risk of introducing confirmation bias [40]. We took multiple measures to reduce this potential bias. First, we presented the preliminary model neutrally, as *"this model"* — avoiding potentially-biasing adjectives, such as "ours" or "preliminary". Second, we built it by studying in the research literature, reducing the risk of inserting our own opinions in it. Third, we presented the preliminary model to participants only after they had produced their own decomposition — without having seen it. Fourth, we asked participants to consider their own decomposition when they critiqued and extended the preliminary model. We believe that we were successful with these efforts, since our final model of rationale of code commits (see Table 2) is much more extensive than the preliminary model (see Table 1). The preliminary model had only 9 components, whereas the final model has 15.

Developer Experiences with the Rationale of Code Commits. In the second part of the interviews, we asked participants to rate their experiences needing, finding, and recording rationale of code commits and its components in Likert-scale-style questions. We report our specific questions and their scales in Section 4, in Figures 2–7.

Interview Analysis. To answer RQ2, we used card sorting [47] to aggregate all the components that at least one participant included in their final model of rationale of code commits. First, one author of this paper created a card for each component. Then, each of the two paper authors separately sorted the cards to aggregate those that described similar components. For example, we aggregated into *"Side Effects"*: the preliminary component *"Costs"*, and the *"Merge Conflict/Success"*, *"Limitation"*, and *"Impact"* components that were mentioned by different participants. After that, both authors collaboratively consolidated the two sets of individually-aggregated components, comparing them and deciding on disagreements. Then, we characterized each of the resulting aggregated components with a name, a question, and an example answer to the question based on a hypothetical commit. Finally, we categorized the resulting components into themes.

To answer RQ1 and RQ3, in Section 4, we report the percentage of participants that provided each answer to each Likert-scale-style question. All participants provided answers about the rationale of code commits in general. For individual components of rationale, the answers include only the participants that included them in their final model. Whenever we aggregated components through card sorting, we also aggregated the responses about experience with them.

Recruitment. We used snowball sampling [6], *i.e.,* we asked participants to refer our study to their own contacts. We advertised our study in mailing lists in our university that covered software developers of diverse experience, *e.g.,* developing various university software systems, and graduate students with professional software development experience. We compensated interview participants with a $20 Amazon gift card.

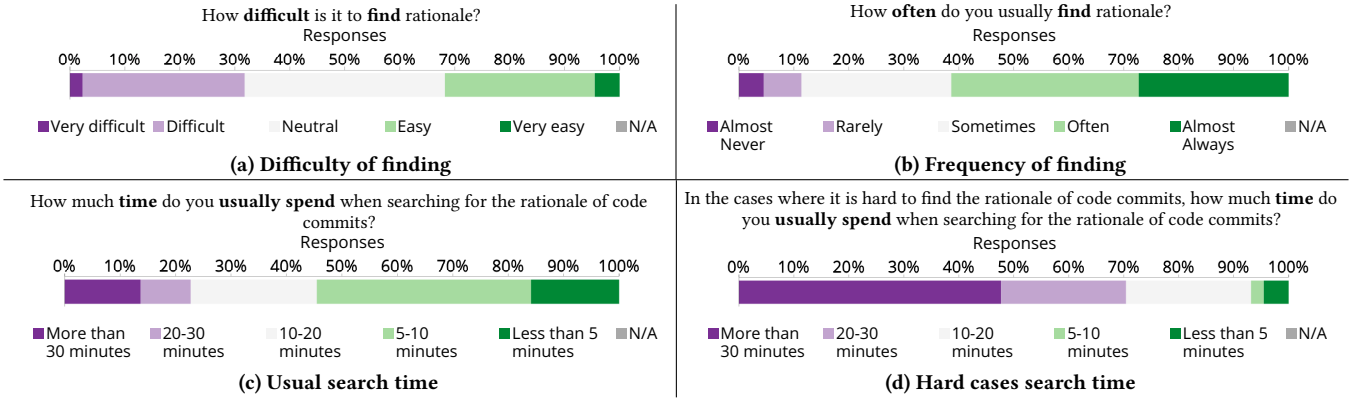
We interviewed 20 participants, after having discarded three other interviews for various reasons: one participant could not describe an example of seeking rationale of code commits, another voluntarily expressed lack of experience throughout the interview, and we found out that the last one had knowledge about our interview materials.

3.2 Survey

Once we had identified the components of the rationale of code commits through our interviews, we used a survey to obtain more answers about developers' experiences needing, finding, and recording it and its components. We refined our survey through four pilot versions, improving its clarity and the time required to complete it. Our survey included the same Likert-scale-style questions that we asked our interview participants for RQ1 and RQ3, but the reference model of rationale of code commits that we gave survey respondents was the final model resulting from our analysis for RQ2. Our results for RQ1 and RQ3 include the answers that we obtained both from our interviews and from the survey.

We also used snowball sampling for our survey, asking interview participants to advertise it to their contacts. We also advertised it through public channels and social media. We encouraged participation by raffling a $50 gift card.

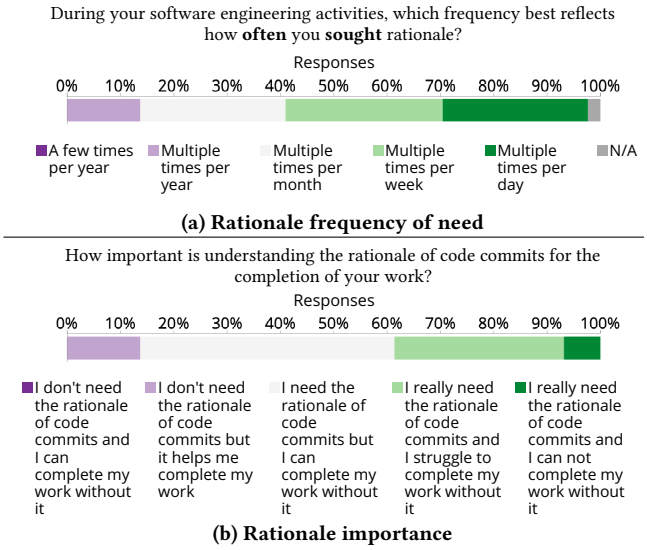
We analyzed 24 survey responses, after having discarded four responses. We discarded two survey responses that reported longer



(a) Difficulty of finding (b) Frequency of finding

(c) Usual search time (d) Hard cases search time

Figure 2: Experience of developers finding rationale



(a) Rationale frequency of need

(b) Rationale importance

Figure 3: Experience of developers needing rationale

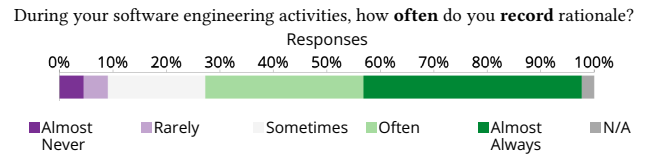


Figure 4: Experience of developers recording rationale

experience with version control than with software development to avoid bias introduced from experience with version control for purposes other than coding. We also discarded two surveys that we deemed as having been done carelessly — taking less than 10 minutes. We determined this cut-off point through our pilot surveys — we asked one pilot participant to fill the survey carelessly and it took 10 minutes.

4 RESULTS

We answer our four research questions in this section by presenting the results of our study. We represent in Figure 1 the demographic information of our interview and survey participants.

RQ1: What is the experience of developers needing, finding, and recording the rationale of code commits?

Need. The participants of our study reported needing to seek rationale with diverse frequencies (see Figure 3a): multiple times per day (27%), multiple times per week (29%), multiple times per month (27%), and multiple times per year (13%). This also means that, overall, the majority (56%) of our study participants need rationale relatively frequently: multiple times per week or more often.

When asked about how important it is to understand the rationale of code commits, 86% of our participants reported needing the rationale of code commits (see Figure 3b), from which: 7% cannot complete their work without understanding it, 31% struggle to complete their work without it, and 47% can complete their work without it but still need it. The remaining 14% do not need the rationale of code commits, but report that it still helps them complete their work. A very similar question was studied by Tao et al. [50], whose participants *"generally considered knowing the rationale of a change as the top priority in change-understanding tasks"*. Our finding is aligned with theirs, since a majority of our participants reported needing the rationale of code commits, which validates that we are studying a similar population of developers.

Finding. Our participants' responses in Figure 2a indicate that the difficulty of finding the rationale of code commits, in general, is not easy nor difficult. Software developers (on average) selected neutral difficulty of finding the rationale of code commits. This finding also generally agrees with Tao et al.'s, since their participants reported that the rationale of code commits was generally easy to find, but sometimes hard, depending on "the availability and quality of the change description" [50].

Regardless of how hard it is, we were also interested in how often developers end up finding the rationale of code commits altogether. For this aspect, our study participants responses are positive (see Figure 2b). Most software developers find the rationale of code commits often or almost always. Only a few participants (11%) rarely or almost never find the rationale of code commits.

In addition to studying whether software developers find the rationale of code commits, we also studied how much time they spend searching for it. Figure 2c and 2d shows the times that our participants reported spending when searching for rationale. In

Table 2: Resulting model of the rationale of code commits

Theme	Component	Component Expressed as Question	Example Answer
Change Objective	*Goal	What did you want to achieve?	I wanted to implement functionality to sort the product list by price.
	*Need	Why did you need to achieve that?	Our user requested to be able to sort the list of products by price.
	*Benefits	What is the benefit of what you want to achieve?	The new option of sorting products by price will be useful for many customers in addition to the one who requested it.
Change Design (pre-implementation assessment)	Constraints	What were the constraints limiting your implementation choice?	The sorting algorithm had to be space efficient because it should work in embedded devices.
	*Alternatives	What other alternatives did you have?	I could have used the bucket sort algorithm, but this option was not feasible because I would not have known the maximum price before sorting.
	*Selected Alternative	Why did you make those specific changes and not others?	I implemented heap sort because it is space efficient and it has a predictable speed.
	Dependency	What other changes does this change depend on?	This change depends on the API that provides the product list to be updated to use JSON format.
Change Execution	Committer	Who changed the code?	Developer X, who is responsible for the "products" page.
	Time	Why were the changes made at that time?	This change happened before our 3.0 release to meet the customer contract for that release.
	*Location	What artifacts were changed?	The "product" class was updated.
	*Modifications	What specific changes were performed in the artifacts?	I added a "sort" method in the "product" class implementing heap sort and now the "listProduct" method calls "sort" first.
	Explanation of Modifications	What are the details of the implementation?	The code sorts the products by price by performing the following steps: 1- Build a heap from a list of "products" in O(n) operations. 2- Swap the first list-element with the final list-element of the list. 3- Decrease the considered range of the list by one. 4- Shift the new first element to its appropriate index in the heap based on the "price". 5- Repeat step (2) unless the considered range of the list is one element.
Change Evaluation (post-implementation assessment)	*Validation	How did those specific changes achieve the goal?	By using the heap sort algorithm, our customers can now see a sorted product list in their memory-limited hardware.
	Maturity Stage	How mature is this code?	The change is an initial implementation, which still has to be fully tested after the API for the products list is updated.
	*Side Effects	What are the side effects of the change?	The integration test will fail if the API that provides the product list is not updated. At the same time, merging this change with the main branch after updating the API might break the existing code. Also, our implementation of heap sort may be too complex for beginners and may slow down maintenance.

* Components that were included in the preliminary model of rationale of code commits. We extended the preliminary component *Costs* to *Side Effects* to include other side effects mentioned by participants *e.g., Impact.*

the usual cases, slightly more than half participants (54%) spend less than 10 minutes. However, in the hard cases of searching for rationale, only slightly less than half participants (47%) spend more than 30 minutes searching for the rationale of code commits. One participant said about the time they spend searching for rationale in the hard cases that it *"depends how responsive the other person is."* When considering the relatively high frequency with which developers search for rationale of code commits, it can be a rather time-consuming task.

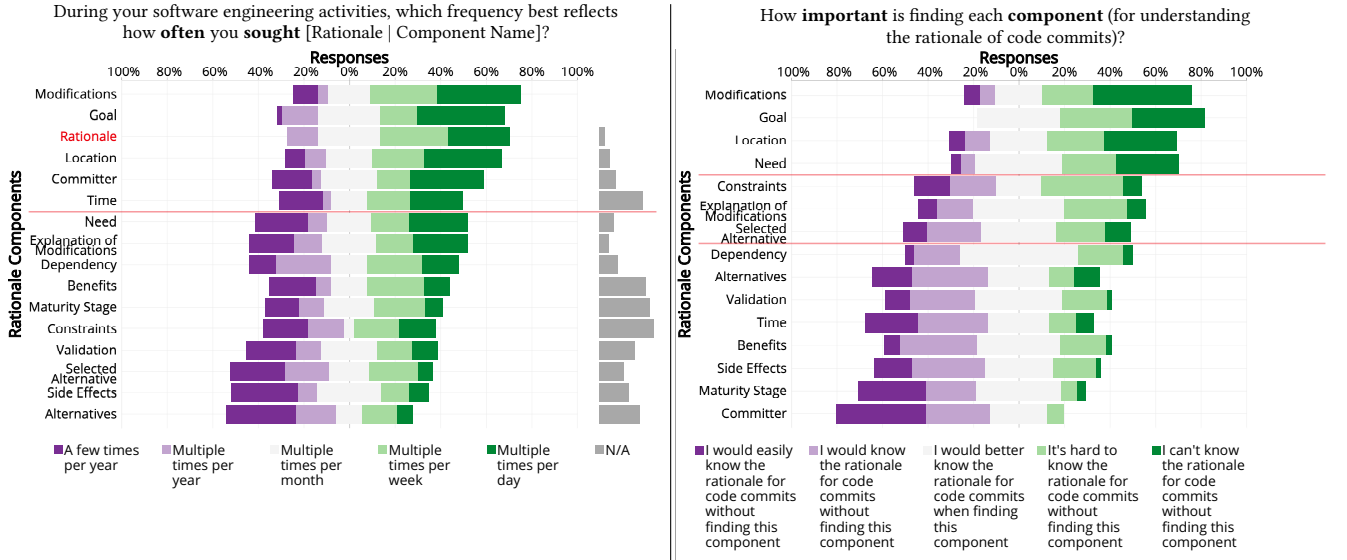
Recording. Regarding the frequency of recording rationale in general, Figure 4 shows our participants' responses. The majority of them reported recording the rationale of code commits often (30%) or almost always (40%). Interestingly, our participants reported recording the rationale of code commits with similar frequencies than they reported finding it, which suggests that documentation efforts generally help others find rationale.

RQ2: Which components do software developers decompose the rationale of code commits into?

We display in Table 2 the model of rationale of code commits that we discovered. It represents the union of all the models that our participants reported. As we discussed in Section 3.1, we obtained this model aggregating all the components that were mentioned by at least one participant in their final interview model of rationale of code commits. Each participant built their final model by adding and/or removing components to the preliminary model, while also considering their own rationale decomposition. Altogether, our participants reported a total of 27 components of the rationale of code commits — they added 18 components to the 9 components in the preliminary model. Since many of those components reported very similar concepts, we aggregated them using card sorting to obtain the final model that we show in Table 2. This resulting model of rationale of code commits includes 15 components into which developers decompose it. We categorized the resulting components into four themes.

Our goal with this model of rationale was to gather an extensive set of specific components of the rationale of code commits that developers may be looking for when they need it. For that reason, when a participant removed a component from the preliminary model, we still kept it in our resulting final model (in Table 2). Besides, only a few participants removed components.

When participants decided to remove components from the preliminary model, they mentioned two main reasons: overlap with other components, and the component being out of scope. In terms of overlap among components, one participant thought that *goal* and *need* can be the same most of the time and preferred to merge them together, deleting the *goal* component. Another thought that *need* is included in *benefits* and *cost*, deleting the *need* component.



(a) Frequency of need **(b) Components importance**

Figure 5: Experience of developers needing individual components of rationale

Another participant deleted *benefits* because it is included in *goal*. Another one considered *location* as part of *modification*.

We believe that it is possible that the answers for different components can be the same in some cases. For a single code commit, components of the same theme (see Table 2) may have very similar answers to their expressive question. However, in many other cases they will be different, making it useful to separate those components. We illustrate the differences between components in Table 2 by including the components expressed as questions and different example answers for different ones.

Other participants removed components that they considered out of scope of rationale. From our 20 interview participants: two participants removed *modifications* because they considered it too low-level; three participants removed *location* because it would not tell why the changes were made; three participants removed *alternatives*, *e.g.*, *"alternatives is not something that you actually implement!"*; and one participant deleted *validation*, saying that *"validation answers why the code is correct, not the rationale"*. Despite these disagreements, the majority of our interview participants (18, 17, 17, and 19, respectively) considered that these components do belong in the rationale of code commits.

Furthermore, our participants generally provided positive comments about the preliminary model — describing it as *e.g.*, *"a good model," "detailed," "thorough," "comprehensive," "holistic,"* or *"exhaustive."* They thought that the model *"formally define[s] rationale"* and that *"the components seem to be related to each other, but classified differently to each other."* One participant said that the model is a *"logical framework for thinking through rationale because [it is] a sort of wide-open concept. It's a little bit hard to know how to think about [rationale]. [The model] makes sense as a directed way to understand a specific commit or a series of commits. Why they are the way they are."*

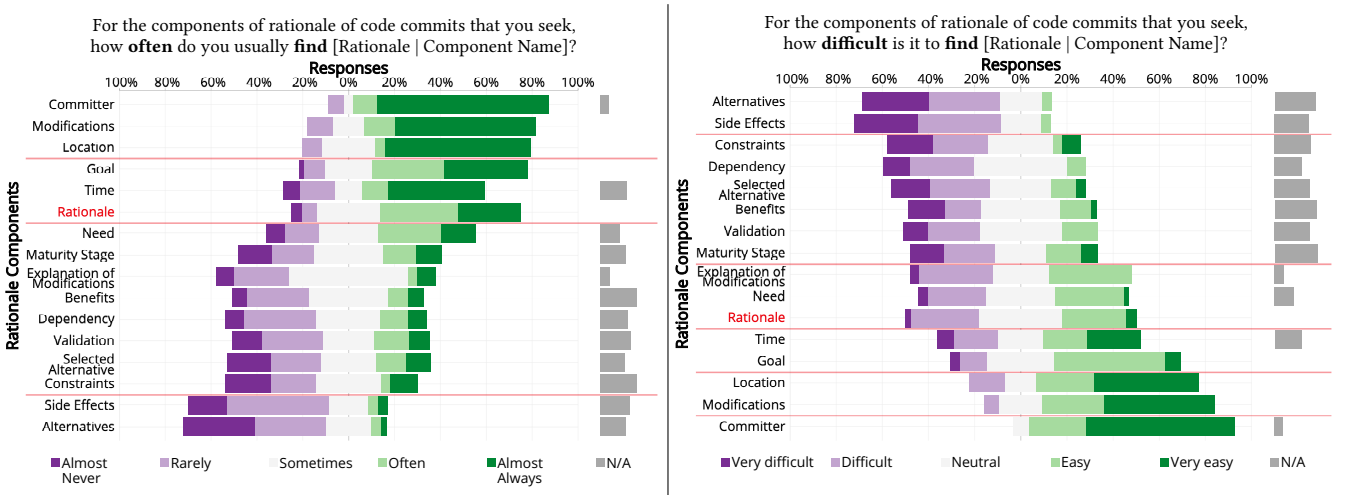
Many participants added components to the preliminary model. As we mentioned earlier, we used card sorting to aggregate them to the preliminary model and with each other. The 18 components proposed by participants were: *technical requirement*, *timeliness*, *documentation*, *guidelines*, *non-feasible alternative*, *opinion selected alternative*, *constraints*, *dependency*, *committer*, *time/date*, *explanation of modifications*, *result*, *environment*, *scope for future development*, *quality*, *merge conflict/success*, *limitation*, and *impact*. For each added component, we also asked participants to describe them with a name, expressive question, and example answer to the question.

The fact that many participants added and some removed components suggests that our participants were not strongly biased towards simply agreeing with the preliminary model. More importantly, it also suggests that different developers seek different components at different times. Our study throws light into this phenomenon. Thus, we pose that the rationale of code commits would be much easier to comprehend, search for, and document when it is expressed as its components — not necessarily all of them at all times, but the ones that are relevant for each situation.

RQ3: What is the experience of developers needing, finding, and recording the individual components of the rationale of code commits?

We plot the distribution of answers to our questions about individual components of the rationale for code commits in Figures 5–7. We cluster components into similar groups according to the mean value of their responses using the Scott-Knott [17] algorithm. We sort the components in our figures by the mean value of their responses and we use red horizontal lines to separate clusters. We also include as reference point in each figure the responses that our participants gave for rationale in general.

Need. Figure 5a shows the distribution of responses for how frequently developers need each component of the rationale of code commits. Overall, the frequency with which developers need different components of the rationale of code commits is highly similar for all components — and for rationale itself in general. In this case,



(a) Rationale components frequency of finding

(b) Rationale components difficulty of finding

Figure 6: Experience of developers finding individual components of rationale

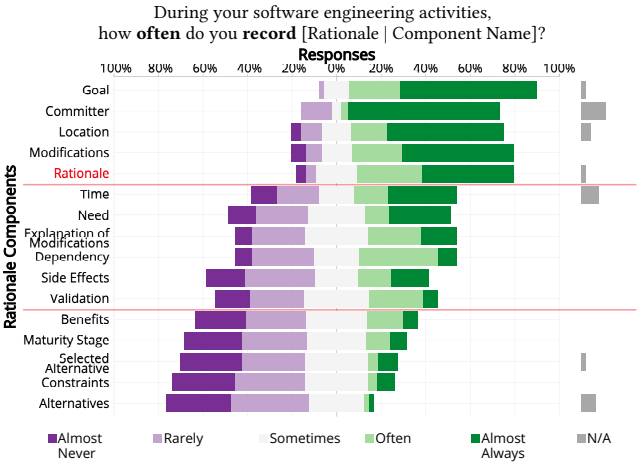


Figure 7: Experience of developers recording individual components of rationale

the Scott-Knott algorithm returns only two very-similar clusters. While some of the most often needed components (like *modifications*, *location* or *committer*) are normally automatically recorded by revision control systems, many other components are similarly often needed and are not recorded automatically (like *need* or *dependency*, or *constraints*). These results show that practitioners would benefit from regularly recording these frequently-needed components.

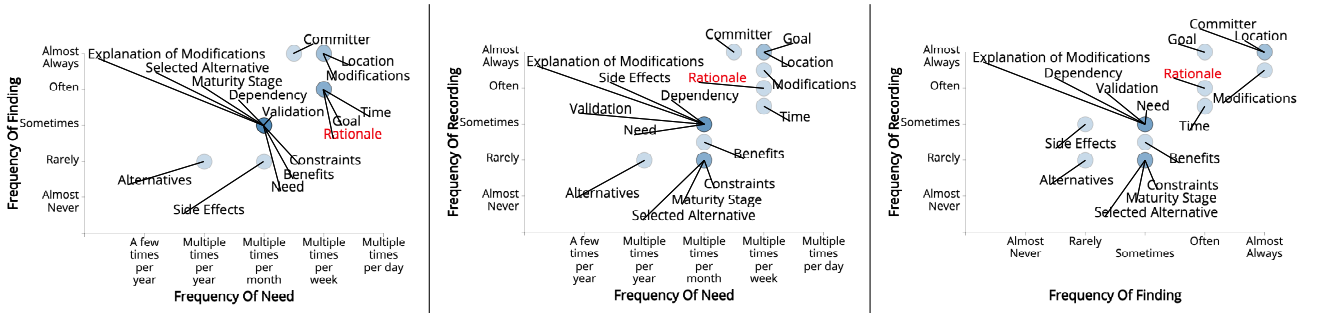
Figure 5b shows the relative importance of each component to understand the rationale of code commits reported by developers. These results show that most developers mentioned that most components are important enough that they would understand the rationale of code commits better if they knew that component. We also observe that developers wanting to document the most important component of rationale should focus on documenting the *goal* and *need* of their changes, since the other most-important components (*modifications*, *location*) are already recorded by revision control.

Finding. Figure 6 shows the relative frequency and difficulty of finding reported by developers for each component. Unsurprisingly, the most frequently found components (and also the easiest to find) are those automatically tracked by revision control (*committer*, *modifications*, and *location*), followed by *goal* and *time*. However, the frequency (and easiness) of finding quickly drops for all other components, bringing our attention to a clear problem in finding the remaining components. These results bring attention to the need to improve documentation for the other components, since they are hard to find. This clear divide could also explain why developers talking about rationale in general say that sometimes it is much harder to find rationale than other times [50] and it takes longer (Figure 2d).

Recording. Figure 7 shows the relative frequency with which developers reported to record components of the rationale of code commits. Again unsurprisingly, the most frequently recorded components are those recorded automatically by revision control, but again the frequency of recording drops dramatically for the remaining components (which also probably explains why they are hard to find). These results show that, even if developers claim to frequently record rationale in general, there are many components of it that they are in fact not recording frequently (even if they are relatively often needed).

RQ4: Would comparing the experience of developers needing, finding, and recording the individual components of the rationale of code commits with each other reveal areas for improvement?

Figures 8a and 8b show that software developers are most frequently finding and recording the most frequently needed components of rationale. Most of the components are in the middle frequency of need and finding. However, this result brings attention to the



(a) Components need vs. finding (b) Components need vs. recording (c) Components finding vs. recording

Figure 8: Cross-dimensional analysis of developers experience with the individual components of rationale of code commits

fact that there are many components that are not too frequently needed, but when they are needed they are really hard to find. Developers most struggled to find *side effects* and *alternatives*, even if they need to find them on average multiple times per month and per year, respectively. In these cases, the difficulty of finding these components may overcome their limited frequency of need. Thus, practitioners may want to pay more attention to documenting these not-so-frequently-needed components.

The difficulty of finding rationale depends on many factors, *e.g.*, the complexity of code commits, the developers' documentation of code changes, and the need for discovering the rationale. One of the participants said about the giving up of searching rationale *"I would completely give up if I couldn't find any record in our system and the author was someone who either is no longer at our company or is somebody who just doesn't write code anymore. Yeah. I give up when I've exhausted all the possibilities, but if I really need to know I would keep trying until I figured it out."* For the components of rationale that are not easy to find, guidelines could be established and tools could be developed to simplify finding these components. One participant said about finding rationale: *"from my experience, the rationale, it's easier to figure out once your team kind of has standards or guidelines."*

The recording of rationale goes hand in hand with the finding of rationale (see Figure 8c). Unsurprisingly, not recording some components makes it hard to find them later. The rarely recorded components were: *alternatives*, *selected alternative*, *maturity stage*, and *constraints* — even when developers need to find them on average multiple times per year (*alternatives*) and per month (remaining ones). The identification of this group of rarely recorded components should encourage researchers to develop tools specifically focused on recording or answering them. For example, a technique to evaluate the maturity stage of a commit will aid developers seeking this component without the need for other developers to manually document it.

5 DISCUSSION AND IMPLICATIONS

We discuss how our findings could improve the management of rationale and how they differ from the knowledge about rationale in other contexts. Then, we discuss implications for educators, practitioners, researchers, and tool builders.

Improving the Management of Rationale. The management of rationale of code commits is an important problem. Software developers need the rationale of code commits (see Figure 3b) and spend a significant amount of time searching for it (see Figures 2c and 2d). Our decomposition of the rationale of code commits based on the perspective of software developers (in Table 2) is a necessary first step for any efforts to try to improve its management.

It is important to notice that we do not argue that developers should document every component of rationale all of the time. In fact, our participants mentioned this concern, *e.g., "I know it might not be doable or possible because no one will ever answer all these in a commit. However, it is a good model."* Our goal is instead to characterize the individual pieces of information that developers may seek from rationale at different times. We, in fact, observed that they seek different components with different frequencies (see Figure 5a). Our characterization informs developers of the broad set (and characteristics) of components of rationale that may be later sought by others — so that they can make an informed decision of which ones are worth documenting at each particular time.

A starting point for improving rationale documentation would be those areas of improvement that we identified studying RQ4. We observed that most components of rationale of code commits are frequently not recorded (see Figure 7 and 8b), not found (see Figures 6a and 8a), or difficult to find (see Figure 6b). This observation suggests the need for improving tools and practices to simplify the management of rationale (recording and finding).

Rationale in Other Contexts. Some of the components of rationale of code commits that we discovered are also relevant for rationale in software requirements, design, and architecture. These are: *constraints* [18, 20, 48, 52**?**], *alternatives* [20, 32, 36**?**], and *validation* [18, 20**?**]. However, we also discovered components that are specific to the context of code commits: *committer*, *time*, *location*, and *modifications*. They generally refer to the execution of the code change. Our participants indicated that these components are more needed and important that those that also show in rationale in other contexts (see Figures 5 and 8a). Similarly, there were some components that were not mentioned in our study and were reported in previous work as part of design rationale. Examples are: design *assumptions* [18, 48, 52] and *weaknesses* [18, 48].

Implications for educators and practitioners. Our results provide a common language for discussing the rationale of code commits in detail — by decomposing it into its individual components. This common language will allows educators to disseminate the multiple dimensions of rationale of code commits. They could advocate the practice of documenting the rationale of code commits considering the components of rationale that other developers need.

We also encourage practitioners to consider the range of rationale when documenting their code commits to avoid missing important components. Managers can collaborate with developers to establish team-specific guidelines for documenting rationale. These guidelines could trigger developers to capture the rationale of their code commits appropriately for each situation, building beneficial habits and long-lasting documentation.

Implications for researchers and tool builders. In practice, software developers fail to document various components of rationale, making it hard on other software developers to find it (Figure 8b). Any efforts to automate or support the process of finding the rationale of code commits will require a rich understanding of the specific pieces of information that developers seek when they need it. The automated assistance for rationale documentation and retrieval is now easier with our model. For example, code-committing interfaces could be enhanced with templates to offer suggestions to record the different components/themes of rationale. This idea was examined in the area of bug reports [5, 13], not only to assist the documentation of various components but also to measure their quality. We expect that these ideas will also translate to the documentation of the rationale of code commits.

New techniques and tools could also be developed to automatically generate the rationale of code commits, saving developer effort. Existing efforts of automatic documentation of rationale treat it as a single piece of information *e.g.,* [7, 25, 26, 29, 42]. Our model now allows future research efforts to generate *targeted* pieces of information to build a more thorough documentation of rationale.

6 THREATS TO VALIDITY

Construct. To answer our research questions, we asked both open and quantitative questions. We scheduled the interview sessions to be relatively long (two hours), making sure that we gave the participants enough time to express their ideas and share their thoughts. At the beginning of each interview section, we asked the participants to *"answer the questions in [their] own words and provide as much detail as [they] feel is relevant to address each question"*. We also placed an open question at the end of the interview to allow the participants to share any additional information about the topic.

Internal. The methods we used in our study, interviews and surveys, can be affected by bias and inaccurate responses. This effect could be intentional or unintentional. We gave gift cards to the interview participants and some of the survey participants, which could have biased our results. To mitigate these concerns, we clearly indicated that the compensation is for the time spent and not the answers given. We repeatedly and constantly used phrases to encourage the participants to provide their own honest opinions, using the phrase *"based on your experience"* in most of the questions. We also clearly indicated that the participants should *"feel free to change/add/delete components or not."* Sometimes, we also indicated that *"there is no right or wrong answer; we are interested in what you think and your perspective."*

We also took multiple steps to reduce potential confirmation bias [40] resulting from using a preliminary model. We asked participants to describe their own examples and decomposition of rationale into components before they ever saw the preliminary model. We formed the preliminary model based on knowledge from the research literature, and presented it neutrally. The fact that the preliminary was largely extended from 9 components into 15 validates that potential confirmation bias was minimal in our study.

Another threat to validity in our study is drawing conclusions based on recollected memories [31]. We are interested in capturing developers' opinions about what components constitute rationale, independently of how accurate their recollection is. We encouraged participants to take their time to recall situations and to report the components that mattered in their experience.

External. Our studied developers may not fully represent the whole developer population. To mitigate this threat, we recruited a diverse population, with diverse types and amounts of experience (Figure 1). Furthermore, our studied population was similar to the ones previously studied in the literature, since we obtained similar answers for our two questions about rationale that were already studied by Tao et al. [50].

7 CONCLUSION

Developers invest valuable time and resources in the process of discovering the rationale of code commits, which they perform frequently and is difficult. However, any efforts aiming to improve this process will necessarily require a good understanding of the specific pieces of information that developers seek when they search for rationale of code commits.

We applied a mixed-methods approach in this study. First, we performed a series of interviews with software developers to discover the components into which developers decompose the rationale of code commits. Then, we ran a survey to better understand their experiences needing, finding, and recording the rationale of code commits. We found that developers decompose rationale of code commits into 15 components along 4 themes, and that they have different experiences with different components. Overall, developers need to find most components with similar frequency. However, they mostly only record and find those components that are automatically recorded by revision control systems. This finding suggests that there is space for both researchers and practitioners to improve the practices of managing the rationale of code commits.

This work provides a descriptive representation of the rationale of code commits that practitioners can use to improve their documentation and communication of rationale. Additionally, researchers and tool builders can support the management of the rationale of code commits using our discovered components of rationale and the experiences of software developers with them.

REPRODUCIBILITY

An artifact containing our interview questions and survey instrument is available at https://doi.org/10.5281/zenodo.3261842.

REFERENCES

[1] A. Alali, H. Kagdi, and J. I. Maletic. 2008. What's a Typical Commit? A Characterization of Open Source Software Repositories. In *2008 16th IEEE International Conference on Program Comprehension*. 182–191. https://doi.org/10.1109/ICPC.2008.24

[2] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge. 2018. How Do Developers Discuss Rationale?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 357–369. https://doi.org/10.1109/SANER.2018.8330223

[3] Daniel Amyot. 2003. Introduction to the User Requirements Notation: learning by example. *Computer Networks* 42, 3 (2003), 285 – 301. https://doi.org/10.1016/S1389-1286(03)00244-5 ITU-T System Design Languages (SDL).

[4] A. I. Anton. 1996. Goal-Based Requirements Analysis. In *Proceedings of the Second International Conference on Requirements Engineering*. 136–144. https://doi.org/10.1109/ICRE.1996.491438

[5] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What Makes a Good Bug Report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16)*. ACM, New York, NY, USA, 308–318. https://doi.org/10.1145/1453101.1453146

[6] Patrick Biernacki and Dan Waldorf. 1981. Snowball Sampling: Problems and Techniques of Chain Referral Sampling. *Sociological Methods & Research* 10, 2 (1981), 141–163. https://doi.org/10.1177/004912418101000205 arXiv:https://doi.org/10.1177/004912418101000205

[7] Alexander W.J. Bradley and Gail C. Murphy. 2011. Supporting Software History Exploration. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. ACM, New York, NY, USA, 193–202. https://doi.org/10.1145/1985441.1985469

[8] Janet E. Burge and David C. Brown. 2008. Software Engineering Using RATionale. *Journal of Systems and Software* 81, 3 (2008), 395 – 413. https://doi.org/10.1016/j.jss.2007.05.004 Selected Papers from the 2006 Brazilian Symposia on Databases and on Software Engineering.

[9] Janet E Burge, John M Carroll, Raymond McCall, and Ivan Mistrik. 2008. *Rationale-Based Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77583-6

[10] Raymond P.L. Buse and Westley R. Weimer. 2010. Automatically Documenting Program Changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '10)*. ACM, New York, NY, USA, 33–42. https://doi.org/10.1145/1858996.1859005

[11] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey. 2015. Software History under the Lens: A Study on Why and How Developers Examine It. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 1–10. https://doi.org/10.1109/ICSM.2015.7332446

[12] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk. 2014. On Automatically Generating Commit Messages via Summarization of Source Code Changes. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. 275–284. https://doi.org/10.1109/SCAM.2014.14

[13] Steven Davies and Marc Roper. 2014. What's in a Bug Report?. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14)*. ACM, New York, NY, USA, Article 26, 10 pages. https://doi.org/10.1145/2652524.2652541

[14] Allen H. Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech. 2006. *Rationale Management in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30998-7

[15] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. 2018. Communicative Intention in Code Review Questions. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 519–523. https://doi.org/10.1109/ICSME.2018.00061

[16] Thomas Fritz and Gail C. Murphy. 2010. Using Information Fragments to Answer the Questions Developers Ask. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE '10)*. ACM, New York, NY, USA, 175–184. https://doi.org/10.1145/1806799.1806828

[17] Enio G Jelihovschi and José Faria. 2014. ScottKnott: A Package for Performing the Scott-Knott Clustering Algorithm in R. *TEMA (São Carlos)* 15 (03 2014). https://doi.org/10.5540/tema.2014.015.01.0003

[18] Fabian Gilson and Vincent Englebert. 2011. Rationale, Decisions and Alternatives Traceability for Architecture Design. In *Proceedings of the 5th European Conference on Software Architecture: Companion Volume (ECSA '11)*. ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/2031759.2031764

[19] Raman Goyal, Gabriel Ferreira, Christian Kästner, and James Herbsleb. 2018. Identifying unusual commits on GitHub. *Journal of Software: Evolution and Process* 30, 1 (2018), e1893. https://doi.org/10.1002/smr.1893 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1893 e1893 smr.1893.

[20] Thomas R. Gruber and Daniel M. Russell. 1996. Design Rationale. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, Chapter Generative Design Rationale: Beyond the Record and Replay Paradigm, 323–349. http://dl.acm.org/citation.cfm?id=261685.261725

[21] Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. 2017. Trade-offs in Continuous Integration: Assurance, Security, and Flexibility. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 197–207. https://doi.org/10.1145/3106237.3106270

[22] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt. 2009. Automatic Classication of Large Changes into Maintenance Categories. In *2009 IEEE 17th International Conference on Program Comprehension*. 30–39. https://doi.org/10.1109/ICPC.2009.5090025

[23] ITU-T. 2018. User Requirements Notation (URN) – Language definition. http://handle.itu.int/11.1002/1000/13711

[24] A. P. J. Jarczyk, P. Loffler, and F. M. Shipmann. 1992. Design Rationale for Software Engineering: A Survey. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Vol. ii. 577–586 vol.2. https://doi.org/10.1109/HICSS.1992.183309

[25] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically Generating Commit Messages from Diffs Using Neural Machine Translation. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 135–146. http://dl.acm.org/citation.cfm?id=3155562.3155583

[26] S. Jiang and C. McMillan. 2017. Towards Automatic Generation of Short Summaries of Commits. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 320–323. https://doi.org/10.1109/ICPC.2017.12

[27] Siyuan Jiang, Collin McMillan, and Raul Santelices. 2017. Do Programmers do Change Impact Analysis in Debugging? *Empirical Software Engineering* 22, 2 (01 Apr 2017), 631–669. https://doi.org/10.1007/s10664-016-9441-9

[28] H. Kaiya, H. Horai, and M. Saeki. 2002. AGORA: Attributed Goal-Oriented Requirements Analysis Method. In *Proceedings IEEE Joint International Conference on Requirements Engineering*. 13–22. https://doi.org/10.1109/ICRE.2002.1048501

[29] A. Ko and B. Myers. 2008. Debugging Reinvented. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. 301–310. https://doi.org/10.1145/1368088.1368130

[30] A. J. Ko, R. DeLine, and G. Venolia. 2007. Information Needs in Collocated Software Development Teams. In *29th International Conference on Software Engineering (ICSE'07)*. 344–353. https://doi.org/10.1109/ICSE.2007.45

[31] Asher Koriat, Morris Goldsmith, and Ainat Pansky. 2000. Toward a Psychology of Memory Accuracy. *Annual Review of Psychology* 51, 1 (2000), 481–537. https://doi.org/10.1146/annurev.psych.51.1.481 arXiv:https://doi.org/10.1146/annurev.psych.51.1.481 PMID: 10751979.

[32] W. Kunz and H.W.J. Rittel. 1970. *Issues as Elements of Information Systems*. Number 131 in California. University. Center for Planning and Development Research. Working paper, no. 131. Institute of Urban and Regional Development, University of California. https://books.google.com/books?id=B-MaAQAAMAAJ

[33] Z. Kurtanović and W. Maalej. 2017. Mining User Rationale from Software Reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 61–70. https://doi.org/10.1109/RE.2017.86

[34] A. Van Lamsweerde. 2001. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*. 249–262. https://doi.org/10.1109/ISRE.2001.948567

[35] Thomas D. LaToza and Brad A. Myers. 2010. Hard-to-answer Questions About Code. In *Evaluation and Usability of Programming Languages and Tools (PLATEAU '10)*. ACM, New York, NY, USA, Article 8, 6 pages. https://doi.org/10.1145/1937117.1937125

[36] Jintae Lee and Kum-Yew Lai. 1991. What's in Design Rationale? *Hum.-Comput. Interact.* 6, 3 (Sept. 1991), 251–280. https://doi.org/10.1207/s15327051hci0603&4_3

[37] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk. 2015. ChangeScribe: A Tool for Automatically Generating Commit Messages. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 709–712. https://doi.org/10.1109/ICSE.2015.229

[38] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 31 (Sept. 2014), 37 pages. https://doi.org/10.1145/2622669

[39] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information Needs in Contemporary Code Review. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 135 (Nov. 2018), 27 pages. https://doi.org/10.1145/3274404

[40] R. Pohl and R.F. Pohl. 2004. Confirmation bias. In *Cognitive Illusions: A Handbook on Fallacies and Biases in Thinking, Judgement and Memory*. Psychology Press, Chapter 4, 79–96. https://books.google.com/books?id=k5gTes7yyWEC

[41] C. Potts and G. Bruns. 1988. Recording the Reasons for Design Decisions. In *Proceedings of the 10th International Conference on Software Engineering (ICSE '88)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 418–427. http://dl.acm.org/citation.cfm?id=55823.55863

[42] Sarah Rastkar and Gail C. Murphy. 2013. Why Did This Code Change?. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 1193–1196. http://dl.acm.org/citation.cfm?id=2486788.2486959

[43] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How Do Professional Developers Comprehend Software?. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 255–265. http://dl.acm.org/citation.cfm?id=2337223.2337254

[44] Christoffer Rosen, Ben Grawi, and Emad Shihab. 2015. Commit Guru: Analytics and Risk Prediction of Software Commits. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 966–969. https://doi.org/10.1145/2786805.2803183

[45] Janice Singer, Susan E. Sim, and Timothy C. Lethbridge. 2008. *Guide to Advanced Empirical Software Engineering*. Springer London, London. https://doi.org/10.1007/978-1-84800-044-5

[46] Davide Spadini, Maurício Aniche, Margaret-Anne Storey, Magiel Bruntink, and Alberto Bacchelli. 2018. When Testing Meets Code Review: Why and How Developers Review Tests. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 677–687. https://doi.org/10.1145/3180155.3180192

[47] D. Spencer and J.J. Garrett. 2009. *Card Sorting: Designing Usable Categories*. Rosenfeld Media. https://books.google.com/books?id=_h4D9gqi5tsC

[48] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. 2006. A survey of architecture design rationale. *Journal of Systems and Software* 79, 12 (2006), 1792 – 1804. https://doi.org/10.1016/j.jss.2006.04.029

[49] Antony Tang, Yan Jin, and Jun Han. 2007. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software* 80, 6 (2007), 918 – 934. https://doi.org/10.1016/j.jss.2006.08.040

[50] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. 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 (FSE '12)*. ACM, New York, NY, USA, Article 51, 11 pages. https://doi.org/10.1145/2393596.2393656

[51] Stephen E. Toulmin. 2003. *The Uses of Argument* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511840005

[52] J. Tyree and A. Akerman. 2005. Architecture decisions: demystifying architecture. *IEEE Software* 22, 2 (March 2005), 19–27. https://doi.org/10.1109/MS.2005.27

[53] Sai Zhang and Michael D. Ernst. 2014. Which Configuration Option Should I Change?. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 152–163. https://doi.org/10.1145/2568225.2568251