



Technische Universität München
Fakultät für Informatik
Lehrstuhl für Angewandte Softwaretechnik



Rationale in Developers' Communication

Rana Mohammed A Alkadhi

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Nassir Navab, Ph.D.
Prüfer der Dissertation: 1. Univ.-Prof. Bernd Bruegge, Ph.D.
2. Univ.-Prof. Dr. Barbara Paech
Universität Heidelberg

Die Dissertation wurde am 18.06.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 04.09.2018 angenommen.

Rana Alkadhi: *Rationale in Developers' Communication,*

To my sister Nada (1984 – 2015).

“If there ever comes a day when we can’t be together, keep me in your heart. I’ll stay there forever.” —A.A. Milne, Winnie-the-Pooh

Abstract

Developers make various decisions during software development. Rationale is the justification behind decisions, including the raised issues, the proposed alternative solutions, and the arguments for or against these alternatives. Rationale is of great importance during software maintenance and evolution. It helps developers understand the intent behind past decisions. Moreover, rationale supports software system documentation, software comprehension, and enhances software artifacts traceability and change impact analysis.

However, developers often resist capturing rationale in practice due to the intrusiveness of rationale capturing, as this activity is not fully integrated into current software engineering practices. Additionally, many rationale capturing approaches are too formal and require heavy human involvement to be acceptable by software developers. As a consequence, rationale remains implicit in the head of software developers or embedded in developers' communication and development artifacts.

This dissertation aims to: (i) understand how developers discuss rationale in text-based communication channels during software development, and (ii) support software developers in capturing rationale by developing capturing methods integrated into these channels.

To this end, we conducted three empirical studies to better understand how developers discuss rationale in two communication channels: chat messages and issue tracking systems, in co-located as well as distributed development teams. We found that developers' communications are valuable sources of rationale during software development. To support developers in capturing rationale, we present REACT, a lightweight manual method to identify rationale elements in the chat messages exchanged by developers. We evaluated REACT in two studies and found that REACT is easy to learn and simple to apply. However, developers still perceive identifying rationale elements as a cognitive overhead while communicating over chat messages. To address this problem, we devel-

oped A-REACT, an automated method for detecting and classifying rationale in developers' communications. We evaluated A-REACT on three communication artifacts and found that it has a good performance in identifying communication artifacts containing rationale, with a recall up to 0.99 and a precision up to 0.92.

This dissertation shows that text-based developers' communication is a valuable source of rationale and with the automation support it can be used successfully to capture rationale during software development.

Acknowledgements

First and foremost, I would like to thank the almighty God for giving me the opportunity to pursue my studies, the strength to chase my dreams, and the patience to keep going through hard times.

I would like to express my deep gratitude to my supervisor Bernd Bruegge, for his guidance, generosity, and continuous support throughout the entire journey of my studies. I had the pleasure to work under his supervision for my master and doctoral studies. His enthusiasm, immense knowledge, and confidence in me have tremendous impact in my research skills. My sincere thanks also go to my second supervisor Barbara Paech, for her insightful comments and wisdom, and for the interesting discussions during CURES workshops. Her valuable feedback and profound knowledge of rationale management and empirical research methods have greatly enriched my dissertation.

Special thanks to Dennis Pagano, my master thesis advisor and my mentor during doctoral studies. From the early stages of this research, he always gives time to discuss the next steps of my research and constantly gives feedback. I am deeply indebted to his continuous support, encouragement, for reviewing parts of this dissertation, and for being there when times were a bit tough. I am also grateful to my co-author Emitzá Guzmán, from whom I learned a lot about the research community and how to be passionate about research. Our research collaborations, her constant help and feedback have played a fundamental part in this work.

Thank you to the current and former members of the Chair for Applied Software Engineering. Being part of this supportive work environment has been an invaluable experience. Special thanks go to my co-author Jan Ole Johansen, I have enjoyed and learned so much from our collaborations and deep delightful discussions. Thank you for reviewing parts of this dissertation and for the Kinder Chocolate that made long working days sweeter. I will miss our coffee break walks! Thank you Hoda Naguib for the moral support since the begin-

ning of my doctoral studies, it is always a joy to have you around. Thank you Stefan Nosović, Nitesh Narayan, Yang Li, and Stephan Krusche for being ready to help and answer my various questions. Thank you Zardosht Hodaie, and Sajjad Taheri for reviewing parts of this dissertation. I would also like to extend my thanks to Monika Markl and Helma Schneider for all the administrative and technical support I was in need during my studies.

I also had great pleasure of co-supervising the master and bachelor theses of motivated students. Many thanks to Teodora Lata, Manuel Nonnenmacher, Sebastian Ober, Ankur Sinha, and Diane Xhymshiti. Working with you has been a very enriching experience.

I would like to thank King Saud University and the Ministry of Higher Education of Saudi Arabia for the scholarship and financial support during my graduate studies.

I would like to express my thanks to my dear friends, Nadine, Dalal, Ahad, Hessa, and Madawi. Thank you for being always close by and for being my family here in Munich. I will always cherish the time we spent together.

Most importantly, without the support and nurturing of my family, the completion of this dissertation would not have been possible. A very special gratitude goes out to my parents, Huda and Mohammed, my first and constant believers. You taught me to strive to be a better person and to always look forward with determination. Thank you for taking care of my daughter to help me pursue my studies. I would not have made it this far without your unconditional love and support, I owe it all to you!

My love and affectionate gratitude to my husband Ammar. Thank you for staying by my side and for your patience when research required most of my time. Great love and thanks go to my lovely daughter Alanoud for bearing to be away from me through the past two years. Thank you for all the love, the joy, and the warmth you two bring into my life!

I would like to thank my grandparents, especially my grandfather Saleh for his care, support, and encouragement since the beginning of my graduate studies; I am so grateful Papa Saleh!

I cannot fully express my thanks to my sister Bushra, you cannot imagine how much strength your support has given me during difficult times and for reviewing large parts of this dissertation. Special thanks to my sisters and brothers,

Ruba and her husband Fahad, Abdulrahman, Mzoon, Abdulaziz, and Reema, and my nephew Suliman. It is your love, unwavering support, and encouragement that have made the hardship of completing this dissertation bearable. Grateful to you for always being there!

Contents

I	INTRODUCTION AND FOUNDATIONS	1
1	Introduction	3
1.1	Research Approach	8
1.2	Scope	10
1.3	Dissertation Structure	11
1.4	Publications	12
2	Foundations	13
2.1	Rationale Definition	13
2.2	Design Rationale Approaches	16
2.2.1	Rationale Capture	17
2.2.2	Rationale Representation	19
2.2.3	Rationale Usage	26
2.3	Text Mining Fundamentals	27
II	ANALYZING RATIONALE IN TEXT-BASED DEVELOPERS' COMMUNI- CATIONS	33
3	Rationale in Chat Messages of Co-located Teams	35
3.1	Study Design	35
3.1.1	Research Questions	36
3.1.2	Research Data	36
3.1.3	Research Method	37
3.2	Results	40
3.2.1	Rationale Frequency	40
3.2.2	Rationale Completeness	44
3.3	Discussion	45
3.4	Threats to Validity	45
4	Rationale in Text-based Developers' Communications of Distributed Teams	49

4.1	Research questions	50
4.2	Rationale in Developers' Chat Messages	51
4.2.1	Study Design	51
4.2.2	Results	58
4.2.3	Discussion	65
4.2.4	Threats to Validity	66
4.3	Rationale in Developers' Comments in Issue Tracking Systems	68
4.3.1	Study Design	70
4.3.2	Results	75
4.3.3	Discussion	79
4.3.4	Threats to Validity	80
5	Related Work Relevant to Analyzing Text-based Developers' Communications	83
5.1	Analyzing Developers' Chat Messages	83
5.2	Analyzing Issue Tracking Systems	86
III RATIONALE CAPTURING METHODS IN TEXT-BASED DEVELOPERS' COMMUNICATIONS		89
6	REACT: A Method for Capturing Rationale in Developers' Chat Messages	91
6.1	REACT Method	91
6.2	REACT Evaluation	93
6.2.1	Study 1: REACT in a short-term Design Task	95
6.2.2	Study 2: REACT in a medium-term Project	98
6.2.3	Questionnaire	102
6.3	Discussion	103
6.4	Threats to Validity	104
7	A-REACT: An Automated Rationale Extraction Method	107
7.1	A-REACT Method	108
7.2	Evaluation	110
7.2.1	Chat Messages of Co-located Teams	110
7.2.2	Chat Messages of Distributed Teams	117
7.2.3	Comment in Issue Tracking Systems of Distributed Teams	122
7.3	Discussion	128
7.4	Threats to Validity	129

8	Related Work Relevant to Rationale Annotation and Automated Capturing Approaches	131
8.1	Rationale Annotation Approaches	131
8.2	Automated Extraction of Rationale	135
IV	CONCLUSION	141
9	Conclusion and Future Work	143
9.1	Contributions	144
9.2	Future Work	147
V	APPENDICES	153
A	Coding Guide: For Annotating Rationale Elements in Developers' Chat Messages of Co-located Teams	155
B	Coding Guide: For Annotating Rationale Elements in Developers' Chat Messages of Distributed Teams	159
C	Coding Guide: For Annotating Rationale Elements in Issue Tracking Systems	163
D	Questionnaire for Evaluating REACT	165
	BIBLIOGRAPHY	167

List of Figures

Figure 2.1	The goals and principles of the by-product approach to capture rationale according to Schneider [168].	18
Figure 2.2	The IBIS model (adapted from Kunz and Rittel [102]).	21
Figure 2.3	An example of a simple issue deliberation using itIBIS (adapted from Burgess Yakemovic and Conklin [26]), where “I” refers to issues, “P” to positions, “AS” to supporting arguments, “AO” to objecting arguments, “?” to open issues, “*” to resolved issues or selected positions, and “-” to rejected positions.	22
Figure 3.1	A screenshot of using GATE for the manual coding of chat messages. (1) The main window displays the list of (anonymized) chat messages to be annotated, (2) when a coder highlights a part of a message that contains rationale, a pop-up window appears (the Annotation Editor Dialog) where the coder can specify the rationale element(s) present in the message, and (3) the color codes for different rationale elements.	38
Figure 3.2	Chat messages containing rationale per team.	41
Figure 3.3	Distribution of all messages as well as messages containing rationale over the duration of the project.	43
Figure 4.1	Applied research method for studying rationale in developers’ chat messages of distributed teams.	52
Figure 4.2	IRC messages containing rationale per project.	59
Figure 4.3	Pair-wise correlation matrices of rationale elements in IRC messages per project. The cells shading and color intensity visualize the sign and magnitude of the correlation.	61
Figure 4.4	The percentage of messages containing rationale written by IRC committers and IRC non-committers.	63

Figure 4.5	Rationale elements distribution per project.	64
Figure 4.6	An example of a reported issue in Ubuntu. It consists of: (1) issue title, (2) issue metadata, e.g., status, importance, reporter, and assignee, (3) issue description, and (4) com- ments.	69
Figure 4.7	A screenshot of the Excel sheet as used during the manual coding of comments in ITS.	73
Figure 4.8	Comments containing rationale per project.	75
Figure 4.9	Pair-wise correlation matrices of rationale elements in ITS comments per project. The cells shading and color inten- sity visualize the sign and magnitude of the correlation.	77
Figure 6.1	Example of a conversation in Slack. Teams using Slack can create (1) <i>channels</i> to organize their conversations accord- ing to topics, team members can include emojis (2) <i>inline</i> within messages, or as (3) a <i>reaction</i> , and (4) the added reactions appear under the message.	93
Figure 6.2	REACT evaluation method.	94
Figure 6.3	Correctness analysis.	96
Figure 6.4	Collaborativeness analysis.	97
Figure 6.5	REACT rationale annotations pinned to team Slack chan- nels in Study 2.	99
Figure 6.6	Rationale in team messages of Study 2.	101
Figure 6.7	Privacy analysis.	103
Figure 7.1	Overview of A-REACT.	108
Figure 9.1	Extension mockup: based on detected rationale elements in the message (1), rationale annotations are suggested (2).	148

List of Tables

Table 2.1	Definitions of rationale elements used in this dissertation (adapted from Bruegge and Dutoit [16]).	25
Table 3.1	Overview of analyzed chat messages.	37
Table 3.2	Frequency distribution of rationale elements across messages containing rationale per team.	42
Table 4.1	Overview of IRC messages.	53
Table 4.2	Overview of commit messages.	54
Table 4.3	Alias resolution in IRC authors and Committers.	58
Table 4.4	Frequency distribution of rationale elements across messages containing rationale per project.	60
Table 4.5	IRC authors in the analyzed sample of IRC messages.	62
Table 4.6	Overview of Issue Tracking Systems Dataset.	71
Table 4.7	Issues coding sample.	72
Table 4.8	Frequency distribution of rationale elements across ITS's comments containing rationale per project.	76
Table 4.9	The results of mapping commenters in the study sample to committers.	78
Table 6.1	Frequency distribution of REACT rationale annotations and examples of annotated messages.	95
Table 6.2	Chat messages analyzed in Study 2.	100
Table 7.1	Binary classification results of chat messages of co-located teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	112
Table 7.2	Examples of binary classification results of chat messages in co-located teams.	113
Table 7.3	Project cross validation results of chat messages of co-located teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	114

Table 7.4	Fine-grained classification results of chat messages of co-located teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	115
Table 7.5	Examples of fine-grained classification results of chat messages in co-located teams.	116
Table 7.6	Binary classification results of chat messages of distributed teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	118
Table 7.7	Examples of binary classification results of chat messages of distributed teams.	119
Table 7.8	Project cross validation results of chat messages of distributed teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	120
Table 7.9	Fine-grained classification results of chat messages of distributed teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	121
Table 7.10	Examples of fine-grained classification results of chat messages of distributed teams.	122
Table 7.11	Binary classification results of comments in issue tracking systems of distributed teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	123
Table 7.12	Examples of binary classification results of comments in issue tracking systems of distributed teams.	124
Table 7.13	Project cross validation results of comments in issue tracking systems of distributed teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	125
Table 7.14	Fine-grained classification results of comments in issue tracking systems of distributed teams. P is Precision, R is Recall, and F ₁ is F ₁ -measure.	126
Table 7.15	Examples of fine-grained classification results of comments in issue tracking systems of distributed teams.	127

Part I

Introduction and Foundations

Introduction

“An investment in knowledge always pays the best interest.”

—Benjamin Franklin [59]

In the 1960s, Horst Rittel coined the term *wicked* problems to refer to ill-defined design problems that are difficult to solve due to incomplete, contradictory, and frequently changing requirements [153], [155]. Multiple stakeholders with different, possibly conflicting, values and needs are involved. Wicked problems cannot be solved algorithmically, but rather need discussions and creative solutions, in contrast to *tame* or *benign* problems [155] that can be solved using conventional analytical methods. Wicked problems have no right or wrong solution, but rather a specific solution in a particular context. This solution might not work in the future with the emergence of new requirements and the need to deal with technological change.

Rittel and Webber [155] argue that tackling wicked problems requires an argumentative approach during which a definition of the problem and the solution evolves gradually and collaboratively among the participants of the problem solving group. As Rittel described in an interview about the future of design methodologies:

“My recommendation would be to emphasize investigations into the understanding of designing as an argumentative process [...] how to understand designing as a counterplay of raising issues and dealing with them, which in turn raises new issues, and so on.

[Argumentative] means the generation of solution specifications towards end statements, and subjecting them to discussion of their pros and cons.” [154]

To the aim of supporting argumentative approaches during design, Kunz and Rittel [102] developed IBIS (Issue-Based Information System) to facilitate the capturing of argumentation. IBIS organizes the discussions of design problems around the deliberation of issues. During the deliberation, *issues* representing questions to solve are raised, *positions* to solve the issues are proposed, and *arguments* supporting or objecting to proposed positions are given. An issue is resolved by selecting a position based on argumentation [126]. The argumentation leading to the decision, i.e., the resolution of an issue, forms the *rationale* behind design decisions.

The use of argumentative approaches to tackle design problems spread into other fields involved with design, including software engineering. In the 1980s, Conklin and Begeman [35], [36] adapted IBIS to be used in software engineering. They noticed that the characteristics of wicked problems are evident in the problems facing software developers [40], [47]. Software developers are faced with complex, evolving, and changing requirements. Moreover, satisfying multiple stakeholders with different needs and expectations plays a critical role in the success of the software product.

Following an argumentative approach, *wicked* design problems in software engineering are resolved through discussions, iterations, and accepting change as a normal part of the process [146]. The justification behind decisions, the other alternatives considered, and the argumentation that led to the decision constitute the rationale [110]. Rationale is an explanation of *why* the system is designed the way it is [116], [151].

Captured rationale is a type of developer documentation, an umbrella term recently coined by Robillard et al. [156] for documents intended to assist developers in the creation and maintenance of a software system. It is considered among the most useful information for developers during software maintenance [112]. The availability of rationale improves the developers' understanding of the system; thus, making it easier to maintain as developers usually seek to understand the system prior to making changes [176]. It helps developers understand the intent behind past decisions [97]. Time and development efforts can be saved by documenting alternatives visited and rejected earlier [47]. In addition, captured rationale enhances software artifacts traceability, change impact analysis, facilitates design verification, and knowledge sharing [24], [98]. Capturing ra-

tionale helps to avoid knowledge vaporization, i.e., the problem of knowledge getting lost when experts leave an organization [28], and supports knowledge acquisition for the newcomers [92].

However, despite the broad consensus on the importance of rationale and the suggestion of many capturing approaches in software engineering, developers often resist capturing rationale in practice. This problem is commonly known in the literature as the *capture problem* [47]. Roehm et al. [159] found that rationale is important information during program comprehension tasks, but it is rarely documented. In their observation of 28 professional software developers, the authors did not observe a single developer documenting rationale for their own code. In another study of developers' information needs conducted by Ko et al. [97], one developer shared their experience: "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".

There are a number of possible causes for the capture problem. Kruchten et al. [98] argue that the adoption barrier for capturing rationale is the intrusiveness of the capturing approaches, as they are not fully integrated into current software engineering practices—i.e., the developers must switch tools to document their rationale. Another primary cause for the developers' reluctance is the additional overhead involved in capturing rationale. Many approaches to capture rationale are too formal and require heavy human involvement to be adopted by software developers in practice. As a consequence, rationale remains implicit in the head of software developers or embedded in development and communication artifacts, and eventually lost over time [47], [119].

Missing rationale information has negative impacts on software development, as development tasks might have to be deferred because of missing knowledge on system design and behavior, e.g., the reasons for the current implementation [97]. The lack of documented rationale forces developers to invest great efforts in recovering implicit rationale knowledge by exploring code and interrupting their teammates, which was found to have negative impacts on their productivity [95], [105]. Knowledge vaporization resulting from undocumented rationale can have effects even on the personal level. In a survey with software

engineering experts from six different companies conducted by Miesbauer and Weinreich [132], about half of the participants reported that they forgot the decisions they made themselves. Furthermore, in a field characterized by developing long-living software systems and a high rate of staff turnover, software organizations face the risk of losing their institutional knowledge if rationale is not captured. In 2017, software industry had the highest turnover rate than any other sector according to the professional social network LinkedIn¹. To retain this important knowledge, rationale need to be captured and externalized from the developers' memory and development artifacts to facilitate the sharing and transferring of this knowledge.

One way for addressing the rationale capture problem is by developing capturing approaches that are integrated into the developers' activities; thus, less intrusive and more likely to be adopted by software developers. To this end, sources of rationale during software development need to be identified and capturing techniques integrated into these sources need to be designed [160]. Considering that rationale emerges from developers' deliberation of issues, discussion of the pros and cons of different alternatives, and the collaborative decision making, developers' communications form a rich source for valuable information about the software system and its rationale [24], [189], [192]. As Seaman describes, "software developers reveal their thought processes most naturally when communicating with other software developers, so this communication offers the best opportunity for a researcher to observe the development process" [169].

Software developers use many different communication channels to communicate with other developers. In addition to traditional channels (e.g., face-to-face communication), text-based communication channels (e.g., chat, issue trackers, and mailing lists) play a pivotal role in software development activities [117], [181]. Previous research has found that written developers' communications are important for maintaining general awareness, sharing knowledge, and coordinating development activities, especially in geographically distributed teams [65], [180]. These communications have been used to perform bug triaging [5], recommending mentors [27], mining source code descriptions [8], [141], and mining developers' purpose of the communication,

¹ <https://business.linkedin.com/talent-solutions/blog/trends-and-research/2018/the-3-industries-with-the-highest-turnover-rates>

such as opinion asking, proposing a new feature, or reporting a bug [43]. However, little is known about how developers discuss rationale in written communications. This lack of knowledge makes it difficult to develop useful capturing approaches of rationale from these channels.

The goal of this dissertation is to investigate how developers discuss rationale in text-based communication channels and to devise methods to support capturing rationale from these channels. To achieve this goal, we make three contributions. First, we present three empirical studies to better understand how developers discuss rationale over two text-based communication channels: chat messages and issue tracking systems. Our focus on these two channels is motivated by their increasing popularity and significance as communication channels during software development [4], [66], [90]. This understanding is essential for building effective rationale capturing tools in written communication channels. The results of these studies provide quantitative evidence that text-based developers' communications are valuable sources of rationale during software development. Furthermore, they provide deeper insights about the nature of rationale in written communications that can aid future research in exploiting these communications as a source of rationale. Second, we present REACT (Rationale ExtrAction from Communication arTifacts), a lightweight manual method to capture rationale in the chat messages exchanged by developers. REACT can be used by developers for the explicit and collaborative capturing of rationale while communicating over chat channels. Furthermore, REACT can be integrated into most modern chat systems, which alleviate its intrusiveness and encourages its adoption by developers. Third, we present A-REACT (Automated Rationale ExtrAction from Communication arTifacts), an automated method to extract rationale from text-based developers' communications. We conducted a series of experiments to compare the performance of various supervised machine learning techniques and configurations in detecting and classifying rationale in chat messages and issue tracking systems. A-REACT can help developers extract rationale from existing communication archives and lower the overhead involved in the explicit capturing approaches of rationale. Furthermore, the results of our experiments can provide guidelines for researchers about the techniques and configurations that yield the most accurate results.

1.1 RESEARCH APPROACH

We followed an explorative empirical approach consisting of two phases: analyzing how developers discuss rationale over written communication channels, and developing rationale capturing methods integrated into these channels.

The first phase focuses on getting a deeper understanding of how developers discuss rationale in written communication channels. This understanding is essential to lay the foundation for an effective capturing of rationale from these channels. To this aim, we conducted three empirical studies in two different settings: co-located and distributed development teams.

We started exploring the question, *how do developers discuss rationale in chat messages?* To answer this question, we conducted the first empirical study on the chat messages of three co-located development teams. In the study, we investigated the frequency of the different rationale elements (i.e., issues, alternatives, pro-arguments, con-arguments, and decisions²), and the completeness of the rationale discussed in the analyzed chat messages. We performed manual content analysis as described by Neuendorf [138] on the chat messages exchanged by developers over the duration of the three projects. Although face-to-face communications are still dominant in co-located teams, we found that developers' chat messages contain valuable rationale. This led us to pose the following question, *how do developers discuss rationale in distributed teams, where developers are geographically distributed and rely heavily on written communications?*

To answer this question, we conducted the second empirical study on the Internet Relay Chat (IRC) channels of three open source projects developed and maintained by globally distributed teams [65]. We investigated the frequency of the different rationale elements and the rationale contributors (committers vs. non-committers) to analyze the correlation between development activities and the rationale contribution in IRC channels. Similar to the first study, we performed manual content analysis [138] on a random sample of IRC messages from the three projects.

During the manual analysis of the chat messages, we observed that developers frequently reference an already opened issues (e.g., by mentioning the issue ID), or they create new issues as a follow up to their discussion in the chat

² Rationale elements are defined in Table 2.1 in Chapter 2.

messages. This observation and the fact that issue tracking systems are typically used for managing and discussing issues in open source projects [4], [77] led us to pose the following question, *how do developers discuss rationale in a more focused discussion channel such as an Issue Tracking System (ITS)?*

In order to answer this question, we conducted the third empirical study on the issue tracking systems used in the three open source projects from the second study. Likewise, we investigated the frequency of the different rationale elements and the rationale contributors (committers vs. non-committers) in the developers' comments to issues. We performed manual content analysis [138] on the comments to a stratified random sample of issues from the three projects.

By the end of the first phase that is focused on the analysis of rationale in written communication channels, we provided quantitative evidence that developers' written communications are valuable sources of rationale during software development. However, previous research has found that when explicit documentation of important information such as rationale is missing, developers avoid browsing written repositories to obtain the needed information [105]. Thus, we realized that without capturing and externalizing the rationale from these communication artifacts, rationale remains implicit. This reflection led us to pose the following question, *how can we support developers to capture rationale from written communications?*

The second phase focuses on devising methods to capture rationale in written developers' communications. To this end, we designed REACT, a lightweight method to capture rationale in developers' chat messages. REACT was evaluated using both quantitative and qualitative measures in two studies: in short-term design task and in a medium-term project. We observed that identifying rationale elements in the chat messages to apply REACT annotations is still perceived by the developers as additional cognitive load and disruption to the development activities. This led us to think about an automated approach to lessen the capturing burden on developers, and hence to pose the following question, *can rationale in written communications be detected and classified automatically?*

In order to answer this question, we developed A-REACT, a method for the automated extraction of rationale on two levels of granularity: binary and fine-grained classification. Binary classification detects communication artifacts containing rationale, and fine-grained classification classifies the communication ar-

tifacts containing rationale into the different rationale elements. We used the three manually annotated datasets from the first phase, i.e., chat messages of co-located teams, IRC messages of distributed teams, and comments to issue tracking systems in distributed teams, for the training and evaluation of different supervised machine learning algorithms.

1.2 SCOPE

Rationale management in software engineering is a broad area. We limited the scope of this dissertation with respect to the following dimensions:

1. *Rationale capturing.* Rationale management in software engineering can be divided into three areas³: capturing, representation, and usage of rationale [47], [73]. Rationale capturing focuses on eliciting rationale during software development. Rationale representation focuses on structuring the captured rationale according to a rationale representation model. This might include the linkage of fragmented rationale elements across multiple sources. Finally, the rationale usage focuses on exploiting the rationale knowledge, creating usage scenarios, and making the captured rationale available for the use of development teams at later stages.

Capturing rationale has been long recognized as a major obstacle in investigating the applications and usage of rationale in real-world settings [47]. Thus, this dissertation focuses on capturing rationale from written developers' communications.

2. *Rationale elements.* Different models have been proposed and evaluated to represent rationale in software engineering and other disciplines⁴. Given the exploratory and empirical nature of the work presented in this dissertation, we focus our analysis on the basic elements of argumentation leading to decisions. Therefore, we focus on rationale elements adapted from IBIS, the issue model proposed by Kunz and Rittel [102] for its conciseness and because it provides the basis for most of the subsequent

³ These areas are discussed in Chapter 2.

⁴ Different representation models, such as QOC and DRL, are discussed in more depth in Chapter 2.

issue models. Namely, we analyze five elements of rationale: issues, alternatives, pro-arguments, con-arguments, and decisions⁵.

3. *Communication channels.* Developers use various channels to communicate during software development. They use verbal communication channels such as face-to-face meetings, conference calls, and private conversations. In addition, they use many text-based communication channels, such as chat, emails, and issue tracking systems. The usage of these text-based channels are seeing a growing and rapid adoption by software developers to communicate and coordinate their development activities [4], [90]. To this end, we focus on investigating written developers' communications as sources of rationale. In particular, we study chat messages and issue tracking systems.

1.3 DISSERTATION STRUCTURE

The rest of the dissertation is organized as follows: Chapter 2 introduces background information on rationale management in software engineering, including rationale definitions, approaches of rationale capturing, representation, and usage. In addition, the background information on text mining and classification techniques is presented. In Chapter 3, we present an empirical study to analyze how developers discuss rationale in the chat messages of three co-located teams. Chapter 4 describes two empirical studies to investigate how developers discuss rationale in distributed development teams by analyzing rationale in two communication channels, chat messages and issue tracking systems, of three open source projects. Chapter 5 discusses work related to the analysis of developers' written communications. Chapter 6 presents REACT, a lightweight method to support developers in capturing rationale when communicating over chat messages. We conducted two studies to evaluate REACT: in a short-term design task and in a medium-term project. Chapter 7 presents A-REACT, an automated method for detecting and classifying rationale in developers' communication artifacts. We report on the evaluation of A-REACT on three developers' communication artifacts: chat messages of co-located teams, chat message of

⁵ Rationale elements are defined in Table 2.1 in Chapter 2.

distributed teams, and developers' discussions on issue tracking systems. Chapter 8 discusses work related to rationale annotation and automated capturing approaches. Chapter 9 concludes the dissertation by summarizing the contributions of this work and identifying future work directions.

1.4 PUBLICATIONS

Parts of this dissertation have appeared previously in the following publications:

- [1] R. Alkadhi, J. O. Johanssen, E. Guzman, and B. Bruegge, "REACT: An Approach for Capturing Rationale in Chat Messages," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 175–180, (Best Paper Award).
- [2] R. Alkadhi, T. Lața, E. Guzman, and B. Bruegge, "Rationale in Development Chat Messages: An Exploratory Study," in *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 436–446.
- [3] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, "How Do Developers Discuss Rationale?" In *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 357–369.

Foundations

“At the risk of appearing to exaggerate, I will argue that the pursuit of rationale in engineering is nothing less than a search for meaning.”

—Bashar Nuseibeh [25]

Rationale management in software engineering is a broad research area. Many different approaches have been proposed to capture, represent, and make use of rationale during software development. In this chapter, we present background on rationale in software engineering. In addition, we present the fundamentals of text mining techniques that we apply for the automated extraction of rationale in Chapter 7.

We are going to discuss the work related to analyzing written developers' communications, and annotation and automated rationale capturing approaches in Chapter 5 and Chapter 8, respectively. This enables to discuss the work presented in this dissertation in the light of related work.

The chapter is structured as follows: Section 2.1 explores the different definitions of rationale used in literature. Section 2.2 discusses different approaches to rationale. In particular, we discuss rationale capture, rationale representation including different representation models, and rationale usage. Section 2.3 provides the foundation of text mining and classification techniques applied in this dissertation.

2.1 RATIONALE DEFINITION

Historically, research investigating rationale has focused on *Design Rationale (DR)*, which is defined by Lee and Lai [111] as “an explanation of why an artifact is designed the way it is”. However, decision-making is not restricted to a

specific phase of software development. Developers make decisions that affect the software design throughout the software lifecycle, ranging from requirement elicitation to design, implementation, and testing of the software system, and to its continuous maintenance during software evolution. To emphasize the central role of rationale during all activities of software engineering, Dutoit et al. [47] and Burge et al. [25] use the term *software engineering rationale* to refer to rationale during software development.

In this chapter, we use the terms *design rationale* and *rationale* interchangeably to cover the fundamentals of rationale. However, as this dissertation deals with rationale in the context of software engineering, we use the term *rationale* instead of *software engineering rationale* to refer to rationale during software development.

The term *design rationale* has been defined according to many view points over the years, as stated by Moran and Carroll [133], “It is easy to get confused, because the term is used in many different senses [...] and no one of them is standard in this stage of research on the topic”.

Moran and Carroll [133] introduced six flavors of design rationale definitions. They first define rationale as “(i) an expression of the relationships between a designed artifact, its purpose, the designer’s conceptualization, and the contextual constraints on realizing the purpose”. This definition uses the term rationale to refer to the reasons behind decisions as understood by the designers. The second definition, “(ii) the logical reasons given to justify a designed artifact”, implies that rationale is constructed for a specific purpose (e.g., to persuade a client), hence it looks at rationale as a form of justification. The third definition, “(iii) a notation for the logical reasons for a designed artifact”, defines rationale based on the representation used for capturing it¹. In the fourth definition, “(iv) a method of designing an artifact whereby the reasons for it are made explicit”, rationale is used as a design method to improve the reasoning process of the designers. The fifth definition, “(v) documentation of the reasons for the design of an artifact, the stages or steps of the design process, the history of the design and its context”, looks at rationale as a design documentation, which can range from documenting only the reasons behind decision, to document the process steps resulting in the design, or to a complete historical documentation of the design. Finally, the sixth definition, “(vi) an explanation of why a designed artifact (or

¹ Rationale representation is discussed in Section 2.2.2.

some feature of an artifact) is the way it is”, defines rationale as an explanation that is generated to answer a given question about the design.

Shipman and McCall [174] distinguish between three different, albeit overlapping, perspectives on design rationale. In the *argumentation perspective*, design rationale means “the argumentation—i.e., reasoning—that designers use in framing and solving problems.”. While in the *communication perspective*, design rationale means “capturing and retrieving naturally occurring communication—e.g., design discourse—among members of a project team”. Finally, in the *documentation perspective*, design rationale means “the documentation of information about design decisions: what decisions are made, when they are made, who made them, and why”.

Explaining the reasons and argumentation behind decisions is a shared notion in various definitions of rationale in literature. MacLean et al. [121] define design rationale as “a representation for explicitly documenting the reasoning and argumentation that make sense of a specific artifact”. Similarly, Dutoit et al. [47] define design rationale as “the reasoning that goes into determining the design of the artifact”.

Lee [110] states that rationale “can include not only the reasons behind a design decision but also the justification for it, the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision”. Burge et al. [25] define the term rationale as “the reasoning underlying the creation and use of artifacts”.

In line with the above definitions that refer to rationale as the reasoning and argumentation behind design decisions, we define rationale within the scope of this dissertation as follows:

Definition 1. Rationale is the justification behind decisions, including the issues discussed, the alternative solutions considered, the arguments for or against proposed alternatives, and the decision, i.e., the selected alternative, during software development.

2.2 DESIGN RATIONALE APPROACHES

Design rationale is an established research area. A large number of design rationale approaches have been devised in different design domains, such as software engineering and human-computer interaction, and there are many ways of categorizing these approaches. One way is to differentiate between *descriptive* and *perspective* approaches [47]. Descriptive approaches, as the name indicates, focus on describing the design thinking process of the designers. On the other hand, perspective approaches aim at improving the reasoning process of the designers. Conklin et al. [34] called them *process-oriented* and *structure-oriented* approaches, respectively.

Another way for categorizing design rationale approaches is according to their intrusiveness to the design process and the amount of participation required from designers to *more-intrusive* and *less-intrusive* approaches [47].

Furthermore, approaches to design rationale can be categorized into *argumentative* and *non-argumentative* approaches [47]. Argumentative approaches represent rationale as arguments structured in rhetorical steps [48], these approaches are discussed further in Section 2.2.2.1. The non-argumentative approaches argue that structuring rationale as argumentation may not be sufficient for evaluating different alternatives and for capturing all the relevant rationale information that might be needed later. A common non-argumentative approach is structuring rationale using the artifact structure rather than an argumentative scheme, e.g., by linking textual rationale to code fragments [168]. Lewis et al. [113] present a non-argumentative approach in which concrete examples of the user goals, *problems*, are used to describe the intended functionality of the system and to evaluate design alternatives. Another example of a non-argumentative approach is the generative design rationale approach proposed by Gruber and Russel [64]. The authors argue for a generative approach in which rationale is constructed and inferred, in response to information request, from the data collected earlier during the design process. This perspective of customizing the generated rationale according to individual information requests is analogous with a recent work by Robillard et al. [156], which advocates for a paradigm shift in supporting the information needs of developers through the automated generation of developer documentation.

In the following, we discuss three basic processes of design rationale approaches: capture, representation, and usage of rationale [47], [133].

2.2.1 Rationale Capture

Rationale capture is “the process of eliciting rationale from designers and recording it” [47]. Capturing rationale has long been an issue in software engineering. In consequence, many capturing approaches have been proposed in the literature. Regli et al. [151] observed that the proposed design rationale systems have applied both *automatic* capturing approaches, i.e., without user intervention, and approaches requiring *user-intervention*, i.e., the designers document rationale themselves. A more detailed categorization was identified by Lee [110] that categorizes rationale capturing approaches into five main categories, according to their degree of designers’ participation. In the following, we provide a brief description of each category:

RECONSTRUCTION In this approach, the rationale is created retrospectively after the system design has been completed. Developers or other individuals, e.g., a rationale manager, capture the rationale without using a design rationale system [110]. Reconstructing rationale allows a careful reflection of the complete design process and has the advantage of not interrupting the flow of the design activities. However, deferring the rationale capture until after the design has been completed and relying on the designers’ memory increase the risk of losing important rationale knowledge. An example of a system that implements this capturing approach is Hyper-Object Substrate (HOS) [174], [175] which uses a combination of hypermedia representation with knowledge-based features. HOS supports the retrospective capturing of rationale by attaching informal design communications, such as emails, as textual annotations to the artifact design. These captured communications can be converted over time into a more formal representation of the rationale.

RECORD-AND-REPLAY In this approach, the rationale is captured as it unfolds synchronously, e.g., via video recordings of design meetings, or asynchronously, e.g., via developers email discussions [110]. The main objective

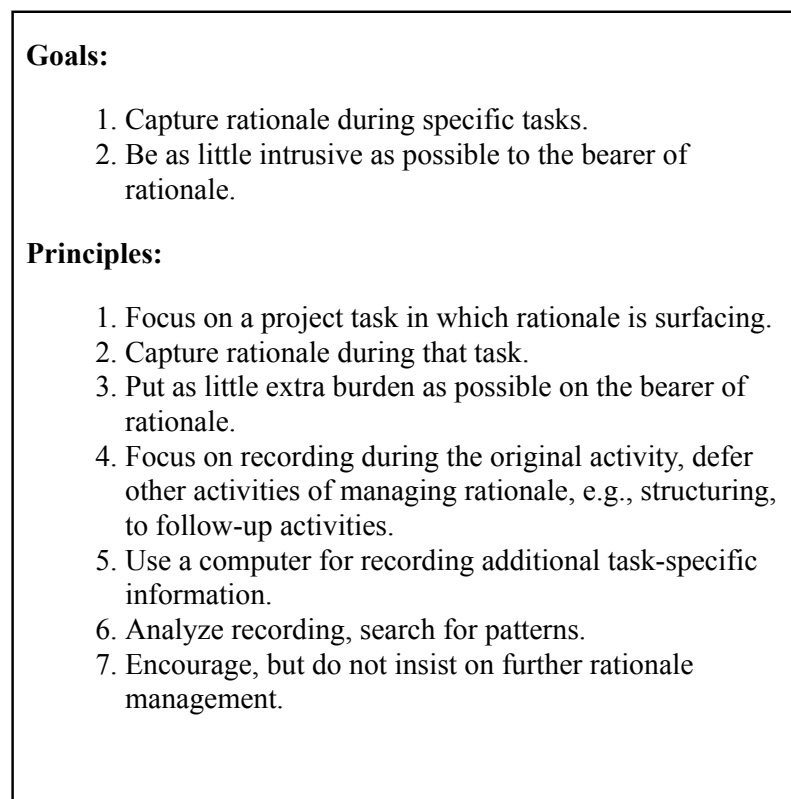


Figure 2.1: The goals and principles of the by-product approach to capture rationale according to Schneider [168].

of this approach is to capture rationale while it is elicited as part of the design communication in its informal form without disrupting the flow of the design activities.

RATIONALE AS A BY-PRODUCT In this approach, rationale emerges from the design process without imposing excessive overhead on developers [110]. Burge [21] argues that this can be done by either using design tools that automatically capture rationale as a side-effect or by using a design process that forces rationale capturing. Schneider [168] defines two goals and seven principles of the by-product approach to capture rationale, shown in Figure 2.1. An example of this approach is the Explainable Expert System (EES) approach [137] that uses the domain knowledge, the development history, and the execution history to produce the design rationale during the development of expert systems.

APPRENTICE In this approach, the system generates the rationale by monitoring the designer's actions and asks questions when it does not understand or disagrees with an action [110]. An example is Augmenting Design Documentation (ADD) [61] that acts as an intelligent apprentice to the designer. ADD learns about the features that make a design action different from the standard ones, and whenever a designer makes an action that differs from the ADD's expectations, it asks the designer for a justification. As a result, rationale is constructed using a combination of the system's domain knowledge and the justifications supplied by the designers.

AUTOMATIC GENERATION In this approach, rationale is generated automatically from the execution history [110]. For example, Rationale Construction Framework (RCF) [134] monitors designers interactions with a computer-aided design (CAD) tool to produce a rich design history, which is structured according to a theory of design metaphors to construct the rationale behind certain design aspects.

2.2.2 *Rationale Representation*

Rationale representation, also referred to as rationale formalization or rationale structuring, is "the process of transforming rationale into the desired representation form" [47]. Traditionally, the capturing and representation of rationale were carried out in a single activity. However, approaches proposed in recent years tend to split capturing and representing rationale into two consecutive activities.

The rationale capture and usage depend heavily on the selected representation of rationale, which can be classified into three broad categories [21], [110], [133]:

FORMAL REPRESENTATION In this representation, rationale is represented as objects and relations according to a formal language [110]. The main advantage of a formal representation is that it allows the interpretation and manipulation of rationale by computer systems. However, this representation may not be easily understood by a human. Another drawback is that formal representations are costly and might hinder the designers' creativity by imposing constraints on the design process [110].

INFORMAL REPRESENTATION In this representations, rationale is represented in unstructured form: design notes, audio/video recordings, and raw drawings [110]. Representing rationale informally allows the collection of a large volume of design information, however, the lack of structure makes this information difficult to parse and analyze.

SEMI-FORMAL REPRESENTATION In this representation, rationale is represented as argumentation, according to a predefined set of nodes, or rhetorical steps, and relationships between them. The rationale information of different nodes are often stored as pieces of natural language text [86]. The main advantage of a semi-formal representation is that it structures rationale in a form that is both understandable by humans and interpretable by computer systems. However, formal and semi-formal representations share the drawback that many aspects of the design may not be captured due to the predefined specification of the argumentation that constraint the type of collected information.

In the following section, we present some of the most common argumentative representation models of rationale.

2.2.2.1 *Rationale Representation Models*

There are two major classes of argumentative approaches of design rationale [47]. The first class of approaches are based on Tuolmin's model of argumentation [185]. In this model, an argument consists of a *claim* which is the main assertion being made (e.g., "9 out of 10 dentists surveyed use Crest."), *datum* which is a fact or observation that supports the claim (e.g., "Dentist's think Crest is the best toothpaste."), a *warrant* which is the basis on which datum is said to support the claim (e.g., "When given a choice, people use the toothpaste they think is best."), a *backing* that gives additional support to the warrant (e.g., "People do not want to get cavities."), and a *rebuttal* which is a counter-argument objecting to the claim (e.g., "Crest surveyed dentists they recently sent free samples to.") [86]. The Tuolmin's model of argumentation represents a single claim in isolation, relationships among different claims are not captured. The other class of approaches to structure design rationale argumentation either modified

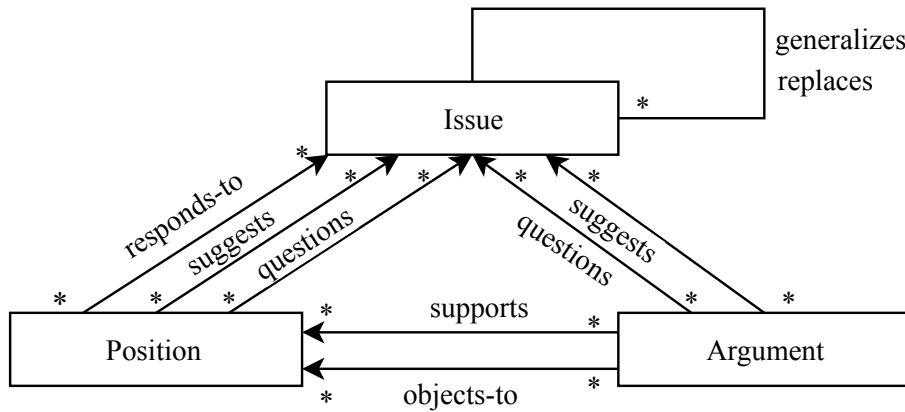


Figure 2.2: The IBIS model (adapted from Kunz and Rittel [102]).

the Kunz and Rittel's IBIS approach or created their own argumentation model. The use of the second class of approaches tends to be more dominant in the field of design rationale [47]. In the following, we discuss some of the prominent argumentative representation models of this class:

IBIS (ISSUE BASED INFORMATION SYSTEM) was developed by Kunz and Rittel [102] as an argumentative approach to tackle wicked design problems. IBIS structures rationale according to an issue model consisting of three key elements: *issues* that describe the problem under consideration, *positions* proposed to solve the problem, and *arguments* that support or object to the proposed positions. Figure 2.2 shows the nodes of the IBIS issue model and the relationships between them.

IBIS was the basis for most of the subsequent argumentative approaches to rationale. Conklin and Begeman [35], [36] adapted IBIS to be used in software engineering by creating gIBIS (graphical IBIS), a hypertext tool to support early design deliberations. gIBIS added two additional nodes, in addition to the original IBIS issue model: *other* for expressing additional thoughts, and *external* to allow the linkage of external material, such as requirement documents or design sketches.

Other extensions to IBIS include rIBIS (real-time IBIS) [152] and itIBIS (indented text) [26]. rIBIS extends gIBIS to allow distributed users to simultaneously browse and edit multiple views of an IBIS network; while itIBIS

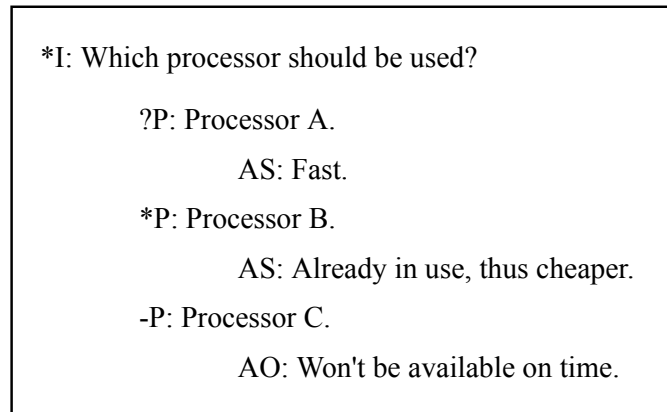


Figure 2.3: An example of a simple issue deliberation using itIBIS (adapted from Burgess Yakemovic and Conklin [26]), where “I” refers to issues, “P” to positions, “AS” to supporting arguments, “AO” to objecting arguments, “?” to open issues, “*” to resolved issues or selected positions, and “-” to rejected positions.

uses indentations to represent the hierarchical relationships between issue model elements using text editors available on personal computers. Figure 2.3 shows an example of an issue deliberation in itIBIS.

PHI (PROCEDURAL HIERARCHY OF ISSUES) is a refinement to IBIS developed by Rittel’s student McCall [123], [125]. It uses the same nodes of IBIS but replaces *positions* with *answers* to better reflect the terms used by the IBIS community. Its main addition was the reduction of the number of relationships between issues by introducing a *subissue* relationship to indicate that the resolution of one issue is dependent on the resolution of another issue. Examples of rationale systems that implement PHI are: MIKROPLIS [122], [124], which provides access to rationale as hypertext documents, and its successor PHIDIAS [126], which integrates CAD graphics to offer hypermedia-based computer-aided design system. Another example is JANUS [56] which uses a knowledge-based critic that alerts the users about the availability of rationale in relevant situations; thus, reducing the change of context required for using the captured rationale.

QOC (QUESTIONS, OPTIONS AND CRITERIA) was developed by MacLean et al. [121] as a semi-formal notation to represent the design space, i.e., the

set of possible alternative designs of an artifact, using three main components: *questions* identifying the key design issues, *options* providing possible answers to the questions, and *criteria* representing the bases for evaluating the different options.

The main difference between IBIS and QOC is that IBIS is considered to be more comprehensive as QOC can only represent features of the artifact being designed, which is a subset of the issues represented by IBIS. However, QOC is more expressive than IBIS as it allows the explicit representation of criteria [47].

DRL (DECISION REPRESENTATION LANGUAGE) is a language for representing the qualitative aspects of the decision making process [107], [108], [109], [111]. DRL has been implemented in SIBYL [108], a hypertext system to support the collaborative decision making. The primary elements of DRL are: *decision problems* that require a decision, *alternatives* proposed to address the decision problem, *goals* specifying the properties that should be satisfied by the selected alternative, and *claims* representing arguments for selecting alternatives. Other auxiliary objects in DRL includes *groups*, *procedures*, *viewpoints*, and *questions*. The main drawback of DRL is its complexity which increases the efforts required for structuring the captured rationale [16].

DRL differs from IBIS in that DRL adds an explicit representation of *design goals*, while goals are implicitly represented as *arguments* in IBIS. Another major difference to IBIS is that relations between rationale elements in DRL are represented as subclasses of *claim*, which allows arguing and questioning them as any other claims. Moreover, DRL has additional nodes for representing *procedures* and *viewpoints* [107].

The main difference between DRL and QOC is in their focus. While QOC is focused on describing the design space explored by the developers, DRL aims at capturing the deliberation leading to a decision [16], [166].

DECISION DOCUMENTATION MODEL has been proposed by Hesse and Paech [75], [78] and consists of two main components: *Knowledge Elements*

for representing general development artifacts, and *Decision Knowledge Elements* for modeling particular aspects of the decision knowledge.

The basic element is *decision*. All the elements related to decision knowledge are added as *decision components*. A decision component can be a *problem*, its *solution*, *context*, or *rationale*. *Issue* or *goal* are used to document details of the problem under consideration. *Alternatives* and *claims* represent the different solutions to the decision problem. Contextual information is documented as *assumptions* influencing the decision, *constraints* restricting the decision, or *implications* resulting from different alternatives. Finally, the *rationale* behind the decision is expressed as *arguments* or *assessments*.

The Decision Documentation Model has two main advantages. First, it allows the incremental documentation of decision knowledge elements as they become available over time. Second, it supports the collaborative capturing of decision knowledge.

The Decision Documentation Model has been implemented by DecDoc [72], a tool for supporting the incremental and collaborative documentation of design decisions. DecDoc has been used in documenting decisions made to address security requirements [74], to derive code annotations to document implementation decisions [76], and in the management and visualization of decision knowledge in continuous software engineering [87], [93], [94].

QUARC (QUESTIONS, ACTIVITIES, RATIONALE, COMMUNICATION) a metamodel proposed by Nagel [135]. The QUARC metamodel consists of only one entity, the *quarc* which is defined as “a problem for which potential solutions must be found and decided on, and/or a work item to be completed”. Dependencies between quarks are represented by directed typed relations. Communication media, such as videos and drawings, can be linked directly to quarks. The quarks, relations between them, and linked communications are stored in quarc repositories. To facilitate the interaction with quarc repositories, Quarc Query Language (QQL) was developed to access, modify, and validate the contents of a quarc repository.

Table 2.1: Definitions of rationale elements used in this dissertation (adapted from Bruegge and Dutoit [16]).

Rationale element	Definition
Issue	A problem that needs discussion and negotiation to be solved. An issue typically can not be resolved algorithmically and does not have a single correct solution.
Alternative	A possible solution that could address the issue under consideration.
Pro-argument	A positive reason supporting an alternative.
Con-argument	A negative reason against an alternative.
Decision	An alternative selected to resolve an open issue.

In the QUARC approach, ideas and concepts can be captured in their original form, such as voice recordings and informal sketches, in a QUARC model. The QUARC model evolves over time into system model elements, and the original ideas and concepts represent the design rationale which can be attached as annotations to the system model elements.

The QUARC metamodel is an evolution of the IBIS issue concept. However, the author argues that capturing rationale using QUARC is much simpler than in IBIS, as the developers can simply create quarks that represent the main discussion topics and link communication segments during the meeting. The quarks and communication segments can be refined at later stages.

RATIONALE ELEMENTS IN THIS DISSERTATION Given the exploratory and empirical nature of the work presented in this dissertation, we focus our analysis on the basic elements of argumentation leading to decisions. Thus, we base our classification of rationale elements on IBIS for its conciseness and because it provides the basis for most of the subsequent issue models including DRL and QOC. In particular, we focus our analysis on five elements: issues, alternatives, pro-arguments, con-arguments, and decisions. Table 2.1 lists the rationale elements used in this dissertation and their definitions. The rationale elements definitions were adapted from Bruegge and Dutoit [16].

2.2.3 *Rationale Usage*

There are many potential uses of rationale at different phases of software development and it is nearly impossible to cover them all; thus, this section aims at giving an overview, rather than a comprehensive list of potential uses of rationale.

Lee [110] classifies the services that can be provided by design rationale into four main categories according to the user group who it benefits:

BETTER DESIGN SUPPORT The availability of design rationale can support designers as it yields to a better understanding of the design reasoning and the decision-making process. It supports the traceability of requirements to design decisions and vice versa. In addition, the captured rationale can be used to analyze the impact of design changes, i.e., to determine which decisions need to be revisited if the proposed change is made; as rationale captures the relationships between decisions. Rationale can also assist in the changes needed if the technology changes [24]. Capturing rationale also facilitates verification, validation, and reuse of design as it acts as an external design memory. Furthermore, design rationale supports collaborations and participatory design by providing a common vocabulary for discussions and negotiations among different stakeholders and by facilitating the early exposing of conflicts. For example, the SHARED-Design Recommendation and Intent Management System (SHARED-DRIMS) [144] uses design rationale to mitigate conflicts in collaborative environments.

BETTER MAINTENANCE SUPPORT Documented rationale increases the developers' understanding of the system; thus, making it easier to adapt and maintain [47]. Furthermore, documenting decisions made during maintenance allows the sharing and transferring of reusable maintenance knowledge. For example, SEURAT [21], [22], [23] uses rationale during maintenance by presenting rationale to the maintainers and inferencing over the rationale. The authors focus on three types of maintenance: adaptive, i.e., improving desirable qualities of the system without changing its functionality, corrective, i.e., correcting failures of the system, and enhancive maintenance, i.e., adding new functionality to the system.

Inferencing over the rationale is performed to check for completeness and consistency, to evaluate decision alternatives, and to perform impact assessment when requirements, development criteria, and assumptions change.

LEARNING SUPPORT Rationale facilitates the training of new members in a development team [62]. Moreover, systems used for managing rationale can learn from the captured rationale. For example, JANUS [56] uses a knowledge-based critics to monitor designers' actions. If a guideline is violated, it alerts the designer and provides recommendations that can be either accepted or discarded by the designer.

DOCUMENTATION SUPPORT Rationale can serve as documentation by capturing knowledge of the original designers [24]. Beside designers, rationale documentation can be valuable for other people involved in software development. For example, rationale documentation can be used by managers for design evaluation. Rationale can be also used by lawyers for checking intellectual property. Shipman and McCall [174] argued that the documentation perspective on design rationale has more widespread acceptance in practice, compared to the argumentation and communication perspectives.

Dutoit et al. [47] group the design rationale uses into four main categories: supporting collaboration, supporting reuse and change, improving quality, and supporting knowledge transfer. Burge et al. [25] list some of the potential uses of rationale at different phases of software development, i.e., supporting requirements engineering, design, implementation, and maintenance. It should be noted that these categorization are only different ways for grouping the potential uses of rationale and they share similarities and dependencies among each other.

2.3 TEXT MINING FUNDAMENTALS

Text mining, also known as knowledge Discovery in Text (KDT), is the process of mining useful information from text documents [190]. Similar to data mining [69], text mining seeks to discover interesting patterns in data. However, instead of formalized database records, text mining is interested in identifying

patterns in unstructured natural language documents. Text mining uses techniques adapted from data mining, machine learning, information retrieval, and computational linguistics [54], [190].

One application of text mining is *text classification*, also known as *text categorization*, where a collection of text documents are classified into a predefined set of categories [54]. There are two types of text classification, *single-label* and *multi-label* classifications. In single-label classification, exactly one class is assigned to the document. A special case of the single-label classification is the *binary* classification, in which each document is assigned to one of two categories. In multi-label classification, a document can be assigned to more than one category.

Text classification is often used in *supervised machine learning*, where a classifier is built by learning from a set of pre-classified documents [170]. The resulting classifier is a prediction model that predicts the classes of unseen text documents. Building a text classifier using supervised machine learning algorithms consists of three main steps. First, transforming the text documents into an intermediate representation that is suitable for the learning algorithm. Second, providing a pre-labeled dataset of documents that is split into *training set* and *test set*. Finally, feeding the training set to the learning algorithm and evaluating the performance of the resulting classifier using the test set.

Transforming the data into a representation that is suitable for the machine learning algorithm is known as *text preprocessing*, and it plays an important role in text mining [190]. In the following, we describe four preprocessing steps that are also applied in this dissertation:

LOWERCASE CONVERSION The process of converting the characters of words into lowercase letters.

STEMMING The process of reducing inflected words into their root form, so-called word stems [190]. For example, the words: “developers”, “developing”, and “development” are all reduced to “develop”.

TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY (TF-IDF) Term Frequency (TF) is the number of times a word occurs in a document; while Inverse Document Frequency (IDF) measures how important the word is to the document [131]. TF-IDF is the product of the two ($TF * IDF$) which

is proportional to the discrimination power of a word, i.e., rare words have higher TF-IDF than words commonly used in other documents.

N-GRAM TOKENIZATION The process of converting a text document into a sequence of n words [190]. For example, applying 3-gram tokenization on the sentence “This is an example of tokenization” results in the following tokens: “This is an”, “is an example”, “an example of”, and “example of tokenization”.

In this dissertation, we compare the performance of the following different machine learning algorithms that have been proven to perform good text classification [15], [58], [85], [170]:

MULTINOMIAL NAIVE BAYES (MNB) is a variation of Naive Bayes [114], [128] that overcomes the weakness of Naive Bayes of only modeling the absence or the presence of words in a text document. MNB captures the frequencies of words in a document, making it more suitable for text classification.

SUPPORT VECTOR MACHINE (SVM) creates a representation of text documents as points in n -dimensional space (where n is the number of classification features) and looks to separate the classes by a hyperplane that is as wide as possible. Support vectors are the coordinates of the individual documents [15], [37].

DECISION TREE (DT) uses the manually labeled documents to construct well-defined true/false queries using a tree structure. The tree nodes represent questions, tree leaves represent document categories, and the branches represent conjunctions of features leading to these categories [15], [131].

RANDOM FOREST (RF) is an ensemble learning algorithm that constructs multiple decision trees in randomly selected spaces of the features space. The prediction of the individual decision trees are combined by applying bagging or bootstrap aggregating to generate the final classification [44], [81].

LOGISTIC REGRESSION (LR) is a modified regression technique that predicts the document class by calculating the probability for each class and then choosing the class with the maximum probability [89], [145].

To train and evaluate machine learning classifiers, an important statistical technique known as *k-fold cross validation* is applied [194]. In *k-fold cross validation*, the data is split into *k* partitions of equal sizes, called *folds*, *k-1* folds are used for training the classifier and the remaining fold is used for testing it. The procedure is repeated *k* times, rotating the training and testing folds, so that at the end every fold has been used exactly once as a testing set. The final evaluation is calculated by averaging the results of the *k* iterations.

One approach to perform multi-label classification is by applying problem transformation methods [147] which involves the transformation of the input instances into representations suitable for single-label classification. In this dissertation, we apply two of the most popular problem transformation methods [147], [187], [197]:

BINARY RELEVANCE (BR) is the base line approach for problem transformation, in which a binary classifier is independently trained for each label and the prediction of unseen document is the aggregated predictions of all independent classifiers. The main drawback of this approach is the assumption that the classes are independent.

LABEL POWERSET (LP) takes into account class correlations by generating a class for every combination of labels. The main drawback of this approach is that the number of generated classes can grow exponentially.

A common challenge in supervised machine learning is *imbalanced datasets* where the classification categories are not equally represented [32]. Imbalanced datasets might cause the classification algorithm to skew towards the majority class and ignore the minority class, which is usually more important to be correctly classified. Two popular techniques for handling this problem are under-sampling and SMOTE (Synthetic Minority Over-Sampling Technique). Under-sampling [46] uses a subset of the majority class for training the classifier; while SMOTE [31] applies oversampling on the minority class by generating synthetic examples. In this dissertation, we compare between the application of under-sampling and a combination of SMOTE and under-sampling as previous research has proved that classifiers achieve better performance when combining both sampling techniques [31].

To evaluate the classification performance of the resulting classification models, we used the following standard metrics in machine learning:

PRECISION measures the exactness of the resulting predictions, and it is defined for a category as the percentage of correctly classified documents among all documents that were classified as belonging to this category [54]. It is calculated as follows:

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i} \quad (2.1)$$

RECALL measures the completeness of the resulting predictions, and it is defined for a category as the percentage of correctly classified documents among all documents belonging to that category [54]. It is calculated as follows:

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i} \quad (2.2)$$

Where TP_i is the number of documents that are correctly classified as being of type i , FP_i is the number of documents incorrectly classified as being of type i , and FN_i is the number of documents incorrectly classified as not being of type i .

THE F1-MEASURE is the harmonic mean of precision and recall, and it is calculated as follows:

$$F1 = 2 * \frac{\text{Precision}_i * \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \quad (2.3)$$

Part II

Analyzing Rationale in Text-based Developers' Communications

Rationale in Chat Messages of Co-located Teams

“Media—i.e., development tools and communication channels—play a critical role in how externalized and tacit knowledge is formed, shared, manipulated, and captured.”

—Storey et al. [181]

In the previous chapters, we discussed some approaches of rationale management in software engineering and the developers reluctance to capture rationale during software development. Considering that the goal of this dissertation is to support developers in capturing rationale, we hypothesize that part of the rationale can be captured by looking at text-based communication channels.

In this chapter, we report on an empirical study to explore how developers discuss rationale in chat messages. In particular, we analyzed chat messages of three co-located development teams for a duration of three months to better understand the rationale hidden in chat messages with regards to its frequency and completeness.

The chapter is structured as follows: Section 3.1 explains the study design in terms of research questions, research data, and research method. Section 3.2 presents the study results. We discuss our findings in Section 3.3 and the limitations of our study in Section 3.4.

3.1 STUDY DESIGN

In this section, we introduce our research questions, the data we used to perform our analysis, and describe the followed research method.

3.1.1 *Research Questions*

The aim of this study was to evaluate chat messages of development teams as a potential source for rationale. For this purpose, we explored the rationale frequency and rationale completeness in chat messages.

RQ1. Rationale frequency: *What is the frequency of messages containing rationale in developers' chat messages?*

This question describes how often rationale appears in chat messages. This information gives a first insight whether it is worth to consider chat messages as a source of rationale.

RQ2. Rationale completeness: *How complete are the rationale elements extracted from developers' chat messages?*

This question describes the syntactical check of the recorded rationale in chat messages. Rationale is distributed across different development artifacts, e.g., bug reports [162] and design session transcripts [79], and all of these sources could be used together to capture a more complete rationale of the software system. We use the completeness check defined by Burge et al. [25], in which *rationale completeness* occurs when for each documented decision, all the rationale elements justifying the decision are documented. Answering this question could help practitioners and researchers by providing insights on how to integrate the rationale extracted from chat messages with the rationale extracted from other sources.

3.1.2 *Research Data*

We analyzed the chat messages of three development teams that were part of a multi-project capstone course at the Technical University of Munich in 2015 and 2016 [18], [100]. During the course, teams developed mobile applications for industrial partners and dealt with incomplete and evolving requirements following an agile software methodology [101]. The participants used Atlassian HipChat for instant messaging¹. HipChat supports diverse integrations to exter-

¹ <https://www.hipchat.com>

Table 3.1: Overview of analyzed chat messages.

Team	Chat messages before filtering	Chat messages after filtering	Analyzed period
Team A	4,106	3,974	Apr. 15 – Aug. 13, 2016
Team B	2,214	2,164	Apr. 15 – Aug. 11, 2016
Team C	3,026	2,564	Oct. 16, 2015 – Feb. 23, 2016
Total	9,346	8,702	

nal services and bots, e.g., with Bamboo², a continuous integration and continuous deployment server, to send notifications about build results and Standup Bot³ to report and retrieve statuses for stand-up. Each team consisted of 8 to 9 developers and a project leader. When selecting the teams for the study, we considered only teams who communicated in English and wrote more than 2,000 messages. To focus our analysis on the messages written by members of development teams, messages that were automatically generated by one of the services or bots were filtered out. Table 3.1 shows the number of chat messages before and after filtering automatically generated messages, and the analyzed period for each of the three teams. In total, we analyzed 8,702 chat messages.

3.1.3 Research Method

To explore the *frequency* of rationale in chat messages, we analyzed how often rationale appears in chat messages and the frequency of different rationale elements. We manually analyzed the chat messages in the dataset by applying content analysis techniques as described by Neuendorf [138]. Two researchers, one of them is the author of this dissertation, independently inspected the chat messages of the three teams in the dataset and identified the contained rationale. This process consisted of three steps:

- A. **Developing a coding guide:** The aim of this step was to systematize and minimize disagreements between the two coders. Since many representations have been proposed in literature to model rationale, it is important

² <https://www.atlassian.com/software/bamboo>

³ <http://botlab.hipch.at>

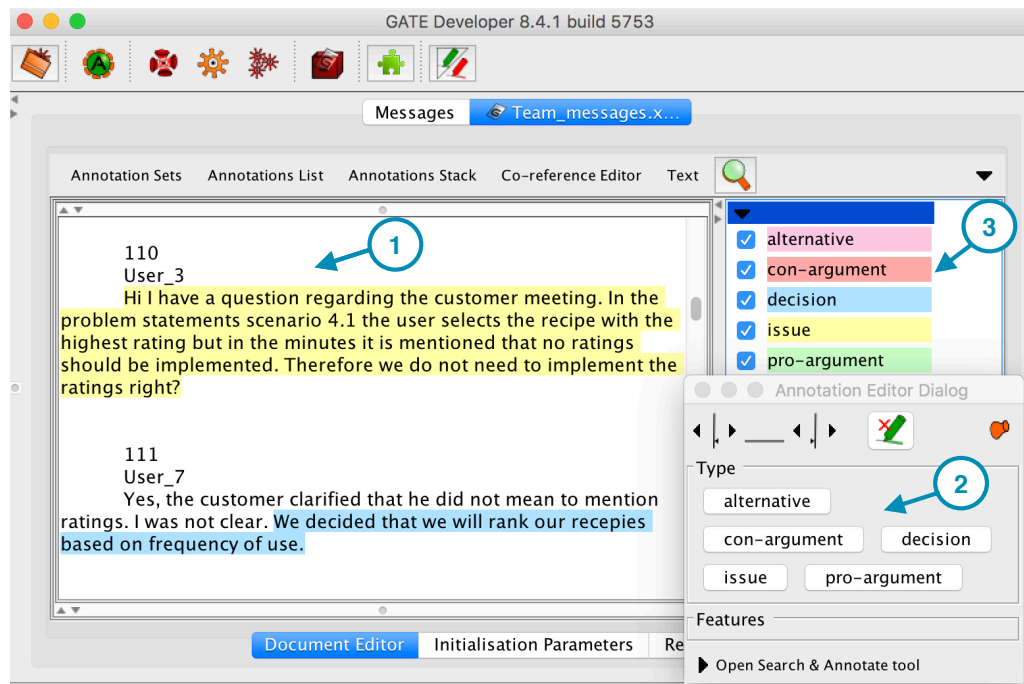


Figure 3.1: A screenshot of using GATE for the manual coding of chat messages. (1) The main window displays the list of (anonymized) chat messages to be annotated, (2) when a coder highlights a part of a message that contains rationale, a pop-up window appears (the Annotation Editor Dialog) where the coder can specify the rationale element(s) present in the message, and (3) the color codes for different rationale elements.

that the two coders share a unified understanding of the elements that constitute rationale. To this end, we developed a coding guide that includes clear definitions of the rationale elements and examples for each element (see Appendix A).

The coding guide was developed in an iterative process consisting of two trial iterations. In each iteration, the two coders independently identified rationale elements in 1,000 chat messages. The coders' disagreements were analyzed and the coding guide was modified accordingly to minimize disagreements in the next iteration.

- B. **Manual coding of chat messages:** For the manual coding task, we used GATE (General Architecture for Text Engineering) [38], a Java-based framework for a diverse set of natural language processing applications.

Figure 3.1 shows a screenshot of GATE as used by coders. The main window displays the list of chat messages to be coded. If a message contains rationale, the coder highlights the message part containing the rationale and specifies its type.

The coding unit, i.e., the highlighted part, can be one sentence, multiple sentences of a message, or the complete message. We refer to the coded units as *text snippets*. For each snippet, coders specified whether it contains rationale and what type of rationale elements are present. A text snippet might contain multiple rationale elements. Coding text snippets allows for capturing the text containing rationale in the finest-grained manner since a message might contain additional irrelevant information.

The two coders independently coded each message in the dataset. The average time to code 8,702 messages was 13 hours per coder, highlighting the large efforts required to manually extract rationale elements.

- c. **Disagreement reconciliation:** Disagreements included situations when the two coders identified different rationale elements in the same message or when only one coder coded a message as containing rationale. The average inter-rater agreement between the two coders was 95% for identifying messages containing rationale, and 94% for identifying different rationale elements. The disagreements were resolved through discussions between the two coders.

To explore the *completeness* of rationale in chat messages, it is important to identify the semantic relationships among rationale elements identified in different chat messages as the rationale discussion could span multiple messages. For example, a developer might discuss a specific issue in a single message and other developers propose different alternatives to resolve the issue and argue for and against these alternatives in other messages.

To achieve this, the same two coders who conducted the manual content analysis of the chat messages manually inspected the messages containing rationale and identified the semantically related rationale elements contained in these messages. The semantically related rationale elements were assigned the same code, i.e., a number. Disagreements were resolved through discussions between

the two coders. After identifying the semantically related rationale elements, we asked the following questions:

1. For each discussed issue, were alternative solutions proposed and was a decision made?
2. For each selected alternative (i.e., decision), were pro-arguments supporting its selection presented?

3.2 RESULTS

This section presents the analysis results of the rationale frequency in developers' chat messages and the completeness of the existing rationale.

3.2.1 *Rationale Frequency*

Although the coding unit during the manual coding was text snippet to allow for a more fine-grained annotation of rationale elements, we noticed that messages are of short length and when containing rationale most likely contain only one text snippet with rationale⁴. Thus, we reported the results on the message level. A message was considered to contain a particular rationale element, if it contains at least one text snippet that was annotated as containing that element.

Figure 3.2 shows the percentage of chat messages identified as containing rationale per team. On average, 9% of the team chat messages contain rationale (8% in Team A, 11% in Team B, and 7% in Team C). In total, 752 of the 8,702 analyzed chat messages contain rationale. Although the number of chat messages containing rationale may not be considered high, the manual analysis revealed that developers discuss various elements of rationale in these messages that comprises valuable knowledge about the software system. Table 3.2 shows an overview of the frequency distribution of different rationale elements across messages and examples of coded rationale elements.

Overall, we found that proposing alternatives to different issues is predominant in almost 51% of the messages containing rationale. The second most fre-

⁴ 87% of the messages with rationale contain only one text snippet with rationale.

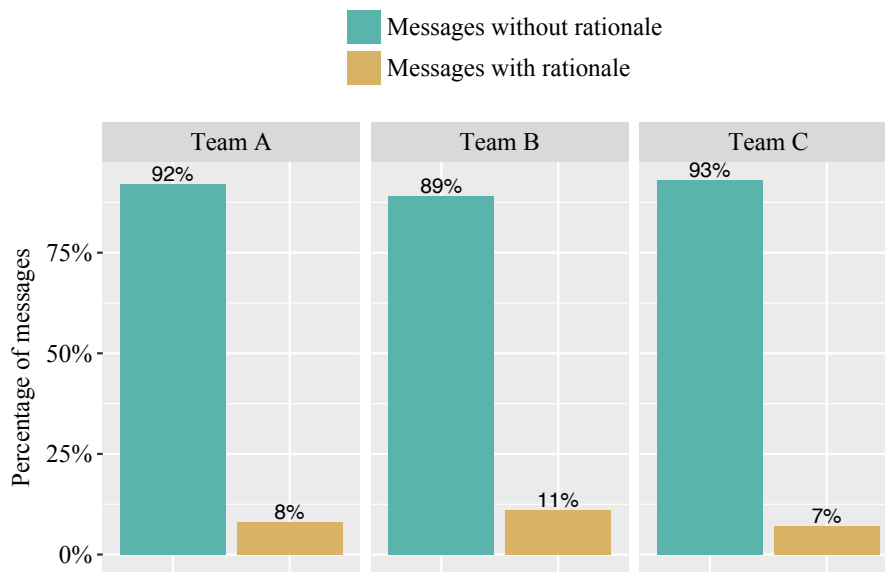


Figure 3.2: Chat messages containing rationale per team.

quent rationale element is issue, present in 24% of the messages containing rationale. Pro-arguments were mentioned in 23% and con-argument in 18% of the messages containing rationale. These numbers confirm our observation, during the manual coding, that team members tend to argue for the alternatives they proposed and their reasons to select these alternatives (i.e., pro-arguments) more frequently than arguing against other alternatives (i.e., con-arguments). Lastly, decisions were identified in 10% of the messages containing rationale.

The message length in the dataset ranges from 1 to 2,026 characters with a median of 50 characters across the three teams ($Mean=81.84$, $SD=117.36$). However, messages containing rationale have a higher median length of 100 characters ($Mean=146.68$, $SD=150.02$). Although the manual coding was performed on the text snippet level rather than on the message level (see Section 3.1.3), the majority of messages containing rationale (87%) consists of only one text snippet of rationale. In addition, we found that 80% of the messages containing rationale discuss one rationale element, 16% discuss two rationale elements, and only 4% discuss more than two elements. From this result, it can be seen that developers tend to discuss rationale elements in a sequence of short chat messages rather than long messages with intertwined rationale elements.

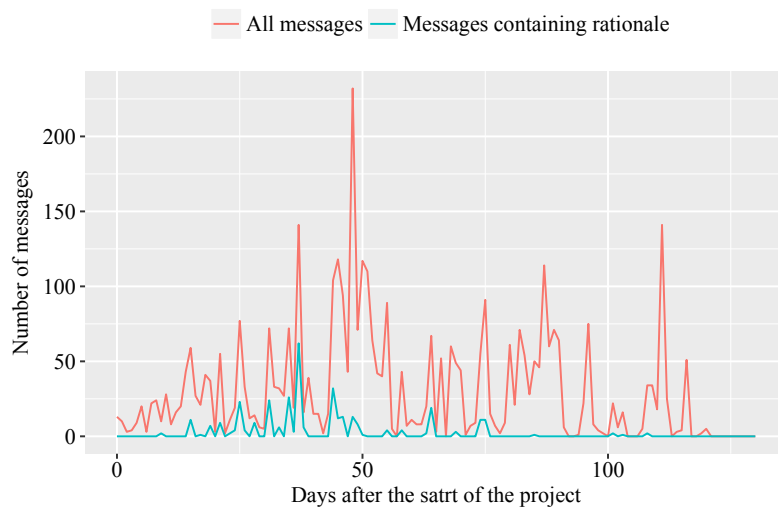
Table 3.2: Frequency distribution of rationale elements across messages containing rationale per team.

Frequency distribution					
Rationale element	Team A	Team B	Team C	Total	Example
Issue	25%	28%	17%	24%	"Plus if this is implemented using segueways, what screen do you go back to when you click 'back'?"
Alternative	45%	54%	57%	51%	"What do u think of having a "start cooking" button? Clicking on the recipe name might not be the most intuitive? Thoughts?"
Pro-argument	17%	26%	30%	23%	"It's better UX :) definitely"
Con-argument	18%	17%	19%	18%	"But still, I think it is too complicated for now to build it with tabs."
Decision	13%	7%	9%	10%	"We decided that we will rank our recipes based on frequency of use."

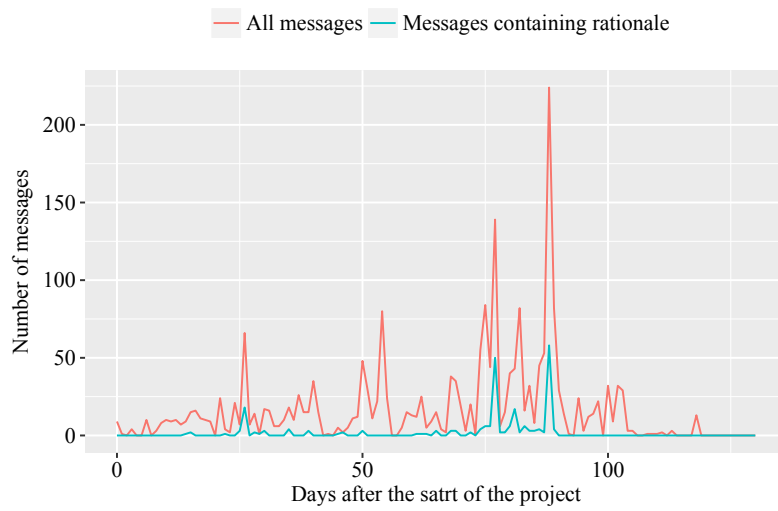
To explore how the passage of time and the number of messages influence the amount of rationale found in chat messages, we investigated the distribution of all messages as well as messages containing rationale over the duration of the project. Figure 3.3 visualizes the distribution of all messages and messages containing rationale over the project duration per team.

We found that different development teams had different rationale distribution in their chat messages. Teams A and B had a significant increase in the number of messages containing rationale at certain time points, shown as sharp rises in their line slopes. One possible interpretation of this result is that discussions between developers might be denser around particular milestones during the project, e.g., at the beginning of development sprints or before releasing to the customer. Team C had a steady distribution of chat messages containing rationale that spans the entire project duration.

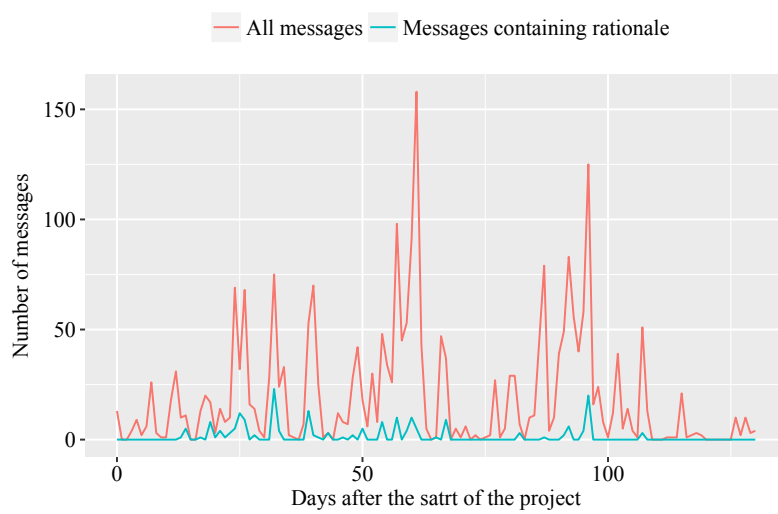
The Spearman's correlation coefficients between the number of days passed since the start of the project and the number of messages containing rationale were -0.2 (p -value=0.03), 0.11 (p -value=0.28), and -0.26 (p -value=0.01) for teams A, B, and C, respectively. This indicates a weak negative correlation in teams A



(a) Team A



(b) Team B



(c) Team C

Figure 3.3: Distribution of all messages as well as messages containing rationale over the duration of the project.

and C, which means that the more days passed, the less rationale was discussed in the chat messages. A possible explanation is that the main functional requirements and important decisions about the system design were discussed at the early phases of the system development. However, the value of the correlation coefficient in Team B indicates a very weak correlation.

As shown in Figure 3.3, the increase in the number of chat messages is not always an indicator of an increase in the number of messages containing rationale. However, the Spearman's correlation coefficient between the total number of chat messages and the number of messages containing rationale was 0.5 in all three teams indicating a moderate positive correlation.

3.2.2 *Rationale Completeness*

Our results show that for 79% of the issues identified in the chat messages, alternatives were proposed to resolve the issues. In 48% of these issues, we were able to identify the decisions made, i.e., the selected alternatives, in the chat messages. However, in only 48% of the cases in which a decision was made, the pro-arguments in support of this decision were present in the chat messages. Finally, we found that in 21% of the issues found in the chat messages, neither alternatives were suggested nor a decision has been made. One possible interpretation is that the team members discussed and resolved the open issues through face-to-face communication. For example, while discussing an issue one developer wrote, "*Probably best if we discuss this in the meeting*". These results indicate that chat messages are not to be used as the only source for rationale but rather as one of many potential sources. The rationale extracted from chat messages should be integrated with rationale extracted from other sources for a more complete rationale, e.g., the decisions made to resolve some of the open issues extracted from chat messages might be available in the meeting minutes of the development team.

3.3 DISCUSSION

The results of our study show that (1) chat messages are a valuable source for rationale during software development, (2) chat messages should be used in combination with other development artifacts for capturing a complete rationale. In the following we revisit our research questions.

RATIONALE FREQUENCY: Although a small percentage of chat messages contain rationale (9%), the manual content analysis results show that these messages contain valuable knowledge about the software system. In chat messages, team members actively engage in discussing issues, proposing alternatives, arguing for and against these alternatives and collaboratively making decisions. However, the informality and unstructured nature of chat messages poses a number of challenges for extracting rationale from chat messages. First, rationale could span multiple messages complicating its identification. Second, multiple elements of rationale could be discussed in a single message, and distinguishing between the different elements is a nontrivial task even for a human.

RATIONALE COMPLETENESS: In almost half of the identified issues (48%), chat messages contained a complete rationale with respect to the alternatives considered, the selected alternative, i.e., decision, and the arguments supporting the decision. However, for the remaining issues, the identified rationale was incomplete. A possible explanation is that developers use different communication channels in addition to chat messages. For example, developers might continue some of the chat messages discussions in face-to-face meetings and decisions made to resolve the issues identified in chat messages might be documented in other development artifacts such as meeting minutes. This finding emphasizes the importance of linking related rationale elements extracted from different development artifacts for a more complete capturing of rationale.

3.4 THREATS TO VALIDITY

In this section, we discuss the threats to the validity of our results according to the four validity aspects as defined by Runeson et al. [164].

CONSTRUCT VALIDITY is concerned with how accurately the study observations interpret and measure the theoretical constructs [164]. The messages coders have not been involved in the development of the analyzed projects. Thus, the consideration of whether a chat message contains rationale depends on the coders' judgment, which can be different from what the actual developers consider as rationale. To mitigate this threat, coders were asked to read the description of the mobile applications, from which the messages are analyzed, to make sure that they understand the analyzed applications' main functionalities. In addition, both coders have software engineering background.

Although the list of rationale elements used in our analysis are based on the well-known IBIS model [102], the list of elements could be incomplete and its descriptions simplified. This threat could lead to the capture of incomplete rationale. However, the analyzed rationale elements are shared among most rationale representation models.

INTERNAL VALIDITY is concerned with the confounding factors that may influence the study results [164]. During the manual coding, the determination if a message contains rationale and which rationale elements are present is a subjective decision. To mitigate this risk, we created a coding guide with precise definitions and examples for different rationale elements. The guide was used by the coders during the coding task. Furthermore, each message in the dataset was coded by two people and the disagreements were discussed and resolved by the two coders.

EXTERNAL VALIDITY is concerned with the generalizability of our results [164]. We analyzed the chat messages exchanged during a university course which might affect the generalizability of our results. However, the students in the three development teams worked closely with industrial customers and dealt with incomplete and evolving requirements on innovative projects. Previous research found that subject's experience level might have more effect on the results than the experiment setting (academic or industry) [167]. In our study, the majority of the student participants described themselves as semi-professional developers and they reported having part-time jobs in the industry.

RELIABILITY is concerned with to what extent the study results are dependent on a specific researcher, i.e., whether the study yields the same results if replicated by other researchers [164]. Although, the analyzed messages could not be made available due to privacy issues, the coding guide with clear definitions and examples of the rationale elements is made available to other researchers (see Appendix A). Furthermore, peer-coding was performed on all the analyzed messages to minimize the bias that could result from individual coding.

Rationale in Text-based Developers' Communications of Distributed Teams

“Social tools leave a digital audit trail, documenting our learning journey—often an unfolding story—and leaving a path for others to follow.”

—Marcia Conner [12]

In the previous chapter, we presented an empirical study of rationale in the chat messages of three co-located teams. We found that on average 9% of their chat messages contain rationale, despite the dominance of face-to-face communications and the absence of geographical and time zone differences. This finding raises intriguing question regarding the presence of rationale in written communication channels of distributed teams, where developers are geographically distributed and rely heavily on written communications.

In this chapter, we present two empirical studies to understand how developers discuss rationale in the written communication channels of three Open Source Software (OSS) projects, which are often developed and maintained by globally distributed teams [65]. In particular, we analyze rationale in the following text-based communication channels:

DEVELOPERS' CHAT MESSAGES: We examine the frequency of messages containing rationale and contributors of rationale in the Internet Relay Chat (IRC) channels of three OSS projects.

DEVELOPERS' COMMENTS IN ISSUE TRACKING SYSTEMS: We examine the frequency of comments containing rationale and contributors of rationale in the comments of Issue Tracking Systems (ITS) of three OSS projects.

The chapter is structured as follows: Section 4.1 introduces our research questions. Section 4.2 presents the study of rationale in IRC messages, including study design, results, discussion of the results, and threats to validity. Section 4.3 presents the study of rationale in the comments of issue tracking systems, including study design, results, discussion of the results, and threats to validity.

4.1 RESEARCH QUESTIONS

In our analysis, we aim to understand how developers discuss rationale in IRC messages and issue tracking systems of distributed teams. To this end, we investigate the frequency and contributors of rationale.

RQ1. Rationale frequency: *What is the frequency of messages, or comments, containing rationale in chat messages and issue tracking systems of distributed teams?*

While it is widely accepted that IRC messages and issue tracking systems in OSS projects contain valuable information about the software system and its history, there is still a lack of empirical evidence about the presence and volume of rationale in these channels. Answering this question provides insights about the nature of existing rationale and provides the basis for training and evaluating automated classification techniques presented later in this dissertation.

RQ2. Rationale contributors: *Which developers contribute rationale in chat messages and issue tracking systems of distributed teams?*

This question is inspired by the work of Brunet et al. [20] that found a strong correlation between development activities (committing into code repository) and contributing to design discussions in pull requests, commits, and issues. By asking this question, we aim to find if such correlation holds between development activities and rationale contribution in IRC discussions and issue tracking systems; Do developers who commit more often contribute more rationale? This could provide first insights on linking the rationale found in IRC messages and issue tracking systems to different parts of the source code.

4.2 RATIONALE IN DEVELOPERS' CHAT MESSAGES

Internet Relay Chat (IRC) channels are increasing in popularity for synchronous communications in OSS projects [30], [45], [88]. Developers use IRC channels for discussing development and implementation details and exchanging knowledge and ideas with other developers [70], [181]. Mozilla Foundation describes IRC as “the primary form of communication for members of the Mozilla community”¹.

While several prior studies have examined the general role of IRC channels in OSS development [30], [70], [172], there is still no empirical evidence of how OSS developers discuss rationale in IRC messages. The following excerpt from the Apache Lucene website reinforces our motivation for studying rationale in developers' IRC messages.

“The IRC channel can be used for online discussion about Lucene related stuff, but developers should be careful to transfer all the official decisions or useful discussions to the issue tracking system.”²

This excerpt sheds light on two important aspects. First, developers' discussions over IRC channels may contain valuable rationale about development decisions. Second, there is no systematic methodology for transferring such knowledge to official documentation artifacts (e.g., issue trackers for most OSS projects) other than relying on the developers to transfer it manually. As a consequence, potential rationale lying hidden in IRC messages is rarely made explicit or taken advantage of.

4.2.1 *Study Design*

This section introduces the design of our empirical study. We describe the different phases of the applied research method, including data collection, manual coding process, and multiple alias resolution.

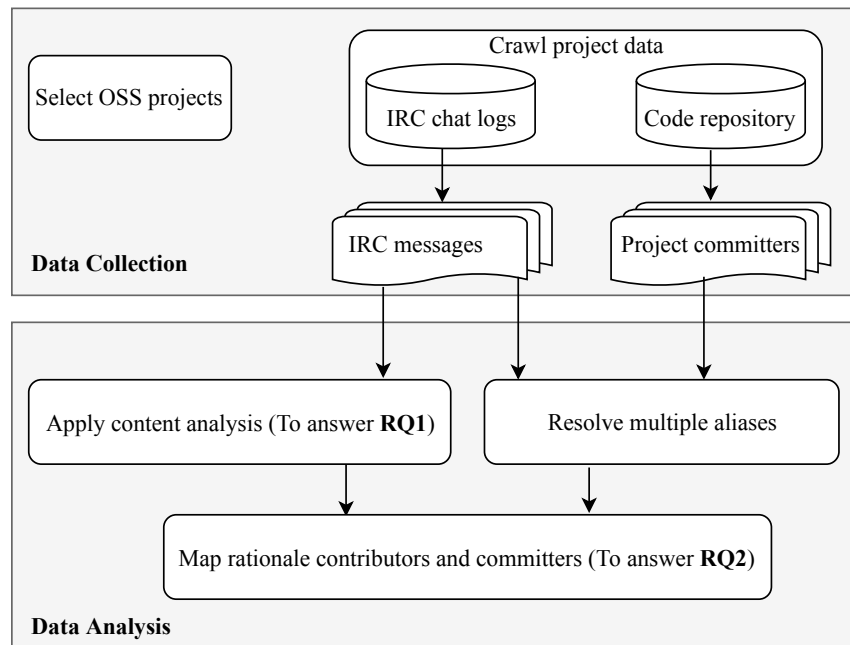


Figure 4.1: Applied research method for studying rationale in developers' chat messages of distributed teams.

4.2.1.1 Research Method

Our research method consisted of two phases: data collection and data analysis, as depicted in Figure 4.1. In the data collection phase, we first selected the OSS projects for our study. We selected three OSS projects: Apache Lucene, Mozilla Thunderbird, and Ubuntu. In accordance with selection criteria applied by similar studies [140], [142], [172], we chose the three OSS projects for the following reasons. First, to mitigate threats to external validity, we selected projects from diverse domains. Second, the archived IRC logs and source code repositories of the selected OSS projects are publicly available. Third, the three projects are popular and mature OSS projects with a large community of active developers and users. After selecting the projects, we crawled the IRC chat logs and source code repositories of the three projects to extract the IRC messages and project committers. We detail on each step in Section 4.2.1.2.

¹ https://developer.mozilla.org/docs/Mozilla/QA/Getting_Started_with_IRC

² <https://lucene.apache.org/core/discussion.html>

Table 4.1: Overview of IRC messages.

Project	IRC messages				Coding sample
	Archived years	Channel	#Messages (before filtering)	#Messages (after filtering)	
Apache Lucene	Apr. 2010 – May 2017	# <i>lucene-dev</i>	273,123	71,897	2,500
Mozilla Thunderbird	May 2012 – May 2017	# <i>maildev</i>	299,771	291,416	2,500
Ubuntu	Oct. 2004 – May 2017	# <i>ubuntu-devel</i>	2,897,987	2,438,812	2,500
Total	—	—	3,470,881	2,802,125	7,500

To answer **RQ1**, we first manually analyzed a sample of 7,500 IRC messages by applying content analysis techniques as described by Neuendorf [138]. The message coding process is explained in Section 4.2.1.3.

To answer **RQ2**, we mapped IRC authors who contributed rationale with the project committers. This process consisted of two steps. First, we applied an alias resolution approach on IRC and committers identifiers separately, as developers can use multiple identifiers within a single channel. Second, we associated the IRC identifiers with the identifiers used in the commit history whenever viable. Both steps are elaborated further in Section 4.2.1.4.

4.2.1.2 Research Data

In this section, we describe the process of collecting and extracting our research data from three OSS projects: Apache Lucene³, Mozilla Thunderbird⁴, and Ubuntu⁵. For each project, we crawled the IRC logs and commit history, parsed the data to extract the required fields, and stored them into a MySQL database for further analysis. For each IRC message, we extracted: message author, message, and message date. For each commit, we stored: committer, commit message, and commit date. An overview of the collected IRC messages and commit messages are shown in Table 4.1 and Table 4.2, respectively. In total, we

³ <https://lucene.apache.org>

⁴ <https://www.thunderbird.net>

⁵ <https://www.ubuntu.com>

Table 4.2: Overview of commit messages.

Project	Commit history	
	Years	#Commits
Apache Lucene	2001-2017	27,787
Mozilla Thunderbird	2007-2017	20,569
Ubuntu	1999-2017	349,813
Total	—	398,169

collected 3,470,881 IRC messages and 398,169 commits from the three projects. We detail on the collection process for each project in the following.

APACHE LUCENE is a Java-based full-text search engine library. In recent years, it has become one of the most popular free information retrieval libraries [129]. We obtained the complete archive of the development IRC channel *#lucene-dev* logged by Colabti⁶. Next, we filtered out automatically generated messages, for example, messages generated when users join or leave the channel (“*** *hoss* joined”). This resulted in 71,897 messages written by 266 authors over the last 8 years. From Lucene’s code repository, we collected 27,787 commits done by 152 committers over the last 16 years. We also obtained the official list of committers from Lucene’s website.

MOZILLA THUNDERBIRD is a cross platform email client with an estimation of 25 million active users⁷. We crawled 299,771 IRC messages from the development channel *#maildev* logs. Afterwards, we filtered out messages posted by Firebot (a general-purpose Mozilla chatbot) which resulted in 291,416 messages written by 1,180 authors over the last 6 years. From Thunderbird’s code repository, we collected 20,569 commits performed by 895 committers over the last 11 years. Additionally, we imported the official list of core developers provided by Thunderbird.

⁶ http://colabti.org/irclogger/irclogger_logs/lucene-dev

⁷ <http://blog.mozilla.org/thunderbird/>

UBUNTU is a Debian-based Linux operating system. Ubuntu is one of the most popular Linux distributions and has over 40 million desktop users and more than 500 active members from 100 countries. The development IRC channel *#ubuntu-devel* is “home to many Ubuntu developers for real-time communication”⁸. We fetched the complete channel archive⁹ and filtered out automatically generated messages such as “=== bob2 [rob@bob2.user] has joined #ubuntu-devel”. This resulted in 2,438,812 messages written by 13,645 authors over the last 14 years. Likewise, we extracted 349,813 commits done by 6,724 committers over the last 18 years. Finally, we queried the official list of core developers provided by Ubuntu.

4.2.1.3 Coding of IRC Messages

To analyze the frequency of the rationale elements in IRC messages, two researchers, including the author of this dissertation, applied manual content analysis [138] on a sample of IRC messages from the three studied projects. The manual coding process consisted of the following steps:

- A. **Developing a coding guide:** To systematize the coding process and assure a common understanding, we designed a coding guide (see Appendix B). The coding guide provides instructions about the coding task and definitions and examples of the different rationale elements (listed in Table 2.1). It was developed in two iterations. In each iteration, the two coders used the guide to annotate a random sample of 300 messages. The disagreements were analyzed and the guide was refined accordingly.
- B. **Sampling of IRC messages:** IRC messages are short in length with a median of 42 characters (Mean=54.31, SD=48.89), written in informal language and context-dependent. Analyzing messages in isolation of the context in which they were exchanged might lead to imprecise results. To keep the conversation context, we created our sample by randomly selecting complete chat days instead of single messages. For each project, we randomly selected chat days so that the total adds up to 2,500 messages. Overall, our sample consisted of 35 chat days from Apache Lucene, 10

⁸ <https://wiki.ubuntu.com/UbuntuDevelopment>

⁹ <https://irclogs.ubuntu.com>

days from Mozilla Thunderbird, and 8 days from Ubuntu. This results in a sample of 7,500 messages from the three OSS projects.

- c. **Manual coding of IRC messages:** To avoid bias during the coding, each message was coded by the two coders independently. For each message, the coders indicated if the message contains rationale and specified the type of rationale element(s) included in the message. A message can be annotated with more than one rationale element. For example, “*Maybe for later version support, we should do it like in StandardTokenizer*” is annotated with alternative and pro-argument. We used GATE [38] for the manual coding of the messages (a screen shot of using GATE for the manual coding is shown in Figure 3.1 in Chapter 3). During the coding task, the complete chat days were displayed for the coders. This allowed the coders to obtain the conversation context while coding single messages. We refer to the coding guide for the detailed annotation instructions (see Appendix B). Coders reported an average of 20 hours to complete the coding task.
- d. **Disagreements reconciliation:** All messages were coded twice. We consider that disagreements occur when only one of the coders annotated a message as containing rationale or when the two coders annotated a message with different rationale elements. The average inter-rater agreement was 83% for identifying messages containing rationale and 78% for identifying different rationale elements. Coders discussed and resolved their disagreements.

4.2.1.4 *Alias Resolution*

The multiple alias problem occurs when multiple nicknames (aliases) are assigned to the same person [173]. It is commonly faced in studies examining OSS repositories [13], [157], [172]. Resolving this problem is an essential step to prepare our data for further analysis. The cause of this problem in our study is twofold. First, we use multiple data sources in our study, namely, IRC channels and commit history, and developers might use different identifiers on these sources. In IRC channels, participants assign themselves nicknames when joining the channel. A nickname is a self-chosen name [14], which can be an abbreviation of the real name (e.g., *markmiller* for *Mark Miller*), or a pseudonym (e.g.,

luceneuser). While in source code repositories, developers use names, nicknames, emails, or a combination of them. Second, developers might use multiple aliases within a single source; mostly very similar ones. For example, *sagarwal*, *sshagarwal* and *sshagarwaltb* are used by the same person to write to IRC. As an Apache Lucene developer described the problem, “*I wish we didn't have these pseudo-names here; I don't know who's who half the time.*” Similarly, a developer can commit code to a repository with different identifiers. For example: *Michael McCandless* <*mikemccand@apache.org*>, *Michael McCandless* <*mail@mikemccandless.com*>, *Mike McCandless* <*mikemccand@apache.org*>, and *mikemccand* <*mike@elastic.co*> are used by the same person to commit code to the repository.

To resolve aliasing in our collected data, we applied an approach similar to the ones by Bird et al. [13] and Panichella et al. [142]. We started with the extracted IRC messages' authors and committers from the three projects (see Section 4.2.1.2). For each project, we performed the following steps automatically¹⁰:

1. **Extract email login names:** We removed emails' domains (anything after “@”). For example, *sarowe@gmail.com* and *sarowe@apache.org* are converted to *sarowe*.
2. **Normalization:** We converted identifiers to lowercase, removed punctuation (e.g., “_”), numbers, and eliminated extra whitespace. For example, *JoeS*, *JoeS1* and *JoeS11* are all mapped to *joes*. Additionally, we removed the term “-guest” that was commonly attached to Ubuntu IRC identifiers (e.g., *yeager-guest*).
3. **Ignore middle names:** We removed middle names and initials if real names were provided. For example, *Jory A. Pratt* is converted to *Jory Pratt*. We use ‘*real names*’ to refer to the names that were used together with the emails or nicknames to commit code. There is no guarantee that they are the developers' actual names and they can be also abbreviated versions of their names.

¹⁰ This process was applied on the IRC authors and committers of the entire collected data, and not only on the study sample.

Table 4.3: Alias resolution in IRC authors and Committers.

Project	IRC authors		Committers		Mapped IRC authors to committers (IRC committers)
	Original	After alias resolution	Original	After alias resolution	
Apache Lucene	266	221	152	107	26
Mozilla Thunderbird	1,180	1,029	895	633	83
Ubuntu	13,645	10,657	6,724	5,961	186

4. **Name similarity:** We applied a string similarity algorithm, Levenshtein edit distance [136], [188], to resolve aliases within IRC and committers identifiers separately. We set a conservative similarity threshold of 80%.
5. **Email-like similarity:** If multiple slightly different names have the same email login names, we considered them a match. For example, *nicholas knize* and *nick knize* both have the email login name *nknize*. We excluded commonly used email login names such as *mozilla@email_domain*.
6. **IRC authors-committers mapping:** We mapped the final list of IRC identifiers to committers identifiers to detect IRC committers. For a more accurate mapping, we consolidated the available information with the additional identifiers provided on the official lists of contributors on the project website.

Manual corrections were applied for some cases. An overview of the IRC authors and committers for each project after resolving aliases and mapping IRC authors to committers is shown in Table 4.3. Although, we were able to resolve the majority of aliases within the single source (i.e., IRC and commit history), the mapping of IRC authors to project committers was not feasible in some cases. The main reason is that only the nicknames of IRC authors were available, while in most cases names or emails were used for committing code.

4.2.2 Results

This section presents the analysis results of the rationale frequency in developers' IRC messages and the contributors of rationale.

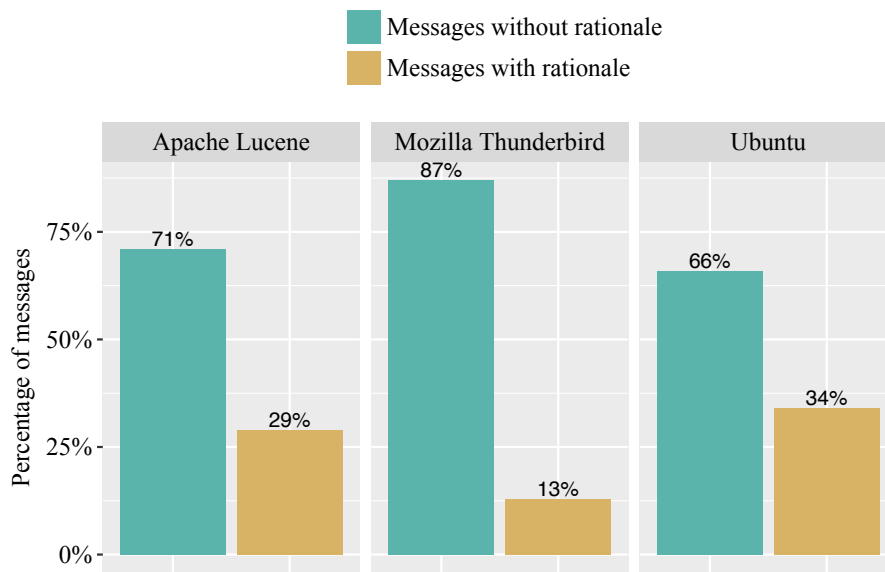


Figure 4.2: IRC messages containing rationale per project.

4.2.2.1 Rationale Frequency

To answer **RQ1** on the rationale frequency in IRC messages, we report on the results of the manual content analysis (see Section 4.2.1.3).

On average, 25% of the analyzed messages contain rationale with a total of 1,910 out of the 7,500 annotated messages. We found that rationale frequency varies among the analyzed OSS communities, as illustrated in Figure 4.2. Ubuntu IRC channel contains the highest number of messages containing rationale in our sample (34%), followed by Apache Lucene (29%), and Mozilla Thunderbird IRC channel which contain the lowest number of messages containing rationale (13%).

Chat channels of OSS projects contain over twice (25% on average) as much messages containing rationale as chat channels of co-located teams (see Chapter 3). A possible explanation for such increase is that OSS developers are geographically distributed and working across various time zones. Thus, OSS developers communicate and discuss development issues in IRC messages more often than co-located development teams, in which regular face-to-face meetings are more common.

Table 4.4: Frequency distribution of rationale elements across messages containing rationale per project.

Rationale element	Apache Lucene	Mozilla Thunderbird	Ubuntu	Total	IRC example
Issues	27%	24%	36%	30%	"[...] I think we should find a way to configure the real system packages properly; [...]"
Alternatives	37%	50%	33%	37%	"I'd rather just see php7 in experimental ASAP, and then slowly work out all rdeps until they all seem to more or less work, then just transition [...]"
Pro-arguments	19%	25%	19%	20%	"Maybe for later version support, we should do it like in StandardTokenizer"
Con-arguments	11%	18%	15%	14%	"But if you have lots of data this could make your GC on Solr go wild."
Decisions	17%	4%	15%	14%	"I had considered it, but it didn't fit with the design I had, so I ignored it [...]"

Table 4.4 presents the frequency distribution and examples of fine-grained rationale elements. Alternative is the most prevalent rationale element in developers discussions with a total of 37% among the three projects. A possible explanation is that developers use IRC channels to discuss proposed alternatives with other developers, in the form of *"So I have some ideas and I want to get your opinion"*, more often than other rationale elements. There are other complementary channels in OSS projects for reporting issues or documenting final decisions, such as issue tracking systems, which might affect their frequency in IRC messages. However, the argumentative discussions of possible alternatives happen mostly through written communication channels in the absence of regular face-to-face meetings. For example, messages like *"Hi guys..I want to patch the issue [Issue#]"* in which developers reference an already opened issue to discuss their solution alternatives were commonly encountered during the manual annotation.

The second most frequent rationale element is issue with a total of 30%, followed by pro-arguments (20%), and finally con-arguments and decisions with an equal frequency of 14%. Developers tend to provide pro-arguments supporting their proposed alternatives, e.g., *"to make it 100% correct it would need to be volatile"*, which might explain the higher frequency of pro-arguments. The two

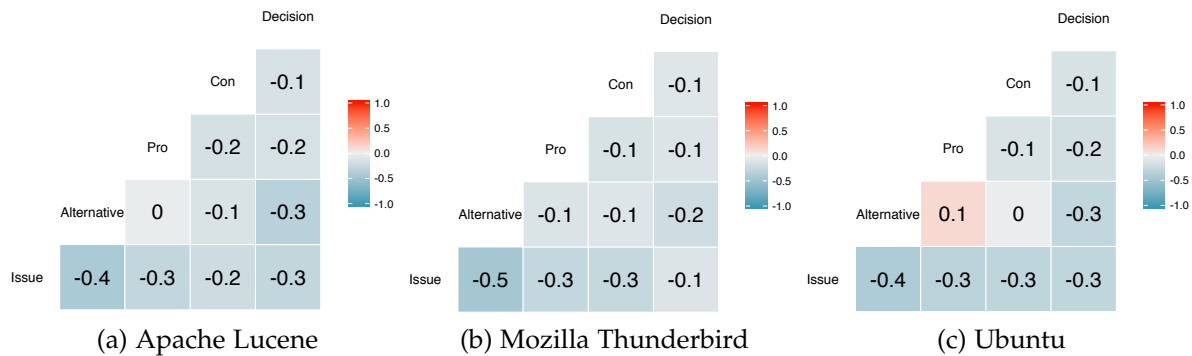


Figure 4.3: Pair-wise correlation matrices of rationale elements in IRC messages per project. The cells shading and color intensity visualize the sign and magnitude of the correlation.

coders agreed that identifying decisions was the most difficult among other rationale elements. We noticed that the decisions are usually not clearly stated in the messages even when a consensus is reached.

We found that 85% of the messages containing rationale only discuss one rationale element, 13% discuss two elements, and a few messages discuss three elements (1%). To gain further understanding on how developers discuss rationale in IRC messages, we analyzed the pair-wise co-occurrence correlation between different rationale elements at the message level. Figure 4.3 shows correlation matrices [60] that plot the Pearson's correlation coefficient [106] between all pairs of the rationale elements¹¹. The correlation coefficient measures the strength and direction of the relationship between two variables, i.e., two rationale elements in our case. In all three projects, there is a moderate negative co-occurrence correlation between issues and alternatives (Pearson's correlation ≤ -0.4), and with a lesser degree between issues and other rationale elements (Pearson's correlation ≤ -0.1). A possible interpretation of this result is that the message author who raised the issue wants to draw the attention of other developers and investigate different alternatives to solve the issue, rather than presenting the issue and the solution alternative to that issue in a single message. We also found a mild negative co-occurrence correlation between alternatives and decisions in all three projects (Pearson's correlation ≤ -0.2). When developers write the decision

¹¹ Equals to Phi coefficient for binary variables.

Table 4.5: IRC authors in the analyzed sample of IRC messages.

Project	IRC authors	
	IRC committers	IRC non-committers
Apache Lucene	14	20
Mozilla Thunderbird	27	43
Ubuntu	41	116

they made after a discussion in IRC or to inform other developers of a decision they took earlier, it is unlikely that they will quote other considered alternatives which explains the weak correlation between alternative and decision.

Most developers communicate on IRC messages through informal short messages. The short length of these messages could explain the absence of any strong correlations between the different rationale elements, as messages when containing rationale most likely contain only one element.

4.2.2.2 *Rationale Contributors*

In this study, we analyze messages from IRC channels dedicated to discussing development issues and we interpret the results with the underlying assumption that authors of these messages are developers contributing to the OSS project. However, IRC channels are public and anyone can join the ongoing discussion. Answering **RQ2** provides the opportunity to investigate how participation in rationale discussions in IRC messages is related to committing actual code changes.

We distinguish between two types of IRC authors (process described in Section 4.2.1.4). *IRC committers* are the IRC authors who were mapped into committers identified from the project commit history, and *IRC non-committers* are the IRC authors who were not mapped into committers.

Table 4.5 shows the distribution of the IRC authors of the analyzed sample of 7,500 messages (2,500 messages from each project). In all three projects, the number of IRC non-committers is greater than the number of IRC committers. This is typically expected due to the public nature of IRC channels in OSS projects. However, the developers make efforts to keep the discussions in the development channels focused on development matters, e.g., a developer replied to an

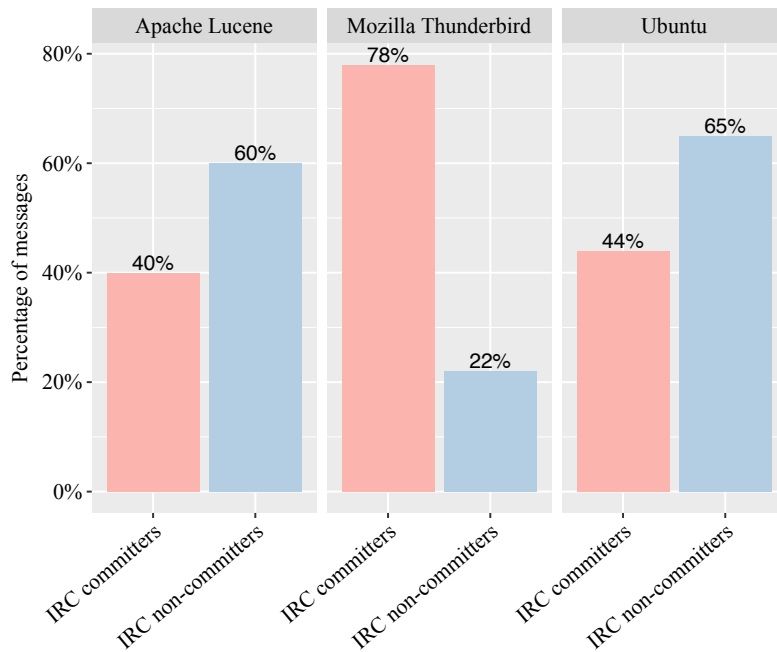


Figure 4.4: The percentage of messages containing rationale written by IRC committers and IRC non-committers.

end-user who posted a general question with “[...] please ask support questions in #ubuntu; I realize that it’s noisier there, but consider what it would be like here if everyone asked for user help here rather than in #ubuntu :-)”.

Figure 4.4 shows the percentages of messages containing rationale written by each group, IRC committers and IRC non-committers. In both Apache Lucene and Ubuntu, the percentage of messages containing rationale written by IRC non-committers exceeds the ones written by IRC committers. While in Mozilla Thunderbird, IRC committers wrote more messages containing rationale than IRC non-committers, even though the number of identified IRC committers is less than IRC non-committers. The percentage of rationale contribution is proportional to the number of messages written by each group. In both Apache Lucene and Ubuntu, IRC non-committers wrote more messages than IRC committers (58% and 60%, respectively); while IRC committers wrote more messages in Mozilla Thunderbird (60%). On average, IRC committers contributed 54% of the messages containing rationale in the analyzed IRC messages.

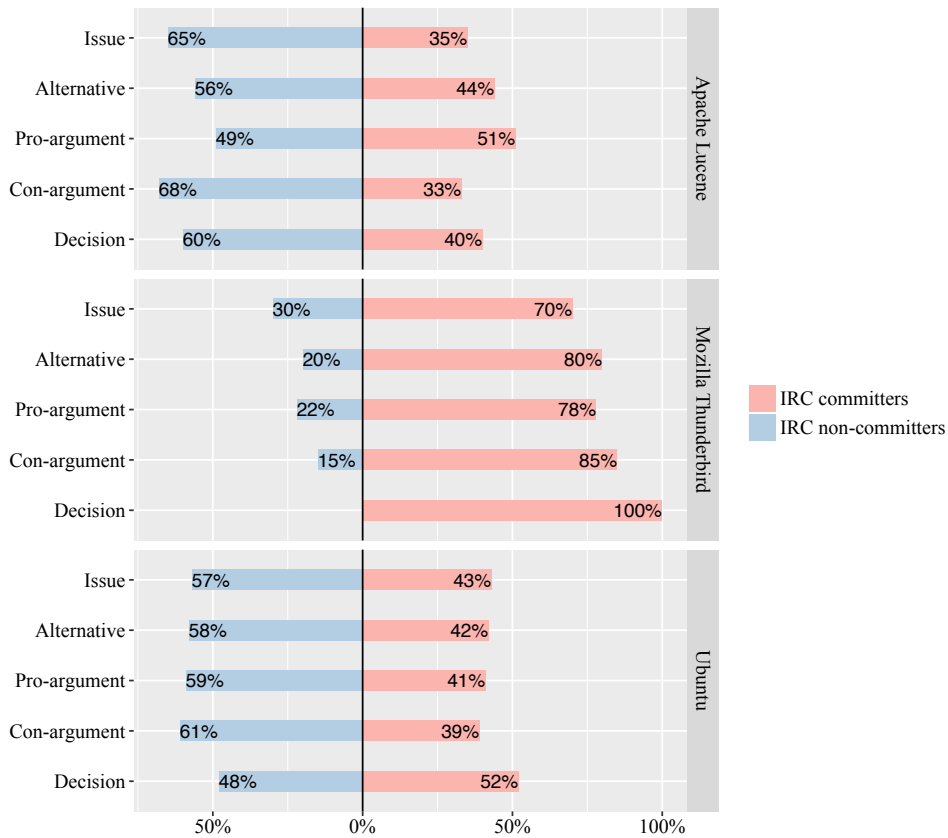


Figure 4.5: Rationale elements distribution per project.

To deepen our understanding of how different groups contribute to rationale discussions, we analyzed the distribution of the rationale elements among IRC committers and IRC non-committers, as shown in Figure 4.5. IRC committers contributed more decisions than IRC non-committers in both Mozilla Thunderbird and Ubuntu. A possible interpretation is that IRC committers have more influence on the project development and authority to make decisions. However, messages written by IRC non-committers contain more decisions than the IRC committers in Apache Lucene. For other rationale elements, there is no strong difference between the IRC committers and non-committers and the frequency of the rationale elements.

We found a strong positive correlation¹² between the number of commits and the number of messages containing rationale contributed by the IRC commit-

¹² Calculated using Spearman's correlation coefficient.

ter in Apache Lucene ($r_s=0.75$, p -value=0.02). However, we found a very weak ($r_s=0.15$, p -value=0.55) and weak ($r_s=0.26$, p -value=0.14) positive correlations between them in Mozilla Thunderbird and Ubuntu, respectively.

4.2.3 Discussion

We discuss the study findings by revisiting our research questions.

RATIONALE FREQUENCY: On average, 25% of developers' IRC messages contain development rationale. Alternatives are the most discussed rationale elements, followed by issues, arguments and finally decisions. We also found that techniques analyzing rationale in IRC messages should consider the context of the exchanged conversation. During our analysis, we observed that developers discuss rationale in a sequence of short messages with mostly one rationale element. With this in mind, recovering rationale as chunks of conversations allows for a better comprehension of the argumentation flow leading to the decision.

The use of IRC messages as a complementary channel with other communication mediums is apparent. Consequently, the rationale is fragmented across these channels. References to emails, issue trackers, and commits were common cases. Exploiting these references can be a first step towards the linkage of the rationale across different channels. Messages discussing particular commits or code branches could be employed to extract traceability links between code fragments and the rationale related to them in IRC messages.

RATIONALE CONTRIBUTORS: A developer might discuss the rationale behind an implementation when communicating with other developers through IRC messages. Linking the development activities of developers to their discussion is a first step towards recovering the related rationale. To achieve this, it is necessary to identify OSS developers across multiple communication channels.

In this study, we distinguished between two groups of IRC authors: IRC committers, who are committing code to project repository, and *IRC non-committers*, whose IRC identifiers could not be mapped to committers names. IRC committers contributed an average of 54% of the messages containing rationale in IRC channels. An interesting finding is that the volume of the rationale

contributed by each group is correlated to the number of messages written by that group (i.e., *IRC committers* or *IRC non-committers*) rather than *who* wrote the message. This posts an important question: *Who are the IRC non-committers?* If they are contributing to the rationale discussions, could they not be developers themselves?

We applied a set of name resolution heuristics and the Levenshtein edit distance algorithm [136], [188] with a conservative similarity threshold of 80%. Within the single source, i.e., IRC channels and code repositories, we resolved an average of 20% aliases. However, on average, only 9% of IRC authors were mapped to committers. This result might be due to the fact that developers use different identities on different channels and linking these identities is not always feasible. For example, in some cases, very short versions of the developers' names are used in IRC channels complicating its mapping to an actual developer name, e.g., *mvq*. Also using pseudonyms is common, e.g., *lovemoblender* and *blackbug*. Future improvements to the resolution method could explore different values for the threshold and follow the automated approaches with a manual inspection for resolving ambiguous cases.

In the ideal case, identification methods such as the one proposed by Robles and Gonzalez-Barahona[157] should be applied to track and maintain awareness of developers' activities across different project repositories. Such methods should take developers' privacy into account while being designed and applied.

4.2.4 Threats to Validity

In this section, we discuss the potential threats to the validity of our results according to the four validity aspects as defined by Runeson et al. [164].

CONSTRUCT VALIDITY is concerned with how accurately the study observations interpret and measure the theoretical constructs [164]. The messages coders have not been involved in the development of the analyzed projects. Thus, the consideration of whether a chat message contains rationale depends on the coders' judgment, which can be different from what the actual developers consider as rationale. To mitigate this threat, the coders read the description of the analyzed OSS and their main functionalities from their corresponding web sites.

Furthermore, both coders are graduate students with a software engineering background and they are users of some of the analyzed projects themselves.

Although the list of rationale elements used in our analysis are based on the well-known IBIS model [102], the list of elements could be incomplete and its descriptions simplified. This threat could lead to the capture of incomplete rationale. However, the analyzed rationale elements are shared among most rationale representation models.

INTERNAL VALIDITY is concerned with the confounding factors that may influence the study results [164]. We analyzed the messages through manual analysis by human coders which is a highly subjective process. To mitigate this threat, we applied a peer-coding process in which each message is annotated by two coders independently. Moreover, a coding guide was used during the coding process and the disagreements were discussed and resolved by the two coders.

We rely on an automated alias resolution approach to map IRC authors to committers. However, some aliases might remain unresolved. Also, the fact that developers can use freely-chosen nicknames might result in missing mappings between IRC authors and committers.

EXTERNAL VALIDITY is concerned with the generalizability of our results [164]. We selected popular OSS projects with a large community of users and developers, and thus, we cannot claim that our results are generalizable to other projects of smaller communities and different development settings.

Another threat to the external validity is the sampling bias. To mitigate this threat, we randomly selected a large sample of 7,500 messages from three OSS projects from three diverse domains.

RELIABILITY is concerned with to what extent the study results are dependent on a specific researcher, i.e., whether the study yields the same results if replicated by other researchers [164]. To encourage replication, we made the annotated IRC messages¹³ and the used coding guide (see Appendix B) publicly available to other researchers. Furthermore, peer-coding was performed on all

¹³ <https://figshare.com/s/20f10511dc6e36c98ccd>

the analyzed messages to minimize the bias that could result from individual coding.

4.3 RATIONALE IN DEVELOPERS' COMMENTS IN ISSUE TRACKING SYSTEMS

In Section 4.2, we analyzed how developers discuss rationale in the IRC messages of three OSS projects. We found that, developers discuss rationale in an average of 25% of the analyzed messages; despite the informal nature of IRC channels that contain a high volume of irrelevant and off-topic discussions [33]. In this section, we present an empirical study to analyze how developers discuss rationale in a more focused communication channel, the issue tracking systems used in the three OSS projects.

Issue tracking systems (ITS), sometimes referred to as bug tracking systems, are playing a significant role in software development, especially in open source projects as they provide a communication channel for geographically distributed developers [5], [191]. Issue tracking systems are typically used in OSS projects for discussing and documenting decisions [77], maintaining group awareness [65], and managing requirements [84].

Figure 4.6 shows an example of a reported issue in Ubuntu. In issue tracking systems, an issue usually consists of a title, a description, metadata (e.g., the issue reporter, assignee, status, and importance), and comments. A basic functionality of issue tracking systems as communication channels is their support for developers' discussions in the comments to the reported issues. Developers can discuss different aspects of the issue, explore possible solution approaches, and discuss tradeoffs in the comments to issues. In this study, we analyze the comments to issues rather than issues' descriptions, since our goal is to understand how developers discuss rationale while communicating with each other. We use the term *commenters* to refer to persons who post comments to issues in issue tracking systems.

Ubuntu
unity-webapps-reddit package

Overview Code **Bugs** Blueprints Translations Answers

Reddit webapp locks up firefox, consumes tons of CPU. 1

Bug #1273396 reported by [Robert Bruce Park](#) on 2014-01-27

This bug affects 1 person 6

Affects	Status	Importance	Assigned to	Milestone
unity-webapps-reddit (Ubuntu)	Confirmed	Critical	sanket	

+ Also affects project ? + Also affects distribution/package ? ? Nominate for series 2

Bug Description

Steps to reproduce:

1. make sure you have webapps components enabled (best to try this in a fresh VM)
2. open firefox
3. browse to reddit.com
4. choose to enable integration when prompted
5. notice that CPU usage spikes permanently and firefox becomes unusable / crashes. 3

+ Add tags ?

[Robert Bruce Park \(robru\)](#) on 2014-01-27

Changed in unity-webapps-reddit (Ubuntu):
importance:Undecided → Critical
assignee:nobody → Justin McPherson (justinmcp)

[Justin McPherson \(justinmcp\)](#) wrote on 2014-01-29: #1

With a new install of trusty-386 and updates applied, I can't reproduce this. There are two reddit icons, but I that's a different problem.

[Robert Bruce Park \(robru\)](#) wrote on 2014-01-29: #2

Are you using firefox or chrome?

This bug was so bad for such a long time that I had to disable webapps on my computer. I'll poke around a bit to see if I can reproduce it again. 4

Figure 4.6: An example of a reported issue in Ubuntu. It consists of: (1) issue title, (2) issue metadata, e.g., status, importance, reporter, and assignee, (3) issue description, and (4) comments.

4.3.1 *Study Design*

This section introduces the design of our empirical study. We describe the applied research method, including data collection, manual coding process, and mapping commenters to committers.

4.3.1.1 *Research Method*

We followed a research method similar to the one applied in the study of rationale in the IRC channels of OSS projects (see Section 4.2.1.1), which consisted of two phases: data collection and data analysis. In the data collection phase, we crawled the issue tracking systems of the same three OSS projects examined in the IRC messages study (see Section 4.2.1.1): Apache Lucene, Mozilla Thunderbird, and Ubuntu. The research data is described in Section 4.3.1.2.

In the data analysis phase, we applied content analysis techniques as described by Neuendorf [138] on a stratified sample of 300 issues, i.e., 100 issues from each project, to answer **RQ1**. The manual coding process is described in Section 4.3.1.3.

To answer **RQ2**, we obtained the lists of committers after resolving aliases from the previous study of IRC messages (see Section 4.2.1.4), as we are investigating the same OSS projects. We mapped the rationale contributors in the issues' comments to the code committers whenever feasible. The mapping process is described in Section 4.3.1.4.

4.3.1.2 *Research Data*

In this section, we describe the process of collecting and extracting our research data from three OSS projects: Apache Lucene, Mozilla Thunderbird, and Ubuntu¹⁴. Apache Lucene uses JIRA as its issue tracking system¹⁵, while Mozilla Thunderbird uses Bugzilla¹⁶, and Ubuntu uses Launchpad¹⁷. For each project, we crawled all the reported issues that have at least one comment, as we are interested in the analysis of rationale in these comments, extracted the required

¹⁴ An overview of these projects is given in Section 4.2.1.2.

¹⁵ <https://issues.apache.org/jira/projects/LUCENE/issues>

¹⁶ <https://bugzilla.mozilla.org/buglist.cgi?product=Thunderbird>

¹⁷ <https://bugs.launchpad.net/ubuntu/+bugs>

Table 4.6: Overview of Issue Tracking Systems Dataset.

Project	Years	Issues	Comments
Apache Lucene	2001 - Aug. 2017	7,239	74,012
Mozilla Thunderbird	2008 - Aug. 2017	4,368	26,587
Ubuntu	2004 - Aug. 2017	28,615	150,008
Total	—	40,222	250,607

fields, and stored them into a MySQL database for further analysis. For each issue, we extracted: title, description, reporter, assignee, and comments. For each comment, we extracted: comment, and comment author. Table 4.6 shows an overview of the collected issues and their comments. In total, we collected 40,222 issues with 250,607 comments from three OSS projects.

4.3.1.3 Coding of issue comments

To analyze rationale in the developers' discussions in the comments of issue tracking systems, two researchers, including the author of this dissertation, performed manual content analysis [138] on the comments of a stratified sample of 300 issues, i.e., 100 issues from each project. This process consisted of the following steps:

- A. **Developing a coding guide:** To systematize the coding process and minimize disagreements between the two annotators, a coding guide was developed and followed during the coding task (see Appendix C). The coding guide was developed in two iterations. In each iteration, the two annotators independently annotated comments of a randomly sampled 50 issues. Consequently, the disagreements between the two annotators were analyzed and the coding guide was modified.
- B. **Sampling of issues:** Generating the issues sample for the manual coding consisted of three steps:
 - STEP 1. Our aim was to analyze rationale in ITS in the same time period analyzed in the study of IRC channels to be able to compare between

Table 4.7: Issues coding sample.

Project	Issues	Comments	Sentences
Apache Lucene	100	1,023	2,446
Mozilla Thunderbird	100	989	3,086
Ubuntu	100	995	2,782
Total	300	3,007	8,314

the developers' use of ITS and their use of IRC for discussing rationale. To this end, we only considered the issues that were created within the time periods specified in Table 4.1.

STEP 2. The collected issues in our research data have a varying number of comments, ranging from one to 434 comments. However, only a small percentage of issues (less than 1% in all three projects) have more than 30 comments. Based on a preliminary inspection of some issues in our research data, we decided to consider issues that have between 5 and 30 comments. As our goal was to investigate how developers discuss rationale in these comments, we found that less than 5 comments usually lack any meaningful discussions; while in issues with above 30 comments, discussions often drift from the original issue topic.

STEP 3. We divided the issues in the dataset into five groups (strata) according to the number of comments: 5-10, 11-15, 16-20, 21-25, and 26-30. Next, we applied stratified random sampling to obtain 100 issues for each project, resulting in a sample of 300 issues from the three OSS projects. Table 4.7 gives an overview of the issues coding sample.

- c. **Manual coding of issue comments:** We chose comments' sentences to be the coding unit during the manual coding to allow for a more fine-grained annotation of rationale elements.

We performed peer-coding on the 8,314 sentences of all comments in the coding sample. The two coders independently inspected each sentence,

	A	B	C	D	E	F	G	H	I	J	K	L	M
	Issue ID	Issue Title	Issue Description	Issue Reporter	Issue Assignee	Comment Author	Senetences						
1			<p>Filter s and caching uses										
2	6435	LUCENE	transient	Matt Erics	Unassigne	switchhoo	This is a patch that will allow						
3						switchhoo	This is new and better vers						
4						switchhoo	A new version that will hope						

Figure 4.7: A screenshot of the Excel sheet as used during the manual coding of comments in ITS.

indicated if the sentence contains rationale, and specified the rationale element(s) present in the sentence. A sentence can contain more than one rationale element.

The sentences of all the comments for the sampled issues were presented to the coders in an Excel sheet for each project in a sequential order (one sentence in each row), as shown in Figure 4.7. Displaying all the comments of an issue in the excel sheet allowed the coders to obtain the conversation context while coding. In addition to the comments' sentences, the *issue title*, *issue description*, *issue reporter*, *issue assignee*, and *comment author* were displayed to provide the coders with the context in which these comments were exchanged. Rationale elements are represented by color-coded columns. The coder can indicate that a sentence contains a specific rationale element by assigning an x in the corresponding column. The two coders reported an average of 21 hours to complete the coding task.

- D. **Disagreements reconciliation:** Disagreements between the two coders occur when a comment is annotated as containing rationale by one coder or when a comment is annotated as containing different rationale elements by the two coders. The average inter-rater agreement between the two coders was 78%. The two coders discussed and resolved their disagreements.

4.3.1.4 Mapping rationale contributors to committers

In all three studied OSS projects, only registered users can post comments in ITS. This minimizes the multiple aliases problem faced earlier in IRC and committers identifiers (Section 4.2.1.4), as developers use their login names to post into ITS, hence, a single identifier within ITS. The login names identified in the dataset can be self-chosen nicknames, which can be abbreviations of the developers' real names, e.g., *dsmiley*. Alternatively, the developers' emails can be used as login names, such as *gbowyer@fastmail.co.uk*; while in Ubuntu, a combination of the developer's name and nickname is used (e.g., Adam McMaster (*adammc*)).

However, to map the rationale contributors in the comments of ITS to code committers, the multiple aliases problem is still present as developers might use different identifiers on these two channels, which complicates the mapping of their identifiers across these channels.

To map rationale contributors in the comments of ITS to committers, we used the list of committers after applying the alias resolution approach in Section 4.2.1.4. Next, we applied the following steps automatically on the list of commenters' identifiers to prepare them for the mapping:

1. **Extract email login names:** If the developer's email is used as the login name, we removed the email domain (i.e., anything after "@"). For example, *gbowyer@fastmail.co.uk* is converted to *gbowyer*.
2. **Ignore middle names:** We removed middle names and initials, when developer's name is used as the login name. For example, *Antti S. Lankila* is converted to *Antti Lankila*.
3. **Normalization:** We removed punctuation (e.g., "-" and "."), numbers, and extra white spaces. In addition, we removed "bugzilla" and "mozilla" that were attached to some login names in Mozilla Thunderbird, for example *benjamingslade+mozilla* is converted to *benjamingslade*.

Finally, we mapped the *rationale contributors* in the ITS comments (i.e., the commenters who wrote comments identified as containing rationale in the manual coding in Section 4.3.1.3) to committers whenever viable.

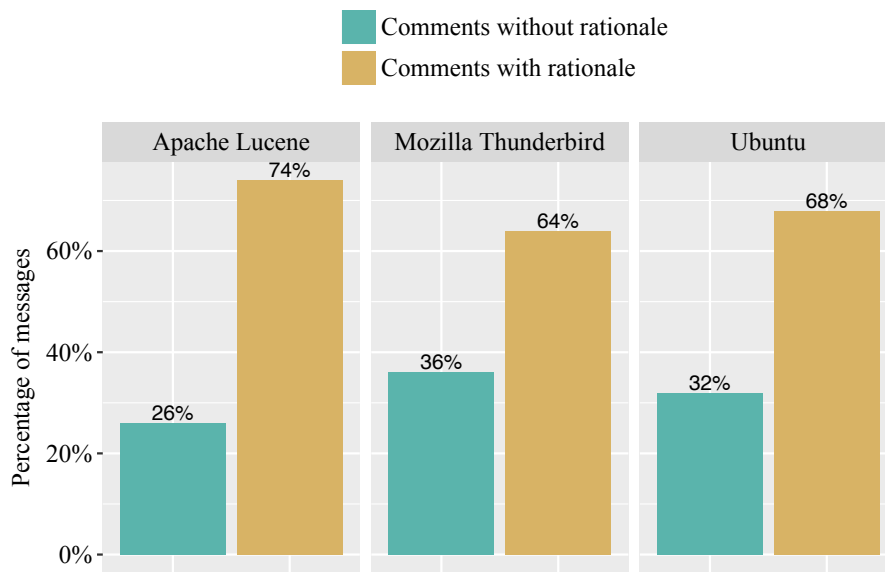


Figure 4.8: Comments containing rationale per project.

4.3.2 Results

This section presents the results of the manual content analysis of rationale elements in the comments in ITS and the mapping of rationale contributors in these comments to the code committers.

4.3.2.1 Rationale Frequency

To answer **RQ1**, we present in this section the results of the manual content analysis (described in Section 4.3.1.3).

Although we conducted the manual coding on the sentence level to allow for a more fine-grained annotation of rationale elements, we noticed that sentences mostly lack the argumentation context. Thus, we reported the results on the comment level. A comment was considered to contain a particular rationale element, if at least one of its sentences was annotated as containing that element.

Figure 4.8 shows the frequency of comments containing rationale among the analyzed comments from the three OSS projects. On average, 69% of the analyzed comments contain rationale (2,066 comments). Compared to an average of 9% of the chat messages of co-located teams and 25% of the chat messages of

Table 4.8: Frequency distribution of rationale elements across ITS's comments containing rationale per project.

Rationale element	Apache Lucene	Mozilla Thunderbird	Ubuntu	Total	Comment example
Issues	22%	62%	57%	46%	"How to reliably detect SSD?"
Alternatives	57%	44%	38%	47%	"Might it be advisable to have an alternative constructor that doesn't clone [...]"
Pro-arguments	46%	24%	21%	31%	"For higher values of minNumberShouldMatch it would probably be good to reuse the implementation from boolean queries."
Con-arguments	24%	24%	17%	22%	"So adding two extra object allocations to clone the incoming term is very unlikely to have noticeable impact on gc activity."
Decisions	25%	10%	16%	18%	"I would close this as won't fix and maybe only fix the remaining places that misses the file name."

distributed teams that were identified as containing rationale, developers' discussions in ITS contain more comments discussing rationale. This result might be expected, since issue tracking systems are more focused channels for discussing development issues; while due to the informal nature of chat channels, the exchanged chat messages contain higher volume of irrelevant and off-topic discussions, such as social aspects.

Table 4.8 presents the distribution frequencies of different rationale elements across the comments containing rationale per project and an example for each. Similar to chat message (in Chapter 3 and Section 4.2), alternative is the most discussed rationale element in the developers' comments (47% of the comments containing rationale discuss alternatives). A possible interpretation is that developers suggest possible solutions to address the issue that was already explained in the issue title and description. Issue is the next frequent rationale element which was discussed in 46% of the comments containing rationale. The developers request further clarifications or ask questions to better understand the reported issue. In addition, subsequent issues (e.g., sub-issues) and issues related to the original reported issues were discussed in the comment. For example, in *"I now remember that I had a very similar issue [...]"* a commenter described an issue in a comment. The number of comments arguing for a suggested alternative (pro-argument) exceeds the number of comments discussing con-arguments

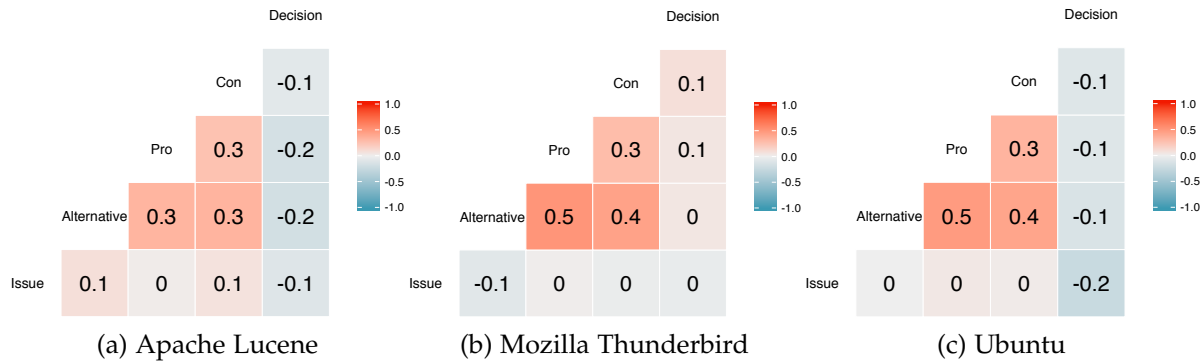


Figure 4.9: Pair-wise correlation matrices of rationale elements in ITS comments per project. The cells shading and color intensity visualize the sign and magnitude of the correlation.

with 31% of the former and 22% of the latter. Finally, decision appears in only 18% of the comments identified as containing rationale. This result echoes our earlier findings in that decision is the most difficult rationale element to identify in written communications.

Similar to the analysis conducted earlier on the IRC messages in Section 4.2.2.1, we analyzed the pair-wise co-occurrence correlation of different rationale elements at the comment level. The correlation matrices that plot the Pearson's correlation coefficient between all pairs of the rationale elements are shown for each project in Figure 4.9. In all three projects, there is a moderate positive co-occurrence correlation between alternatives and pro-arguments. In addition, we found a moderate positive co-occurrence correlation between alternatives and con-arguments, even though in a lesser degree than the correlation between alternatives and pro-arguments. Commenters tend to discuss the pro-arguments and con-arguments for an alternative in the same comment in which they discuss that alternative, e.g., *"I think it's generally useful to keep track of the time(s) in the OneMerge object"* and *"Doing an explicit levenshtein calculation here sort of defeats the entire purpose of having levenshtein automata at all!"*. However, no significant co-occurrence correlation was found between the other rationale elements.

Table 4.9: The results of mapping commenters in the study sample to committers.

Project	All commenters	Rationale contributors	
		Non-committing commenters	Committing commenters
Apache Lucene	70	41	22
Mozilla Thunderbird	130	161	31
Ubuntu	390	246	84
Total	648	417	137

4.3.2.2 Rationale Contributors

In issue tracking system of OSS projects, any user can register and use their login names to post comments. Thus, comments can be post by both end-users and developers who are committing actual code changes to code repositories.

To answer **RQ2**, we differentiate between two groups of rationale contributors in the ITS comments. *Committing commenters* are the commenters who were mapped into committers' names in Section 4.3.1.4; while *non-committing commenters* are the commenters who were not mapped into committers' names. Table 4.9 provides an overview of rationale contributors in the analyzed sample. On average, 87% of the commenters in the manually analyzed sample (of 3,007 comments) discuss rationale in their comments. This result emphasizes the use of ITS for a more focused development discussions compared to other informal written communication channels, e.g., IRC channels.

Committing commenters wrote 40% of the comments containing rationale in Apache Lucene, 44% in Mozilla Thunderbird, and 28% in Ubuntu. However, no significant correlation was found between the number of commits and the number of comments containing rationale contributed by the committing commenters ($r_s < 0.3$ in all three projects).

4.3.3 Discussion

We discuss the study findings by revisiting our research questions.

RATIONALE FREQUENCY: On average, 69% of the analyzed comments in ITS contain rationale. Compared to the two earlier studies that analyzed the same rationale elements in the chat messages of co-located teams (Chapter 3) and chat messages of distributed teams (Section 4.2), ITS systems contain more comments discussing rationale than the number of messages discussing rationale in the chat channels of both co-located and distributed teams.

Even though issue tracking systems contain more focused development discussions than chat channels, the discussion of rationale over these two channels shares three main characteristics. First, alternatives were the most discussed rationale element and developers tend to discuss the pro-arguments in support of these alternatives more than discussing the con-arguments. This result echos the findings of a practitioners' survey conducted by Tang et al. [183] that revealed that designers tend to document positive rather than negative arguments. Decisions were the most difficult rationale element to identify; and thus, the least frequently found during the manual analysis. One interpretation is that some of the decisions may be explicitly documented in other development artifacts, such as meeting minutes and commit messages, or can be inferred from other artifacts, such as source code and design models. However, the argumentation leading to these decision take place mostly over communication channels. Second, developers' written communications are context dependent; developers discuss related rationale elements in a sequence of messages or comments. We argue that preserving the context of the conversation in which these communication artifacts were exchanged is critical for capturing useful rationale information. One possible approach is capturing rationale as chunks of conversations to allow for a better understanding of the argumentation flow leading to the decision. Third, the developers' use written communication channels for discussing rationale in a complementary manner, i.e., they do not use one channel in isolation until the decision is made. Situations in which developers discuss an already opened issue in ITS in the chat channel by referencing its Issue ID or continuing mailing lists or chat discussions in the comments of issues were frequently

encountered during the manual analysis. As a consequence, rationale is fragmented across different channels. Rationale capturing approaches should exploit different traceability links to identify related rationale elements discussed across different channels.

RATIONALE CONTRIBUTORS: The long-term vision of capturing rationale is to make it available to the developers who need it to better inform different development activities [80]. One way to achieve this is by associating rationale to the part of the code that it explains [21]. Linking rationale discussions in written developers' communications to their development activities is a first step towards linking rationale to related code.

In our work, we studied the correlation between contributing to the rationale discussions in written communication channels, i.e., chat channels and ITS, and committing to the code repository. As we carried out our research in the context of OSS development team, we relied on mapping the identifiers used in the communication channels to the ones used for committing code to identify the *committers* and their committing frequencies. However, we found that an average of only 27% of the rationale contributors in ITS were identified as committers and they only contributed 38% of the comments containing rationale. This result might be due to the fact that developers use various identifiers across different development and communication channels which complicates the tracking and integrating of developers' related activities.

4.3.4 *Threats to Validity*

In this section, we discuss the potential threats to the validity of our results according to the four validity aspects as defined by Runeson et al. [164].

CONSTRUCT VALIDITY is concerned with how accurately the study observations interpret and measure the theoretical constructs [164]. The two coders have not been involved in the development of the analyzed projects. Thus, the consideration of whether a comment to an issue contains rationale depends on the coders' judgment, which can be different from what the actual developers consider as rationale. To mitigate this threat, the coders read the description of

the analyzed OSS and their main functionalities from their corresponding web sites. Furthermore, both coders are graduate students with a software engineering background and they are users of some of the analyzed projects themselves.

Although the list of rationale elements used in our analysis are based on the well-known IBIS model [102], the list of elements could be incomplete and its descriptions simplified. This threat could lead to the capture of incomplete rationale. However, the analyzed rationale elements are shared among most rationale representation models.

INTERNAL VALIDITY is concerned with the confounding factors that may influence the study results [164]. We analyzed the issues' comments through manual analysis by human coders which is a highly subjective process. To mitigate this threat, we applied a peer-coding process in which each comment is annotated by two coders independently. Moreover, a coding guide was used during the coding process and the disagreements were discussed and resolved by the two coders.

We rely on an automated alias resolution approach to resolve aliases in the list of committers that were used in the mapping of comments' authors to committers. However, some aliases might remain unresolved. In addition, developers may use different identifiers for writing in ITS and for committing code which might results in missing mappings.

EXTERNAL VALIDITY is concerned with the generalizability of our results [164]. We selected popular OSS projects with a large community of users and developers, and thus, we cannot claim that our results are generalizable to other projects of smaller communities and different development settings.

Another threat to the external validity is the sampling bias. To mitigate this threat, we applied stratified random sampling of 100 issues from three OSS projects from three diverse domains.

RELIABILITY is concerned with to what extent the study results are dependent on a specific researcher, i.e., whether the study yields the same results if replicated by other researchers [164]. To encourage replication, we made the

annotated ITS comments¹⁸ and the used coding guide (see Appendix C) publicly available to other researchers. Furthermore, peer-coding was performed on all the analyzed comments to minimize the bias that could result from individual coding.

¹⁸ <https://figshare.com/s/62da0027efcoe7ae35ao>

Related Work Relevant to Analyzing Text-based Developers' Communications

“Conversation within the digital medium has a property of great importance for our purposes: it persists.”

—Thomas and Kellogg [53]

In Chapter 3 and Chapter 4, we presented three empirical studies that analyze how developers discuss rationale in two text-based communication channels: chat channels and issue tracking systems, in co-located as well as distributed development teams.

In this chapter, we give an overview of the relevant existing research studying written developers' communications during software development. In particular, we focus the related work discussion in two areas: analyzing developers' chat messages, and analyzing issue tracking systems in software engineering.

The chapter is structured as follows: Section 5.1 presents related work analyzing developers' chat messages. Section 5.2 discusses related work analyzing issue tracking systems.

5.1 ANALYZING DEVELOPERS' CHAT MESSAGES

A growing body of literature has investigated the role of chat systems in software development. Storey et al. [181] carried out a survey of 1,449 developers to explore how developers use communication channels during software development. Chat messages were reported as the closest replacement for face-to-face interactions when team members are geographically distributed and were deemed as the most important communication channel by nearly 15 percent of the survey respondents. Lin et al. [117] surveyed software developers

to understand why developers use Slack, a team messaging platform, and how they benefit from it. Their analysis revealed that software developers use Slack for personal, community-wide, and team-wide purposes including communication and collaboration with other team members, file and code sharing, development operation notifications, and software deployment. Gutwin et al. [65] found that the group awareness in distributed development teams is maintained primarily through text-based communication, including chat systems. Dittrich and Giuffrida [45] explored the role of instant messaging in a global software development project and found that instant messaging not only supports the daily work and coordination between developers, but also provides means to build trust and social relationships with co-workers. In a systematic mapping study by the same authors, Giuffrida and Dittrich [63], on the use of social software in distributed development teams, instant messaging has been found to reduce communication barriers between remote collaborators. Handel and Herbsleb [70] analyzed the use of a group chat application by six globally distributed work groups of a software development organization. They found that group chat was primarily used for discussing and coordinating work activities. The findings of these studies that chat messages are playing an increasingly significant role in software engineering motivate our investigation of how developers discuss rationale in chat messages during software development.

Another stream of research has shown that the use of mailing list for development discussions is diminishing in favor of chat channels and issue tracking systems. Panichella et al. [142] investigated collaboration links by analyzing communication data from mailing lists, issue trackers, and chat logs of seven open source projects. The authors found that chat channels and issue trackers are being used as the main communication channels. Similarly, Kaefer et al. [90] and Guzzi et al. [66] provided empirical evidence that there is a shift in discussing development aspects towards an increasing usage of chat and issue tracking systems.

Developers' communication in the Internet Relay Chat (IRC) channels of Open Source Software (OSS) projects is mostly recorded and made publicly available, which attracted researches to study their role in software development as a valuable source of knowledge [71]. Yu et al. [195] investigated the use of IRC (synchronous) and mailing list (asynchronous) communication mechanisms in global

software development projects. They observe that developers actively use both as complementary communication mechanisms. Shihab et al. [172], [173] analyzed the usage of IRC meetings channels by developers to investigate the meeting content, meeting participants, their contribution and communication styles. They found that IRC meetings are increasing in popularity among OSS developers and maintainers. Elliott and Scacchi [50], [52] showed that open source communities use IRC channels, together with other communication channels such as mailing lists, to mitigate and resolve conflicts and to build a community. Elliott [51] studied developers discussions in IRC channels and interviewed some OSS developers to investigate the cultural beliefs and values influencing the OSS development. The author found that developers communication in IRC messages reinforces the cultural beliefs and motivation of OSS developers and contributes to the forming of the community.

There are relatively few studies investigating the role of chat systems in knowledge management during software development. Instant messaging is found to be one of the four most popular communication tools (together with mailing lists, telephone, and video conferencing) used to support knowledge sharing among software developers [196]. Ajjan et al. [2] recognized the need to better understand how the use of instant messaging affects knowledge management within the organization. The authors surveyed 117 knowledge workers to study the impact of the continuous use of instant messaging and found that it positively affects the knowledge creation, retention, and sharing. Dennerlein et al. [42] found that the effectiveness of messaging tools for knowledge management activities (generating, acquiring, organizing, transferring, and saving knowledge) is mainly affected by the intended usage of these tools rather than the context of application. Thus, a messaging tool is "what you make of it!" [42], rather than serving solely as a communication medium.

While these studies have focused on investigating the general use of chat systems during software development and their impact on general knowledge management activities, our work differs in that we analyze a specific type of knowledge, i.e., rationale, the completeness of the existing rationale, and the relation between development activities and rationale contribution in chat messages.

5.2 ANALYZING ISSUE TRACKING SYSTEMS

Issue tracking systems (ITS) are valuable sources for supporting and managing software development and maintenance activities [4], [57]. A number of studies have highlighted how issue tracking systems worked as a knowledge repository and evolving source of information during software development [68], [191]. In an expert survey of software developers from six different companies performed by Miesbauer and Weinreich [132], issue tracking systems were identified as one of the sources used in practice for documenting decisions during software development. Bertram et al. [10] conducted a qualitative study to analyze the use of issue tracking systems by small, co-located software development teams. They found that issue tracking systems serve as both a communication hub and an *outboard brain* of shared knowledge. Additionally, the majority of the study participants reported that viewing issue discussions in the comment history is important for understanding the rationale behind decisions. These studies motivate our investigation of the dialog in these comments as a valuable source of rationale.

Several studies have examined the use of issue tracking systems for documenting decisions and decision knowledge during software development. Hesse et al. [77] analyzed the decision-making strategies, the decision knowledge elements, and the relation between them by coding the comments of 260 issues from Firefox. They distinguished between rational decision-making (RDM), i.e., a systematic search for the optimal solution, and naturalistic decision-making (NDM), i.e., selecting a sufficient solution based on past experience and heuristics. Similar to our work, the authors coded the decision knowledge elements, i.e., rationale, in the comments of ITS. However, they classified rationale elements in the comments according to their own decision documentation model [78] consisting of: *questions, solutions, context, assumptions, constraints, implications, and pro-/con-arguments* supporting or attacking a given question or solution; while in our work, we based our classification of rationale elements on IBIS [102] (see Section 2.2.2). In particular, we analyzed *issues, alternatives, pro-/con-arguments, and decisions*¹ in the comments of issue tracking systems.

¹ Rationale elements are defined in Table 2.1 in Chapter 2.

Bhat et al. [11] analyzed design decisions by manually labeling 1,500 issues of two large open source projects. The authors focused their analysis on the architectural design decisions as classified by Kruchten [99] into existence decisions, property decisions, and executive decisions. There are three main difference between their study and the work presented in this dissertation. First, we did not distinguish between different types of decisions, but rather focused on analyzing the decisions made at various phases of software development. Second, beside decisions, we analyzed additional elements of rationale¹, such as issues and alternatives. Third, they analyzed the summary and description of issues to identify different decisions; while in our work, we analyzed the comments within issues as we are interested in analyzing the argumentation between developers.

In the same vein, Ko and Chilana [96] analyzed the rhetorical argumentation structure of the design discussions in the comments of bug reports by coding the comments of 100 bug reports from three OSS projects. They used six design concepts for classifying the comments, among them are *idea* (i.e., “a description of a change”), *rationale* (i.e., “grounds for a particular opinion”), and *decisions* (i.e., “event marking the closing of a report”). Rationale was identified as the most dominant type of comment in design discussions. Some of their analyzed design concepts are partially similar to the rationale elements analyzed in this dissertation, i.e., ideas, rationale, and decisions are equivalent to alternatives, pro- and con-arguments, and decisions in our work. However, they focused their analysis on *contentious* bug reports²; while we performed our analysis on a more general stratified sample of issues.

Rogers et al. [161], [162] studied bug reports as a potential source of rationale. The authors manually annotated 200 bug reports from Chrome project. They annotated the sentences of bug report descriptions looking for eight different rationale elements: *requirements*, *questions*, *alternatives*, *arguments*, *assumptions*, *answers*, *procedures*, and *decisions*. Only 10.9% of the analyzed sentences were found to contain rationale. The work described in this dissertation differs from their work in analyzing rationale in the comments rather than issues descriptions, as we are interested in the rationale deliberation among developers. In addition,

² Contentiousness of a bug report was measured based on the frequency of personal pronouns in the comments of the report.

the rationale elements analyzed in our work could be viewed as a subset of the elements analyzed in their work.

Collectively, these studies outline the important role played by issue tracking systems in documenting the rationale and argumentation behind various decisions made during software development. Our work is complementary; we focus on quantitatively analyzing rationale in the developers' discussions in issue tracking systems and the relation between development activities and rationale contribution in these discussions.

Part III

Rationale Capturing Methods in Text-based Developers' Communications

REACT: A Method for Capturing Rationale in Developers' Chat Messages

“The general strategy for getting design rationale into practice is to embody rationale capture in tools that are of immediate utility to designers.”

—Moran and Carroll [133]

In Chapter 3 and Chapter 4, we have analyzed how developers discuss rationale in the chat messages of co-located and distributed development teams. Our findings revealed that chat messages constitute a valuable source of rationale about the software system.

In this chapter, we present REACT (Rationale ExtrAction from Communication arTifacts), a novel lightweight method to capture rationale in developers' chat messages. REACT is designed to be integrated into developers' messaging platforms and enables developers to annotate the rationale present in their messages with emojis—pictographs commonly used in text-based communications. Developers can use REACT to (i) individually annotate their own messages, or (ii) collaboratively annotate messages posted by other team members.






The chapter is structured as follows: REACT approach is described in Section 6.1. Section 6.2 presents the evaluation of REACT in two studies: a short-term design task and in a medium-term software project. We discuss our results in Section 6.3 and the threats to validity in Section 6.4.

6.1 REACT METHOD

REACT is a lightweight method for capturing rationale in developers' chat messages. The method is based on the manual annotation of chat messages that

contain rationale by developers. We focus on capturing five rationale elements: *issues*, *alternatives*, *pro-arguments*, *con-arguments* and *decisions*. The rationale elements definitions are listed in Table 2.1.

REACT rationale annotations are designed as a set of emojis, one emoji for each rationale element. The annotations are:

-  for messages containing issues,
-  for messages containing alternatives,
-  for messages containing pro-arguments,
-  for messages containing con-arguments, and
-  for messages containing decisions.

Our use of emojis as rationale annotations was motivated by three factors. First, emojis are very popular in modern text communication and social media [49]. Users of messaging platforms use emojis for liking, voting, checking off to-do items, and sharing knowledge among team members¹. Second, emojis are well integrated with most modern messaging platforms which alleviate its intrusiveness and encourages its adoption by developers. Third, users of messaging platforms use emojis inline within their chat messages or to respond to messages posted by other team members which supports the collaborative annotations of messages among team members.

In the context of this dissertation, we implemented our method within Slack; a widely used messaging platform for exchanging chat messages among members of development teams. In 2017, Slack has over six million daily active users² and the number is growing rapidly due to the wide spectrum of services offered by Slack, including team messaging, archiving, and integrations to a broad range of services and bots. Figure 6.1 shows a screenshot of a conversation in Slack and describes its main components.

¹ <https://18f.gsa.gov/2015/12/08/using-emoji-for-knowledge-sharing/>

² <https://www.statista.com/statistics/652779/worldwide-slack-users-total-vs-paid/>

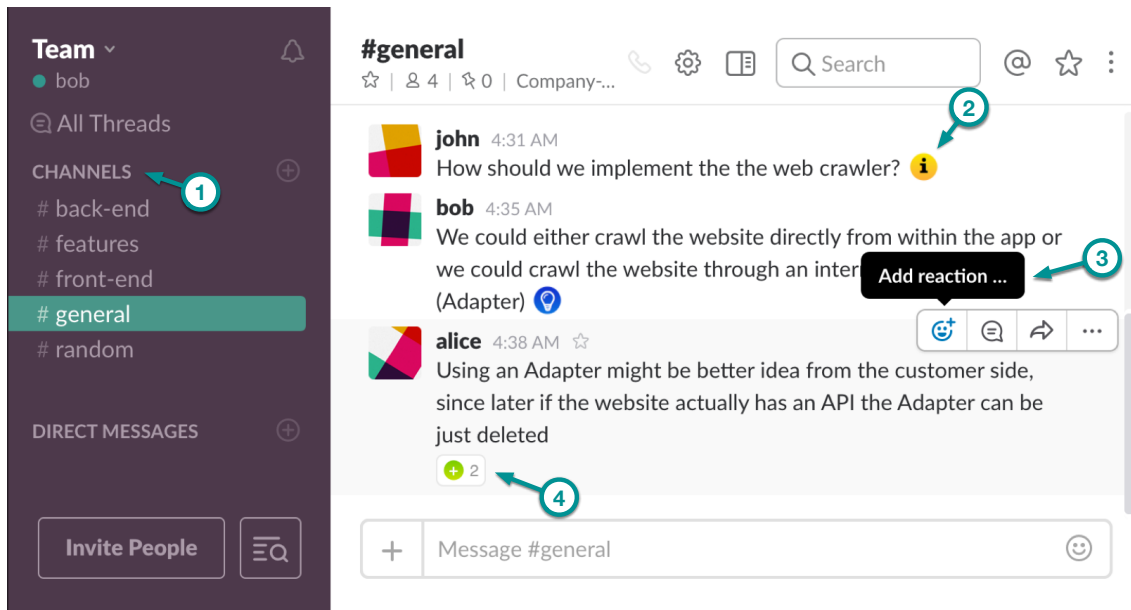


Figure 6.1: Example of a conversation in Slack. Teams using Slack can create (1) *channels* to organize their conversations according to topics, team members can include emojis (2) *inline* within messages, or as (3) a *reaction*, and (4) the added reactions appear under the message.

We added REACT rationale annotations as custom Slack emojis³. When using REACT, developers annotate their own chat messages with inline or reaction annotations. Developers can also annotate chat messages written by other team members using reaction annotations. A message can be annotated with more than one annotation, as a chat message might contain more than one rationale element. For example, when a developer proposes an *alternative* and writes the *pro-argument* supporting this alternative in the same message, the message should be annotated with both 🧠 and +.

6.2 REACT EVALUATION

We conducted two studies to evaluate REACT in different settings. In Study 1, we evaluated REACT during a short-term design task. In Study 2, we evaluated REACT in a medium-term project.

³ <https://get.slack.help/hc/en-us/articles/206870177-Create-custom-emoji>

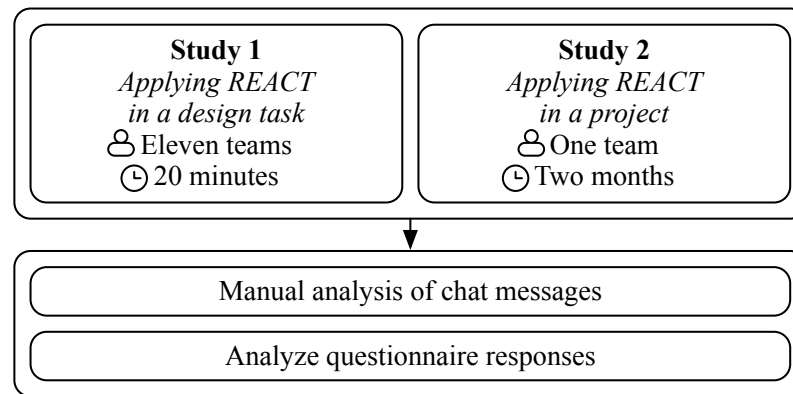


Figure 6.2: REACT evaluation method.






The study participants were part of a capstone course at the Technical University of Munich in 2017 [17]. During the course, students are organized in teams and develop mobile applications for industrial clients. Each development team consisted of eight to eleven students led by a project leader—usually a doctoral student—for management activities.

We evaluated REACT along the following dimensions:

1. *Correctness*: Do developers apply the correct annotations to their messages?
2. *Collaborativeness*: Do annotations encourage the collaborative capturing of rationale in chat messages?
3. *Privacy*: How do privacy concerns affect the annotation of rationale in chat messages?

We measure *Correctness* and *Collaborativeness* quantitatively by performing a manual analysis of all chat messages exchanged over the period of both studies. Additionally, we designed a questionnaire (see Appendix D) to qualitatively investigate *Privacy* and further understand developers' opinion on this issue after using REACT. Figure 6.2 shows the evaluation method. Detailed evaluation settings and results are presented in Section 6.2.1, Section 6.2.2, and Section 6.2.3, respectively.

Table 6.1: Frequency distribution of REACT rationale annotations and examples of annotated messages.

Rationale Annotation	Frequency		Example of Annotated Chat Message*
	Study 1	Study 2	
	11%	43%	"@USER hey there, should our option names also be localizable?"
	18%	36%	"@channel Hello everyone, I summarized and extended the first draft of the architecture that @USER and me created. Feel free to ask questions and come up with more detailed solutions and/or better ideas :slightly_smiling_face: _link_"
	30%	3%	"Using the server lets you use your own API. so you can use the crawling function with different OS (android)"
	18%	3%	"Looks nice here but there will be a black bar on top on a real watch. Not usable though since the icons are too tiny. The one I have now looks similar to the one on the left. You can check out the branch"
	23%	15%	"@channel when creating a String in code please do "let myString = NSLocalizedString("Text of my String", comment: "Some comment for the translation")" from now on for localization""

* We anonymized the messages but the essence of these messages is not affected.

6.2.1 Study 1: REACT in a short-term Design Task

The aim of this study was to evaluate whether REACT is an effective method for capturing rationale in chat messages.

6.2.1.1 Study Settings

Eleven development teams participated in the study. As a starting point in our study, we gave a brief tutorial to the study participants in which we introduced REACT and gave examples of applying it in Slack messages. Afterwards, we presented the participants the task of implementing a web crawler to systematically fetch information from a website. The eleven development teams were asked to use REACT to discuss the *issue*, evaluate the different *alternatives* and make a *decision* in their Slack team channel. When the development team ar-

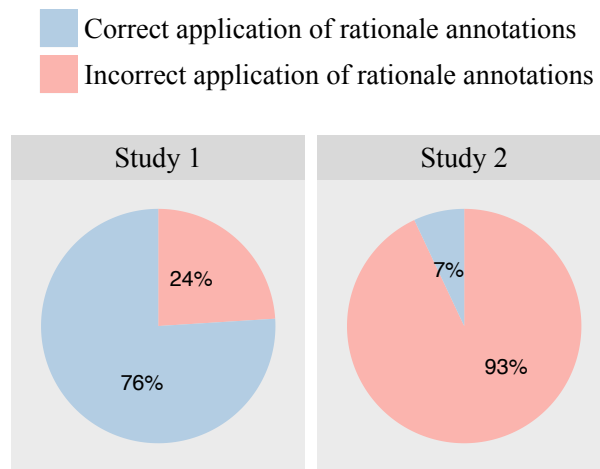


Figure 6.3: Correctness analysis.

rived at a decision, the design task was considered complete. The tutorial and design task took approximately 20 minutes. After the completion of the design task, two researchers, including the author of this dissertation, studied how the eleven development teams applied REACT in chat messages by analyzing the chat messages manually.

6.2.1.2 Study Results

Study participants applied a total of 342 REACT annotations to 421 chat messages. Table 6.1 (2nd column) lists the frequency distribution of REACT annotations in chat messages of the eleven teams during the design task.

To evaluate the correctness of the applied REACT annotations, we considered the annotation application to be incorrect when developers applied a rationale annotation that does not represent the rationale element(s) discussed in the message. For example, when a developer annotates a messages containing an *alternative* with an *issue* annotation instead of an *alternative*. Furthermore, when a message containing rationale was not annotated, we considered it as an incorrect application of the annotations because missing annotations contradict the primary goal of our approach of capturing rationale in chat messages. Figure 6.3 (Study 1) shows the percentages of correct and incorrect application of annotations in the eleven teams' messages. In 76% of the annotation applications, de-

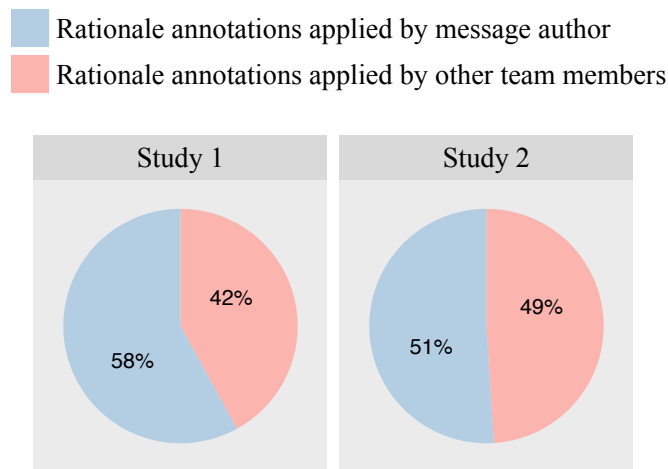


Figure 6.4: Collaborativeness analysis.

velopers applied them correctly for capturing rationale in their messages. This result implies an easy learning curve for applying REACT to annotate chat messages containing rationale knowledge.

With respect to collaborativeness, over half of the applied rationale annotations were applied by the message author (58%), as shown in Figure 6.4 (Study 1); while 42% of the annotations were applied by other team members. This result highlights the collaborative nature of rationale capturing in developers' chat messages as many decisions in software development are made collaboratively [72].

When comparing inline and reaction annotations, we found that 53% of the applied annotations were inline and 47% were applied as Slack reactions. A further analysis showed that message authors tended to use inline annotations when annotating their messages (91% of their annotations were inline). In the case of the reaction annotations, 88% were applied by other team members. This finding highlights the principal advantage of our approach in supporting the annotation of chat messages by their authors as well as collaboratively by other team members.

We examined how developers apply the different rationale annotations, i.e., inline versus reactions, and found that applying *issues*, *alternatives* and *con-arguments* as inline annotations were more prevalent, whereas *pro-arguments* and

decisions were applied more frequently as reactions. A possible explanation for this result is that when developers write *issues* and *alternatives*, they express their need to discuss and receive feedback from other team members. Similarly, they write *con-arguments* to oppose an alternative solution that was proposed by another team member. In these cases, developers might prefer to use inline annotations to attract the attention of other team members. On the other hand, *pro-arguments* and *decisions* are mainly used to show agreement with a statement or to collaboratively decide to select a proposed *alternative*; thus, reaction annotations were more frequent in these cases.

We examined other Slack reactions applied by developers to their chat messages to discover any patterns when applying these reactions. We found that “thumbs up” emojis were applied almost equally to the *pro-argument* annotations to show agreement when a message author proposed an *alternative* or made a *decision*. This result shows that developers are also using already existing and common-use emojis to express their opinions which can be utilized for capturing rationale.

6.2.2 Study 2: REACT in a medium-term Project

The aim of this study was to evaluate the viability of REACT when used by developers in the context of their daily development activities for a longer period of time (two months).

6.2.2.1 Study Settings

In this study, we introduced REACT to one development team of ten members who developed a mobile application for an industrial client. As different teams in the capstone course share similar development settings, we selected this team randomly. This team did not participate in Study 1 (Section 6.2.1). The study participants were introduced to REACT and to examples of applying REACT in Slack messages in a brief tutorial. Afterwards, the team used REACT in their daily exchange of chat messages for a duration of two months. For the duration of the study, participants were encouraged to ask for help or clarifications when faced with ambiguity while using REACT.

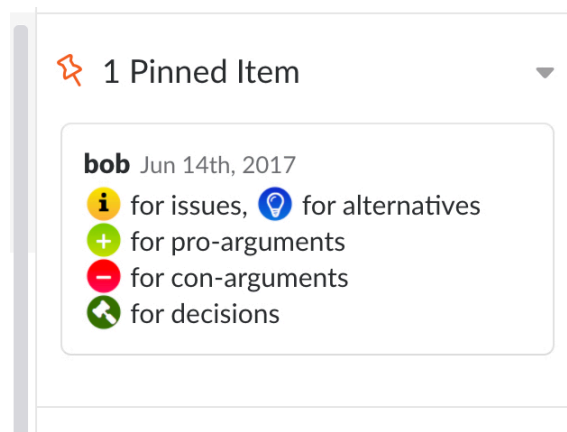


Figure 6.5: REACT rationale annotations pinned to team Slack channels in Study 2.

To organize the topics of their conversations, the team created three Slack channels: a *Main channel*, a *Front-end channel*, and a *Back-end channel*. In Slack channels, important messages or files can be pinned to details pane to make it always visible to team members. During the duration of the study, REACT rationale annotations were pinned to each channel as a constant reminder for developers to use them in their chat messages (as shown in Figure 6.5). Table 6.2 shows the number of chat messages analyzed per channel (excluding automatically generated messages by Slack bots such as reminders or status updates).

In this study, chat messages were exchanged during regular development activities throughout a medium-duration project. The rationale elements discussed in these chat messages were more complex and intertwined compared to Study 1 in which the development teams discussed a concrete and short-term design task (Section 6.2.1). Therefore, we needed systematic and more comprehensive analysis than the one performed in Study 1. For this reason, we decided to use content analysis techniques [138] on the team's chat messages.

Two coders, including the author of this dissertation, manually analyzed all the team messages exchanged during the study duration. The content analysis of the chat messages consisted of three steps. In the first step, we developed a coding guide⁴ that provides definitions and examples of rationale elements to train the coders and minimize disagreements. The coding guide was developed in two iterations. In each iteration, the two coders coded 500 messages indepen-

⁴ Similar to the one used for coding chat messages of distributed teams (see Appendix B).

Table 6.2: Chat messages analyzed in Study 2.

Slack Channel	Chat Messages
Main channel	969
Front-end channel	432
Back-end channel	297
Total	1,698

dently⁵. Then, the disagreement between the coders are resolved and the coding guide was modified accordingly. In the second step, the 1,698 messages were peer-coded manually by two coders. One of the coders was the project leader of the team, which allowed a more accurate and informed assessment of messages containing rationale. We used GATE (General Architecture for Text Engineering) [38] for the manual coding of chat messages⁶. During the coding task, the messages were shown without the applied REACT annotations, to avoid any bias. The average inter-rater agreement between the two coders was 90% for identifying messages containing rationale, and 87% for identifying different rationale elements. The two coders reported an average of 4.3 hours for the coding task. As a final step, the two coders discussed and resolved their disagreements.

6.2.2.2 Study Results

Figure 6.6 compares the number of team messages annotated by developers and the number of messages identified as containing rationale during the manual analysis per channel. During the manual analysis, 14% of the total messages were identified as containing rationale. However, developers applied REACT rationale annotations to 13% of these messages in the duration of the study (two months). The frequency distribution and examples of rationale annotations are listed in Table 6.1 (3rd column).

We found that only 7% of the messages containing rationale were annotated correctly by developers (shown in Figure 6.3 (Study 2)). A possible explanation for this result may be the lack of motivation for developers to annotate messages,

⁵ These messages are not part of the study.

⁶ A screen shot of using GATE for the manual coding is shown in Figure 3.1 in Chapter 3.

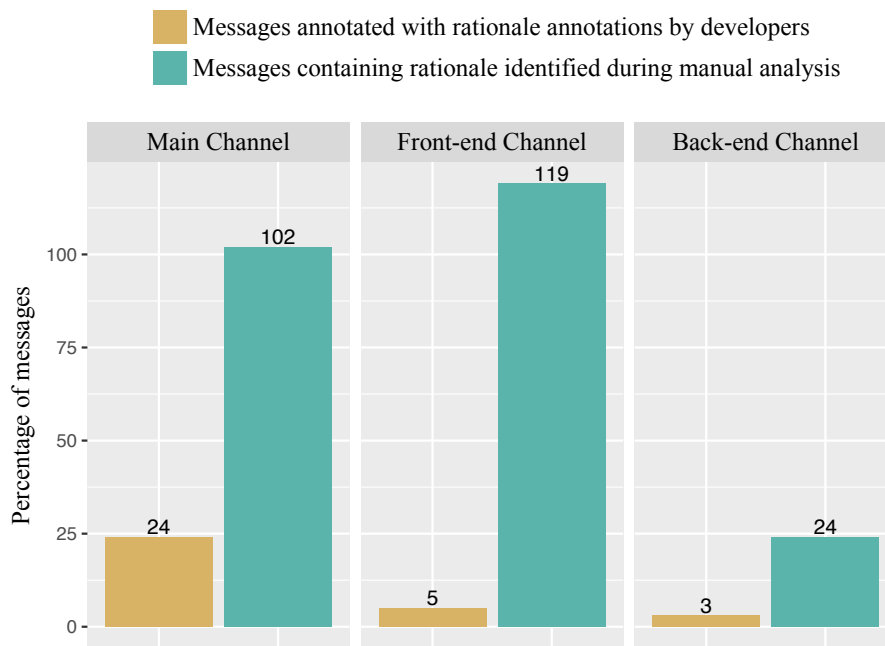


Figure 6.6: Rationale in team messages of Study 2.

even with a well-integrated capturing tool, if they cannot use captured rationale directly. Another possible explanation is that in the short-term Study 1, the design issue was clearly introduced to developers and they focused on discussing it and capturing the rationale in their discussions. However, in real software projects and under the pressure of meeting deadlines, identifying rationale elements in chat messages requires additional effort from developers.

When analyzing how developers apply other Slack emojis to the messages identified as containing rationale during the manual analysis, we found that the majority of the emojis added as reactions to the messages containing *alternatives* or *decisions* were expressing agreement to what is written in the message (68% and 57%, respectively). Examples of these emojis are: “thumbs up”, “ok hand” and “white check mark”. This result echoes the findings of Study 1 and could indicate that already existing emojis are relevant for capturing rationale.

As shown in Figure 6.4 (Study 2), rationale annotations were almost equally applied by message authors (51%) and by other team members (49%). Contrary to Study 1, we found that annotations were applied as reactions in the majority of their application (82%), while inline annotations were used in only 18%. Fur-

thermore, message authors applied inline and reaction annotations equally and 78% of the reaction annotations were applied by other team members. Overall, applying Slack reactions to annotate messages was dominant for all rationale elements in this study, which demonstrates the importance of the collaborative capturing of rationale.

6.2.3 Questionnaire

After the completion of the two studies, an online questionnaire was distributed to the study participants to obtain developers' opinion after using REACT (see Appendix D).

Of the two studies participants, 27 subjects—21 from Study 1 and 6 from Study 2—completed the questionnaire. From Study 1, 50% of the respondents agreed that REACT annotations are easy to learn, while 17% disagreed and 33% were neutral. From Study 2, 86% agreed that REACT annotations are easy to learn, while 5% disagreed, and 9% were neutral. In both studies, the majority of the respondents agreed that rationale annotations are simple to apply (62% and 67%, respectively). However, when asked whether they enjoy using rationale annotations, 33% from Study 1 respondents agreed, and 67% were neutral. In Study 2, 29% agreed, 24% were neutral, and 47% disagreed.

A possible interpretation of this result is the lack of motivation for developers to capture rationale, as one developer explained: *"I think the idea is great, but I don't feel like it generates an immediate value and therefore we often forget to use it."*

Over one-third of the questionnaire respondents agreed that rationale annotations help in documenting the rationale in their team chat messages (37%), whereas 33% disagreed, and 30% were neutral. One developer commented that *"Slack messages are highly context dependent"*, which confirms our earlier findings in that developers' chat messages are not to be used as the only source for rationale, but rather augmented with other development artifacts to capture a more complete rationale.

When the subjects were asked whether rationale annotations encourage all team members to participate in the ongoing discussion, 15% agreed, 44% disagreed, and 41% were neutral. An interesting observation is that while the majority of participants agreed that the proposed annotations are easy to learn and

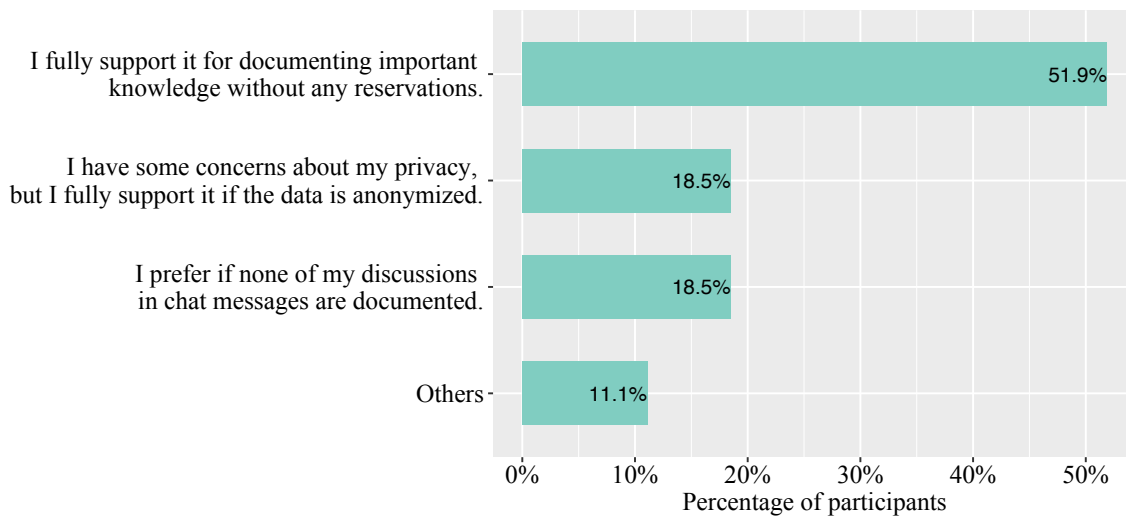


Figure 6.7: Privacy analysis.

simple to apply, the cognitive load of identifying rationale in messages is still perceived by developers as a burden and disruption to development activities; as illustrated by one respondent: *“Using rationale emoji makes you have to reconsider how you phrase messages, in such a way that it becomes a burden to continuously categorize each message”*.

Figure 6.7 presents participants responses when asked if they have any privacy concerns about annotating their chat messages. Surprisingly, over half of the respondents (51%) fully supported the annotations without any reservations. Over 18% of the respondent preferred anonymizing their messages before analysis, and an equal percentage of respondent (18%) preferred that none of their messages are documented. This interesting observation might support the analysis of developers’ chat messages as a source of valuable rationale knowledge. However, this calls for further investigation in other industrial settings in which privacy could be considered more pivotal.

6.3 DISCUSSION

Motivated by our findings that developers’ chat messages constitute a valuable source of rationale during software development, we designed REACT with the aim of supporting developers in capturing rationale in these messages.

The results of our evaluation show that REACT represents a step towards an effective rationale capturing in developers' chat messages. The usefulness of REACT as a rationale capturing method is manifested in three main aspects. First, it can be easily integrated into most modern messaging platforms; thus, lowering its adoption barrier by developers. Second, it is a lightweight method as the evaluation results show that rationale annotations are easy to learn and simple to apply, which alleviates the developers' resistance provoked by heavy-weight capturing approaches. Third, it supports the collaborative capturing of rationale, which mirrors the collaborative nature of decision-making in software development [132].

However, the effectiveness of REACT is highly dependent on providing immediate benefit for developers to justify the efforts of capturing rationale. Additionally, categorizing rationale in chat messages to apply REACT annotations still presents additional cognitive load for developers and perceived as an interruption to the communication flow. The need to think about whether a message contain rationale while writing it may represent a mental disturbance to the developer and causing what is known in communication theory as *communication noise*, which refers to anything interfering with the communication process [29], [139].

Contrary to expectations, the majority of participants did not view privacy as a major concern while using REACT. This echos the findings of Storey et al. [181] who noted, in their survey on the use of communication channels in software development, that "privacy is not a big concern for everyone, whereas being interrupted and feeling overwhelmed by communication traffic are issues for more developers".

Our finding that an already existing emojis are already used by developers to express their opinions raises the possibility that these emojis, while used informally by developers, might be exploited for recovering part of the discussed rationale within development teams.

6.4 THREATS TO VALIDITY

In this section, we discuss the potential threats to the validity of our results according to the four validity aspects as defined by Runeson et al. [164].

CONSTRUCT VALIDITY is concerned with how accurately the study observations interpret and measure the theoretical constructs [164]. The list of rationale elements could be incomplete and its descriptions may be simplified. However, the rationale elements used throughout this dissertation are based on the well-known IBIS model [102], and they are shared among many rationale representation models.

INTERNAL VALIDITY is concerned with the confounding factors that may influence the study results [164]. We relied on an error-prone human judgment for categorizing rationale in chat messages. To mitigate this risk, all the messages were peer-coded by two coders and a coding guide was created.

EXTERNAL VALIDITY is concerned with the generalizability of our results [164]. We evaluated REACT in a university course which might affect the generalizability of our results. However, students worked closely with industrial clients in settings similar to industrial ones. In addition, we evaluated REACT within Slack, a common communication tool in industrial projects. We believe that this evaluation gives insights for further replications of the study in practice.

RELIABILITY is concerned with to what extent the study results are dependent on a specific researcher, i.e., whether the study yields the same results if replicated by other researchers [164]. Although, the analyzed messages could not be made available due to privacy issues, the coding guide with clear definitions and examples of the rationale elements is made available to other researchers (see Appendix B). Furthermore, peer-coding was performed on all the analyzed messages to minimize the bias that could result from individual coding.

A-REACT: An Automated Rationale Extraction Method

“The Rationale Paradox: When most rationale is created, chances to capture it are lowest.”

—K. Schneider [168]

In the previous chapter, we presented REACT, a lightweight method for capturing rationale in developers’ chat messages. Our evaluation results reveal that even though REACT is easy to learn and simple to apply, categorizing rationale in chat messages still presents a cognitive overload and disturbance to the communication process for developers.

To reduce the cognitive overload on developers and to support recovering rationale from archived developers’ communications, this chapter presents A-REACT (Automated Rationale ExtrAction from Communication arTifacts), an automated method for detecting rationale in developers’ written communications and classifying them into the different rationale elements: issues, alternatives, pro-arguments, con-arguments, and decisions¹.

We evaluated A-REACT on the written developers’ communications analyzed in Part II of this dissertation:

CHAT MESSAGES OF CO-LOCATED TEAMS: The analysis and manual annotation of this dataset is described in Chapter 3.

CHAT MESSAGES OF DISTRIBUTED TEAMS: The analysis and manual annotation of this dataset is described in Section 4.2 in Chapter 4.

COMMENTS IN ISSUE TRACKING SYSTEMS OF DISTRIBUTED TEAMS: The analysis and manual annotation of this dataset is described in Section 4.3 in Chapter 4.

¹ The rationale elements are defined in Table 2.1 in Chapter 2.

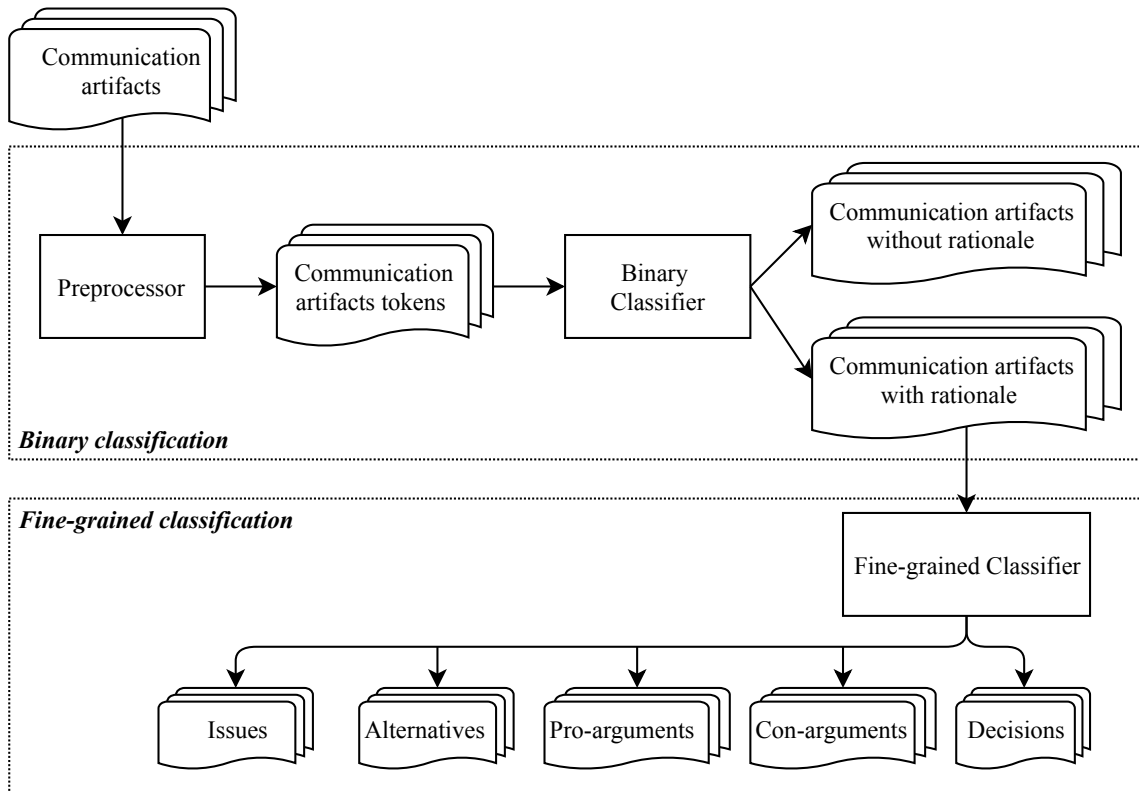


Figure 7.1: Overview of A-REACT.

In this chapter, we use the general term *communication artifacts* to refer to different types of written developers' communications. Even though we evaluate A-REACT in this dissertation on two communication artifacts, i.e., chat messages and comments in ITS, A-REACT is designed to be applicable to different types of written developers' communications.

The chapter is structured as follows: Section 7.1 introduces A-REACT. Section 7.2 describes the A-REACT evaluation settings and results. We discuss our findings in Section 7.3 and threats to validity in Section 7.4.

7.1 A-REACT METHOD

The goal of A-REACT is to automatically detect written developers' communication artifacts that contain rationale and classify them into different rationale

elements: issues, alternatives, pro-arguments, con-arguments, and decisions. An overview of A-REACT is shown in Figure 7.1.

A-REACT applies text mining and supervised machine learning techniques² to classify written developers' communication artifacts on two granularity levels:

1. *Binary classification*: In this step, a binary classifier detects communication artifacts containing rationale and filters out communication artifacts without rationale.
2. *Fine-grained classification*: In this step, the communication artifacts classified as containing rationale in the previous step are further classified into different rationale elements: issues, alternatives, pro-arguments, con-arguments, and decisions.

A-REACT consists of the following steps. First, the communication artifacts are preprocessed to transform them into a representation that is suitable for the machine learning algorithms. The communication artifact is preprocessed by applying the following steps: (1) converting it into lowercase, (2) reducing inflected words into their stems³, (3) applying n-gram tokenization, and (4) converting it into a vector space model by applying TF-IDF as a weighting method⁴. Tokenization converts a stream of characters into a sequence of tokens. We used n-gram tokenizer with 1 and 3 as the minimum and maximum length. By applying an n-gram tokenizer, we expected patterns of terms appearing together to be indicators of rationale presence in the chat messages, e.g., phrases like “*I would suggest*”, “*how about*” could be indicators of proposed alternatives, and “*how do we*” could be an indicator of issues. We chose not to apply stopword removal (i.e., removing common words in English) as we expected them to be representative of some rationale elements. For example, *which* and *how* might be indicators of issues, e.g., “*Which design pattern should we apply?*”, and *but* is commonly used before stating con-arguments against alternatives, e.g., “*but it sucks as UX*”. Second, communication artifacts are classified into communication artifacts *with rationale* and communication artifacts *without rationale* by applying binary classification⁵ on the preprocessed communication artifacts. Third, the communication

² An overview of these techniques is given in Section 2.3 in Chapter 2.

³ “weka.core.stemmers.LovinsStemmer” was used for the stemming.

⁴ The preprocessing steps are described in Section 2.3 in Chapter 2.

⁵ Binary and multi-label classification of text documents are described in Section 2.3 in Chapter 2.

artifacts that were classified as containing rationale in the previous step are further classified into different rationale elements: *issues*, *alternatives*, *pro-arguments*, *con-arguments*, and *decisions*, by applying multi-label classification⁵; as one communication artifact might contain more than one rationale element.

7.2 EVALUATION

In this section, we describe the evaluation of A-REACT on three different communication artifacts: chat messages of co-located teams (in Section 7.2.1), chat messages of distributed teams (in Section 7.2.2), and comments in issue tracking systems of distributed teams (in Section 7.2.3).

We used WEKA⁶ (Waikato Environment for Knowledge Analysis) [67] and MEKA⁷ (A Multi-label Extension to WEKA) [149] for the preprocessing of the communication artifacts and for building the classification models used in the approach evaluation.

7.2.1 Chat Messages of Co-located Teams

In this section, we used the 8,702 manually annotated chat messages of the three co-located development teams from Chapter 3 as a truth set for the training and evaluation of A-REACT.

7.2.1.1 Evaluation Settings

CLASSIFICATION LEVEL We performed the classification on the message level. The consideration of the classification on the message level was motivated by two factors. First, previous work on classifying development artifacts [6], [162] found that considering a sentence's neighbors (context) improved classification performance. Second, our analysis in Chapter 3 revealed that developers tend to discuss rationale in messages of short length, which makes classification on the message level feasible.

⁶ <http://www.cs.waikato.ac.nz/ml/weka>

⁷ <http://waikato.github.io/meke/>

CLASSIFICATION ALGORITHMS We compared the performance of two machine learning algorithms, MNB (Multinomial Naive Bayes)⁸ and SVM (Support Vector Machine)⁹ due to their popularity and good performance in text classification [33], [58], [170].

DATA BALANCING Our dataset is imbalanced with an average of 9% messages containing rationale. Building a classifier from an imbalanced dataset can cause the classifier to be biased towards the majority class, i.e., the class with the greater number of instances, while ignoring the minority class [69]. As previously described in Chapter 2, two popular techniques for handling class imbalance problem are under-sampling and SMOTE (Synthetic Minority Oversampling Technique). Under-sampling [46] uses a subset of the majority class for training the classifier; while SMOTE [31] applies oversampling on the minority class by generating synthetic examples. We compared between the application of under-sampling and a combination of SMOTE and under-sampling as previous research has proved that classifiers achieve better performance when combining both sampling techniques [31].

TRAINING AND EVALUATION For training and evaluating the classifiers, we applied 10-fold cross validation (a k-fold cross validation, as described in Section 2.3 in Chapter 2, in which k=10). We also applied project cross validation to test the generalizability of the results, as suggested by previous research [6], [143], since different development teams tend to use different terminologies and jargons in their communications. In our case, project cross validation is a 3-fold cross validation with one fold per project, as we are analyzing chat messages from three projects. The classifier is trained on the messages of two projects and tested on the messages of the third project. The process is repeated three times rotating the projects.

We evaluated the classification performance using the standard metrics in machine learning: *precision*, *recall*, and *F1-Measure* [54].

⁸ We used “weka.classifiers.bayes.NaiveBayesMultinomial” implementation of MNB.

⁹ We used “weka.classifiers.functions.SMO” implementation of SVM, with a polynomial kernel.

Table 7.1: Binary classification results of chat messages of co-located teams. P is Precision, R is Recall, and F1 is F1-measure.

Configuration	Classifier	Chat messages	Team A			Team B			Team C		
			P	R	F1	P	R	F1	P	R	F1
Imbalanced	MNB	With rationale	0.44	0.59	0.50	0.44	0.61	0.51	0.34	0.50	0.40
		Without rationale	0.96	0.93	0.95	0.95	0.91	0.93	0.96	0.92	0.94
	SVM	With rationale	0.55	0.34	0.42	0.59	0.34	0.43	0.53	0.30	0.38
		Without rationale	0.94	0.98	0.96	0.92	0.97	0.95	0.95	0.98	0.96
Under-sampling	MNB	With rationale	0.60	0.99	0.75	0.59	0.99	0.74	0.58	0.98	0.73
		Without rationale	0.97	0.34	0.50	0.97	0.30	0.46	0.95	0.30	0.46
	SVM	With rationale	0.73	0.82	0.77	0.76	0.79	0.78	0.78	0.80	0.79
		Without rationale	0.79	0.70	0.74	0.79	0.75	0.77	0.79	0.78	0.79
SMOTE + Under-sampling	MNB	With rationale	0.83	0.81	0.82	0.80	0.87	0.83	0.75	0.83	0.79
		Without rationale	0.81	0.84	0.82	0.86	0.78	0.82	0.81	0.73	0.77
	SVM	With rationale	0.87	0.75	0.81	0.92	0.76	0.83	0.90	0.72	0.80
		Without rationale	0.78	0.89	0.83	0.79	0.93	0.86	0.77	0.92	0.84

TRANSFORMATION METHODS When classifying messages containing rationale into different rationale elements (i.e., the fine-grained classification), a chat message may contain more than one element. For example, a developer might propose an alternative and write the pro-argument supporting the alternative in the same message. In machine learning, classifying documents into one or more classes that are not mutually exclusive is referred to as multi-label classification. We applied two of the most popular techniques for multi-label classification¹⁰, the BR (Binary Relevance) and the LP (Label Powerset) [187].

¹⁰ The multi-label classification and the transformation methods are described in Section 2.3 in Chapter 2.

Table 7.2: Examples of binary classification results of chat messages in co-located teams.

Chat message	Manual classification	Automatic classification
"Morning guys. Can someone tell me, what steptype (cooking, measuring, mixing, chopping) preheating the oven is?:/I would make another category - preheating otherwise I would take cooking as the type. What do you guys suggest?"	With rationale	With rationale
"You can send it to me on HipChat :)"	Without rationale	Without rationale
"Do we need to support the iPhone4?"	With rationale	Without rationale
"My big problem right now is that I would need photoshot and illustrator :("	Without rationale	With rationale

7.2.1.2 Evaluation Results

BINARY CLASSIFICATION RESULTS Table 7.1 gives an overview of the binary classification results per team. The numbers in bold represent the corresponding top values. When using the imbalanced dataset for training the classifiers, both MNB and SVM classifiers performed well in classifying chat messages without rationale, achieving high values for both precision and recall (above 0.91) for all three teams. However, they performed less well when classifying messages with rationale; where SVM had a better precision ranging from 0.53 to 0.59 (compared to a precision ranging from 0.34 to 0.44 when applying MNB); while MNB achieved a much higher recall ranging from 0.50 to 0.61 (compared to a recall ranging from 0.30 to 0.34 when applying SVM). A possible explanation for achieving less accuracy in classifying messages with rationale is the sparsity of the messages containing rationale in the results of our manual content analysis (see Chapter 3).

Upon further inspection, we found that rationale discussions spread over multiple messages were a common source of error. In these cases, it is important to consider the contextual information in the neighbor messages to identify the rationale contained in the message. Table 7.2 shows examples of binary classification results when applying MNB with imbalanced dataset. In summary, both

Table 7.3: Project cross validation results of chat messages of co-located teams. P is Precision, R is Recall, and F1 is F1-measure.

Classifier	Chat messages	Train: Teams B, C Test: Team A			Train: Teams A, C Test: Team B			Train: Teams A, B Test: Team C		
		P	R	F1	P	R	F1	P	R	F1
MNB	With rationale	0.23	0.70	0.35	0.30	0.68	0.42	0.22	0.67	0.33
	Without rationale	0.97	0.78	0.87	0.95	0.81	0.88	0.97	0.82	0.89
SVM	With rationale	0.32	0.22	0.26	0.55	0.25	0.35	0.44	0.27	0.34
	Without rationale	0.88	0.90	0.89	0.92	0.98	0.94	0.95	0.97	0.96

classifiers reported significantly better performance in classifying chat messages without rationale, with the highest achieved precision of 0.96 and recall of 0.98.

Applying balancing techniques resulted in a significant increase in the classification performance of messages with rationale. However, as expected, the performance of classifying the majority class, i.e., messages without rationale in our case, decreased. Applying under-sampling alone increases the precision (up to 0.78) and recall (up to 0.99) of classifying messages with rationale, with MNB achieving better recall and SVM achieving better precision, in all three teams. However, applying a combination of SMOTE and under-sampling achieved a higher F1-measure (ranging from 0.79 to 0.83) for classifying messages with rationale in all three teams, i.e., a better balance between precision and recall. Similar to the classification results of imbalanced dataset, MNB has a better recall for classifying messages with rationale (up to 0.87), while SVM has a better precision (up to 0.92) in all three teams. These results suggest applying data balancing techniques on the training set to alleviate the classifier bias towards the majority class as a result of imbalanced training data. Consequently, increasing the amount of detected rationale from chat messages.

When applying project cross validation, the overall classification performance of messages containing rationale decreased, as shown in Table 7.3. This provides a more reliable test of the degree to which the trained classifiers are able to classify rationale in the chat messages of unseen projects, i.e., not used for training the classifiers. This finding demonstrates that training the classifiers with a set of

Table 7.4: Fine-grained classification results of chat messages of co-located teams. P is Precision, R is Recall, and F1 is F1-measure.

Rationale element	Binary Relevance						Label Powerset					
	MNB			SVM			MNB			SVM		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Issue	0.38	0.46	0.42	0.53	0.46	0.49	0.37	0.47	0.41	0.52	0.47	0.50
Alternative	0.64	0.68	0.66	0.67	0.63	0.64	0.60	0.64	0.62	0.64	0.71	0.67
Pro-argument	0.42	0.52	0.46	0.47	0.41	0.44	0.37	0.55	0.44	0.51	0.42	0.46
Con-argument	0.32	0.44	0.37	0.41	0.36	0.38	0.26	0.41	0.32	0.43	0.28	0.34
Decision	0.27	0.39	0.32	0.32	0.25	0.28	0.26	0.40	0.31	0.28	0.14	0.19

annotated messages from the same project yields a more accurate classification results.

FINE-GRAINED CLASSIFICATION RESULTS We evaluated the performance of the fine-grained classifier that further classifies the 752 messages containing rationale, among the 8,702 manually analyzed messages, into the five different rationale elements: issue, alternative, pro-argument, con-argument, and decision. The frequency distribution of rationale elements among the messages containing rationale is shown in Table 3.2. The messages containing rationale in each project do not include enough training instances of different rationale elements. Thus, we decided to train and evaluate the fine-grained classifiers on mixed messages from all the three projects.

Table 7.4 summarizes the obtained classification results. The numbers in bold represent the corresponding top values. No single classifier works best for all rationale elements. However, for predicting the different rationale elements, we argue that recall is more important than precision. We aim at identifying as many rationale elements contained in the messages as possible, with the compromise of falsely predicting messages as containing additional rationale elements. Applying label powerset method achieved better recall for classifying all rationale

Table 7.5: Examples of fine-grained classification results of chat messages in co-located teams.

Chat message	Manual classification	Automatic classification
"@all please have a look at the current speed of the graph. I can't even scroll down fast enough to see it finish. Can we please make it 2 Secs total?[...]"	Issue, Alternative	Issue, Alternative
"And the back button can then be named Cancel"	Alternative	Alternative
"If there is the model number in green everyone should understand that the oven is connectedg"	Alternative, Pro-argument	Pro-argument
"There should be only one Navigation Controller and the Rest are Views."	Decision	Issue, Decision
"I just committed a new branch that added this.Ingredient then has a computed property "full-Price" that sums up the price for all occurrences of this ingredient"	Decision	Alternative

elements, except con-arguments. We achieved the highest recall for predicting alternatives (0.71). Pro-arguments followed with a recall of 0.55, issues with a recall of 0.47, and con-arguments with a recall of 0.44. The accuracy of predicting decisions was the lowest with a recall of 0.40.

Table 7.5 shows examples of fine-grained classification results when applying Binary Relevance and MNB. A possible interpretation of the obtained results is the sparseness of some rationale elements in the messages containing rationale (Table 3.2). Chat messages are informal short messages and the rationale elements discussed in these messages are unstructured and intertwined. Distinguishing between different elements is a nontrivial and intensive task even for a human judgment. Upon further inspection of the results, we found that a possible interpretation of the poor accuracy in classifying decisions compared to other elements, is that it is not always obvious in the messages whether a decision has been made. And in many cases, the decisions were classified as alternatives by the classifier. Furthermore, decisions have the lowest frequency distribution (10%) among the messages containing rationale; thus, fewer instances in the training set.

We replicated the above described experiments on the sentence level. In both binary and fine-grained classification, the classification on the message level performed significantly better than the classification on the sentence level, and thus, classification on the sentence level were not reported.

7.2.2 *Chat Messages of Distributed Teams*

In this section, we used the 7,500 manually annotated chat messages of the three distributed development teams from Chapter 4 as a truth set for the training and evaluation of A-REACT.

7.2.2.1 *Evaluation Settings*

CLASSIFICATION LEVEL Similar to the classification of chat messages of co-located teams, we performed the classification on the message level as it has been found to be more accurate than sentence level classification when categorizing rationale elements from short developers' messages (see Section 7.2.1).

CLASSIFICATION ALGORITHMS In addition to comparing the classification performance of MNB (Multinomial Naive Bayes) and SVM (Support Vector Machine) as in Section 7.2.1, we compared the classification performance of three more classification algorithms: DR (Decision Tree)¹¹, RF (Random Forests)¹², and LR (Logistic regression)¹³ as they have been widely used in text classification [44], [89], [131].

DATA BALANCING The problem of imbalanced training dataset is present in our dataset, in which only 25% of the messages contain rationale. We address this problem by applying a combination of SMOTE and under-sampling, as it proved to achieve better performance than applying under-sampling alone in Section 7.2.1.

¹¹ We used "weka.classifiers.trees.J48" implementation of DR.

¹² We used "weka.classifiers.trees.RandomForest" implementation of RF.

¹³ We used "weka.classifiers.functions.Logistic" implementation of LR.

Table 7.6: Binary classification results of chat messages of distributed teams. P is Precision, R is Recall, and F1 is F1-measure.

Configuration	Classifier	IRC message	Apache Lucene			Mozilla Thunderbird			Ubuntu			
			P	R	F1	P	R	F1	P	R	F1	
Imbalanced	MNB	With rationale	0.60	0.62	0.61	0.43	0.48	0.45	0.61	0.73	0.66	
		Without rationale	0.84	0.83	0.84	0.92	0.90	0.91	0.85	0.76	0.80	
	SVM	With rationale	0.63	0.45	0.53	0.54	0.31	0.39	0.64	0.54	0.59	
		Without rationale	0.80	0.89	0.84	0.90	0.96	0.93	0.78	0.85	0.81	
	DR	With rationale	0.52	0.39	0.44	0.33	0.09	0.15	0.60	0.50	0.55	
		Without rationale	0.77	0.86	0.81	0.87	0.97	0.92	0.76	0.83	0.79	
	RF	With rationale	0.67	0.21	0.32	0.62	0.02	0.05	0.72	0.46	0.56	
		Without rationale	0.75	0.96	0.84	0.87	1.00	0.93	0.77	0.91	0.83	
	LR	With rationale	0.46	0.44	0.45	0.27	0.38	0.31	0.46	0.47	0.47	
		Without rationale	0.77	0.78	0.78	0.90	0.84	0.87	0.73	0.72	0.72	
	SMOTE + Under-sampling	MNB	With rationale	0.82	0.76	0.79	0.90	0.60	0.72	0.83	0.59	0.69
			Without rationale	0.78	0.84	0.81	0.70	0.94	0.80	0.67	0.88	0.76
SVM		With rationale	0.83	0.76	0.79	0.85	0.76	0.80	0.85	0.78	0.81	
		Without rationale	0.78	0.85	0.81	0.78	0.86	0.82	0.79	0.86	0.82	

TRAINING AND EVALUATION Similar to the classification of chat messages of co-located teams (Section 7.2.1), we applied 10-fold cross validation for training and evaluating the classifiers. Additionally, we applied project cross validation to test the generalizability of the results. This corresponds to a 3-fold cross validation, as we are analyzing chat messages of three OSS projects.

We evaluated the classification performance using the standard metrics in machine learning: *precision*, *recall*, and *F1-Measure* [54].

TRANSFORMATION METHODS Similar to the classification of chat messages of co-located teams (Section 7.2.1), we compare the fine-grained classification

Table 7.7: Examples of binary classification results of chat messages of distributed teams.

IRC message	Manual classification	Automatic classification
"This bug was found because I reduced the maximum hit number randomly when collecting results"	With rationale	With rationale
"Never check email on waking up :)"	Without rationale	Without rationale
"I haven't looked at the patch, but it seems like we should also have a more general format or facet.format param"	With rationale	Without rationale
"I feel like windows live mail is dying"	Without rationale	With rationale

performance of two multi-label transformation methods: the BR (Binary Relevance) and the LP (Label Powerset) [187].

7.2.2.2 Evaluation Results

Table 7.6 shows the classification results of the different classifiers. Overall, all the classifiers have a better performance when classifying messages without rationale. This result is expected due to the sparseness of the messages containing rationale in the annotated sample (25%). For the messages with rationale, MNB is the classifier with the best balance between precision (ranging from 0.43 to 0.61) and recall (ranging from 0.48 to 0.73) and the highest F-measure of a range between 0.45 and 0.66 for all three projects.

It can be seen from the results that the classification performance of messages with rationale is correlated with the number of messages containing rationale in the analyzed messages (i.e., the training dataset). Hence, Mozilla Thunderbird has the lowest classification accuracy for messages with rationale as it contains the smallest percentage of messages with rationale (only 13%) among the three projects. Examples of the classification results of the MNB binary classifier are shown in Table 7.7. The second best classifier is SVM with better precision than MNB (ranging from 0.54 to 0.64) but lower recall (ranging from 0.31 to 0.54). Random forests has the highest precision ranging between 0.62 and 0.72, however,

Table 7.8: Project cross validation results of chat messages of distributed teams. P is Precision, R is Recall, and F1 is F1-measure.

Classifier	IRC message	Train: Mozilla Thunderbird, Ubuntu Test: Apache Lucene			Train: Apache Lucene, Ubuntu Test: Mozilla Thunderbird			Train: Apache Lucene, Mozilla Thunderbird Test: Ubuntu		
		P	R	F1	P	R	F1	P	R	F1
MNB	With rationale	0.49	0.56	0.52	0.24	0.64	0.35	0.51	0.51	0.51
	Without rationale	0.81	0.76	0.78	0.93	0.69	0.79	0.75	0.75	0.75
SVM	With rationale	0.61	0.28	0.39	0.26	0.50	0.34	0.65	0.30	0.41
	Without rationale	0.76	0.93	0.83	0.91	0.78	0.84	0.72	0.92	0.81

with a low recall ranging between 0.02 to 0.46. Considering the small percentage of messages containing rationale, a classifier with a low recall is not desirable.

The last part of Table 7.6 shows the increase in the classification performance of messages with rationale when training the two best performing classifiers (i.e., MNB and SVM) on the balanced dataset. SVM has a better performance than MNB for classifying messages with rationale in all three projects. However, the classification performance for messages without rationale decreased slightly as an expected result of under-sampling.

The results of applying project cross validation are shown in Table 7.8 (without balancing techniques applied). The overall performance of the classifiers decreased comparing to the results when the messages from the same project are used for training the classifier. Nevertheless, these results show that the generated classifiers can be applied with a reasonable accuracy across projects.

FINE-GRAINED CLASSIFICATION RESULTS We evaluated the performance of the fine-grained classifier on the 1,910 messages containing rationale among the 7,500 manually analyzed messages. The frequency distribution of rationale elements among the messages containing rationale is shown in Table 4.4. Similar to Section 7.2.1, we trained and evaluated the fine-grained classifiers on mixed messages from the three projects to include enough training instances of different rationale elements.

Table 7.9: Fine-grained classification results of chat messages of distributed teams. P is Precision, R is Recall, and F1 is F1-measure.

Rationale element	Binary Relevance						Label Powerset					
	MNB			SVM			MNB			SVM		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Issue	0.53	0.61	0.56	0.54	0.49	0.51	0.51	0.51	0.51	0.51	0.44	0.47
Alternative	0.53	0.56	0.55	0.54	0.46	0.50	0.52	0.57	0.55	0.50	0.58	0.54
Pro-argument	0.40	0.49	0.44	0.45	0.37	0.41	0.38	0.45	0.41	0.42	0.38	0.40
Con-argument	0.30	0.44	0.36	0.32	0.25	0.28	0.28	0.36	0.32	0.33	0.27	0.30
Decision	0.28	0.41	0.33	0.31	0.23	0.26	0.32	0.34	0.33	0.27	0.22	0.24

We applied the two learning algorithms that have the better performance in the binary classification: MNB and SVM. Table 7.9 provides an overview of the classification results of the different rationale elements. MNB has a better recall than SVM for all rationale elements except in classifying alternatives. On the other hand, SVM has a better precision in detecting all rationale elements except in classifying decisions. When comparing the F1-measure, applying Binary Relevance and MNB performs better for all rationale elements. Classifying decisions and con-arguments has the lowest accuracy among different rationale elements. This is understandable considering the sparseness of these elements in the messages with rationale (only 14% as reported in Table 4.4).

Table 7.10 provides examples of correctly and incorrectly classified messages. The examples were produced by applying Binary Relevance and MNB fine-grained classifier. We observed that the classifier tends to assign more rationale elements than what is present in the message, provided that MNB has better recall than precision for all elements.

Table 7.10: Examples of fine-grained classification results of chat messages of distributed teams.

IRC message	Manual classification	Automatic classification
“So I’ve seen 2 modes of failure. The first is getting more than 1 doc back for query-by-id (and we always use update, so it should be impossible)”	Issue	Issue
“I think it’s reasonable for ebox to say “I cannot configure this” if there is a non-default configuration file”	Alternative, Pro-argument	Alternative, Pro-argument
“I think it is possible with JFlex to make generated methods private”	Alternative	Alternative, Decision
“This would be solved in part by doing the separate scanner class like current StandardTokenizer”	Alternative	Decision, Con-argument

7.2.3 Comment in Issue Tracking Systems of Distributed Teams

In this section, we used the 3,007 manually annotated issue comments of the three distributed development teams from Chapter 4 as a truth set for the training and evaluation of A-REACT.

7.2.3.1 Evaluation Settings

CLASSIFICATION LEVEL We decided to perform the classification on the comment level. This decision is based on our finding during the manual analysis that sentences lack the argumentation context necessary for an accurate classification (see Chapter 4).

CLASSIFICATION ALGORITHMS In a similar fashion to Section 7.2.2, we compared the classification performance of five classification algorithms: MNB (Multinomial Naive Bayes), SVM (Support Vector Machine), DR (Decision Tree), RF (Random Forests), and LR (Logistic regression) as they have been widely used in text classification [33], [58], [170].

Table 7.11: Binary classification results of comments in issue tracking systems of distributed teams. P is Precision, R is Recall, and F₁ is F₁-measure.

Classifier	Comment	Apache Lucene			Mozilla Thunderbird			Ubuntu		
		P	R	F ₁	P	R	F ₁	P	R	F ₁
MNB	With rationale	0.90	0.99	0.94	0.75	0.94	0.83	0.82	0.94	0.88
	Without rationale	0.95	0.68	0.79	0.79	0.44	0.56	0.82	0.57	0.67
SVM	With rationale	0.91	0.96	0.94	0.84	0.84	0.84	0.88	0.90	0.89
	Without rationale	0.88	0.73	0.80	0.71	0.71	0.71	0.78	0.74	0.76
DR	With rationale	0.91	0.95	0.93	0.79	0.79	0.79	0.85	0.88	0.86
	Without rationale	0.84	0.72	0.78	0.62	0.62	0.62	0.73	0.67	0.70
RF	With rationale	0.91	0.99	0.95	0.80	0.92	0.85	0.83	0.97	0.89
	Without rationale	0.97	0.71	0.82	0.79	0.59	0.67	0.90	0.59	0.71
LR	With rationale	0.92	0.87	0.89	0.79	0.65	0.71	0.86	0.65	0.74
	Without rationale	0.67	0.78	0.72	0.52	0.69	0.59	0.51	0.77	0.62

TRAINING AND EVALUATION Similar to Section 7.2.1 and Section 7.2.2, we applied 10-fold cross validation for training and evaluating the classifiers. Additionally, we applied project cross validation to test if the classifiers generated in one project can be used for classifying rationale in another project. This corresponds to a 3-fold cross validation in which each fold corresponds to the comments of one of the three analyzed projects.

TRANSFORMATION METHODS we compare the fine-grained classification performance of two multi-label transformation methods: the BR (Binary Relevance) and the LP (Label Powerset) [187].

7.2.3.2 Evaluation Results

BINARY CLASSIFICATION RESULTS Table 7.11 gives an overview of the binary classification results per project. The numbers in bold represent the cor-

Table 7.12: Examples of binary classification results of comments in issue tracking systems of distributed teams.

Comment sentence	Manual classification	Automatic classification
“But this allow to create Span Disjunction Query, which is considered as a black sheep in Lucene herd. [...]”	With rationale	With rationale
“One possible downside to this change is that it changes a predictable branch (that is handled at the CPU level) into a method call... which if it’s not monomorphic can be un-inlined at the point of the call and thus end up slower (method call vs predictable branch). Will be interesting to see the benchmark results.”	With rationale	With rationale
“Please don’t assign bugs to yourself unless you are a developer who is going to fix it.”	Without rationale	Without rationale
“As an uncomfortable workaround, you can double-click on the area to the left of the twisty to expand/collapse subfolders.”	With rationale	Without rationale
“David Smiley Pinging you in case you want to have a chance to look into it before we release 6.1. FYI the seed still reproduces for me on master.”	Without rationale	With rationale

responding top values. Overall, RF outperformed other classifiers for classifying comments with rationale in both recall and F1-measure, except for Mozilla Thunderbird in which MNB achieved a better recall. It is important to realize that the classification performance of messages with rationale in each project is correlated with the number of these messages in the training set. Thus, Apache Lucene had a better classification accuracy (recall of 0.99 and F1-measure of 0,95) among the three projects, as it contains the highest percentage of messages with rationale (74%). Examples of the binary classification of comments when applying RF are shown in Table 7.12.

SVM outperformed other classifiers with the precision for classifying messages with rationale for Mozilla Thunderbird (0,84) and Ubuntu (0.88). However, LR achieved better precision for classifying messages with rationale in Apache Lucene (0.92).

Table 7.13: Project cross validation results of comments in issue tracking systems of distributed teams. P is Precision, R is Recall, and F₁ is F₁-measure.

Classifier	Comment	Train: Mozilla Thunderbird, Ubuntu Test: Apache Lucene			Train: Apache Lucene, Ubuntu Test: Mozilla Thunderbird			Train: Apache Lucene, Mozilla Thunderbird Test: Ubuntu		
		P	R	F ₁	P	R	F ₁	P	R	F ₁
MNB	With rationale	0.92	0.79	0.85	0.71	0.87	0.78	0.73	0.82	0.77
	Without rationale	0.57	0.80	0.67	0.61	0.36	0.46	0.48	0.36	0.41
SVM	With rationale	0.82	0.61	0.70	0.77	0.76	0.77	0.80	0.82	0.81
	Without rationale	0.36	0.63	0.46	0.58	0.59	0.58	0.60	0.57	0.58
RF	With rationale	0.87	0.72	0.79	0.73	0.95	0.83	0.78	0.87	0.82
	Without rationale	0.47	0.70	0.57	0.82	0.37	0.51	0.63	0.48	0.54

The truth set used for training the comment classifiers is characterized by a high incidence of comments with rationale, in contrast to the truth sets used in Section 7.2.1 and Section 7.2.2. Thus, balancing techniques were not needed.

The results of applying project cross validation are shown in Table 7.13. The overall performance of the classifiers decreased comparing to the results when the messages from the same project are used for training the classifier. However, no single classifier works best for all the three projects.

FINE-GRAINED CLASSIFICATION RESULTS We evaluated the performance of the fine-grained classifier on the 2,066 comments with rationale among the 3,007 manually analyzed issue comments. The frequency distribution of rationale elements among the comments containing rationale is shown in Table 4.8. We trained and evaluated the fine-grained classifiers on mixed comments from the three projects to include enough training instances of different rationale elements.

We compared the fine-grained classification performance of the three best performing classification algorithms in the binary classification: MNB, SVM, and RF. Table 7.14 gives an overview of the fine-grained classification results. The num-

Table 7.14: Fine-grained classification results of comments in issue tracking systems of distributed teams. P is Precision, R is Recall, and F1 is F1-measure.

Rationale element	Binary Relevance								
	MNB			SVM			RF		
	P	R	F1	P	R	F1	P	R	F1
Issue	0.68	0.84	0.75	0.74	0.72	0.73	0.74	0.79	0.76
Alternative	0.62	0.91	0.74	0.76	0.71	0.73	0.74	0.80	0.77
Pro-argument	0.44	0.88	0.59	0.64	0.57	0.61	0.85	0.47	0.60
Con-argument	0.33	0.87	0.48	0.49	0.46	0.47	0.79	0.20	0.32
Decision	0.85	0.53	0.65	0.67	0.63	0.65	0.99	0.53	0.69
Rationale element	Label Powerset								
	MNB			SVM			RF		
	P	R	F1	P	R	F1	P	R	F1
Issue	0.72	0.72	0.72	0.70	0.79	0.74	0.61	0.95	0.74
Alternative	0.67	0.84	0.75	0.77	0.75	0.76	0.88	0.28	0.43
Pro-argument	0.50	0.76	0.60	0.69	0.56	0.62	0.87	0.26	0.40
Con-argument	0.38	0.63	0.47	0.58	0.44	0.50	0.78	0.13	0.22
Decision	0.72	0.56	0.63	0.80	0.62	0.70	0.89	0.56	0.69

Table 7.15: Examples of fine-grained classification results of comments in issue tracking systems of distributed teams.

Comment	Manual classification	Automatic classification
"I think its confusing we have MatchAll but not MatchNone. I wonder if it should just be sugar and rewrite() to a boolean-query with no clauses? Then it wouldn't need a weight and scorer."	Alternative, Pro-argument, Con-argument	Alternative, Pro-argument, Con-argument
"There are other bugs that complain about the fact that a text string in a message body is not detected or not detected reliably when this string is inside a link."	Issue	Issue
"Patch removing context classloader usage. Tests seem to pass, unfortunately Solr trunk is very unstable. Some unrelated tests also fail on Jenkins, so I cannot be sure all is fine. This patch also adds context class loaders on the forbidden api list. Because of that I used the withContextClassLoader(ClassLoader, () -> ...) lambda method."	Issue, Alternative	Issue, Alternative, Con-argument
"But this allow to create Span Disjunction Query, which is considered as a black sheep in Lucene herd. I don't know why exactly, but have an idea."	Con-argument	Issue, Pro-argument, Con-argument

bers in bold represent the corresponding top values. Applying Binary Relevance achieved better recall for all rationale elements (ranging from 0.63 to 0.91), except for issues. Binary Relevance and MNB achieved better recall for alternatives (0.91), pro-arguments (0.88), and con-arguments (0.87); while Binary Relevance and SVM had a better recall for decisions (0.63). Label Powerset resulted in a better recall of 0.95 for classifying issues. When comparing F1-measure, Label Powerset and SVM tended to outperform other classifiers.

Table 7.15 gives examples of fine-grained classifications when applying BR and MNB. We noticed that the classifier tends to assign more rationale than the ones assigned to the comments during the manual analysis. This is expected as MNB achieved higher recall than precision when classifying rationale elements.

7.3 DISCUSSION

In Chapter 3 and Chapter 4, we present empirical evidence that developers' written communications are valuable sources of rationale during software development. However, previous studies have found that developers avoid exploring and searching communication repositories to understand the rationale behind design decisions [1], [105]. Hence, the rationale present in these communication artifacts remains implicit and not taken advantage of.

The manual analysis of communication artifacts to capture rationale is a tedious and labor-intensive process. Even with the introduction of a lightweight approaches integrated into these communication channels (see Chapter 6), developers still perceive the capturing activities as additional overhead and disturbance to the communication process. Our longterm research goal is to minimize this overhead by developing automated techniques to support developers in capturing and linking rationale across different development artifacts. And eventually making the captured rationale available to use during different maintenance and evolution tasks.

A-REACT, the automated rationale extraction method presented in this chapter can help in reducing the overhead involved in the manual capturing of rationale. With the aim of recovering as much rationale from written communications as possible, the binary classification results for detecting rationale are encouraging, with a recall up to 0.99 and a precision up to 0.92. For the fine-grained classification into different rationale elements, the classification performance varies according to the rationale element frequency in the communication artifacts used for training the classifier, i.e., the more instances of the rationale element in the training data, the better the performance.

From the evaluation results, it can be seen that the more rationale instances included in training the machine learning classifiers, the better the classification accuracy. Thus, since the issue comments truth set contains the highest percentage of comments with rationale (69%) among the three analyzed communication artifacts, the binary and fine-grained classifiers achieved better accuracy in classifying comments than classifying chat messages of co-located and distributed development teams.

The results of the project cross validation of the generated classifiers suggest that it is advisable to use communication artifacts from the same project to train the classifiers, rather than using generic rationale classifiers. This is due to the fact that development teams use different terminologies and jargons in their communications; thus, the generated classification features might be different across projects. Our findings confirm the doubts of earlier studies that there might be tradeoffs between a broader applicability of the classifiers and the classification accuracy [160].

Even though a complete automated approach for accurately extracting well-structured rationale is still unrealized, our results of the automatic detection and extraction of rationale from developers' text-based communications form a stepping stone in this direction. It is important to realize that we view A-REACT as a complementary technique to the explicit documentation of rationale, rather than a replacement. For example, the automated rationale extraction from communication artifacts could be followed by an activity to validate and refine the structure of the extracted rationale into a more manageable format. Automated techniques and manual capturing could be integrated to support a systematic capturing of rationale throughout the software lifecycle. As argued by Hassan that automated techniques "should not aim for full automation instead they should aim to create a synergy between practitioners and MSR techniques. Surprisingly full automation is not always the most desired option for practitioners." [71].

7.4 THREATS TO VALIDITY

In this section, we discuss the potential threats to the validity of our results according to the four validity aspects as defined by Runeson et al. [164].

CONSTRUCT VALIDITY is concerned with how accurately the study observations interpret and measure the theoretical constructs [164]. In A-REACT, the accuracy of the fine-grained classifier is affected by the accuracy of the binary classifier. However, to evaluate the feasibility of A-REACT, we evaluated the classification performance of the binary and fine-grained classifiers independently

of each other. Thus, we refrain from claiming the completeness of our classification results.

INTERNAL VALIDITY is concerned with the confounding factors that may influence the study results [164]. The creation process of the truth sets used in the method evaluation is a subjective process. To mitigate this risk, coding guides with descriptions of the coding tasks and clear definitions and examples of the rationale elements were created before the annotation process of each of the three truth sets. The coding guides were developed in two annotation trials for each truth set (see Chapter 3 and Chapter 4). Moreover, each communication artifact, i.e., a chat message or a comment, was annotated independently by two coders and the disagreements between the two coders were resolved through discussions.

EXTERNAL VALIDITY is concerned with the generalizability of our results [164]. A-REACT was evaluated on three written communication artifacts: chat messages of co-located teams, chat messages of distributed teams, and comments in the issue tracking systems of distributed teams. These communication artifacts were generated during different projects and in different development settings. However, we cannot claim that our results are representative of all software projects.

RELIABILITY is concerned with to what extent the study results are dependent on a specific researcher, i.e., whether the study yields the same results if replicated by other researchers [164]. Although the truth set of the chat messages of co-located teams could not be made available due to privacy issues, the truth set of the chat messages of distributed teams¹⁴ and the truth set of the comments in issue tracking systems of distributed teams¹⁵ are publicly available to other researchers. Furthermore, the coding guides used during the annotation process are made available to other researchers (see Appendix A, Appendix B, and Appendix C).

¹⁴ <https://figshare.com/s/20f10511dc6e36c98ccd>

¹⁵ <https://figshare.com/s/62da0027efcoe7ae35a0>

Related Work Relevant to Rationale Annotation and Automated Capturing Approaches

“We should not be dissuaded from our duty by the existence of textual narrative in these early artifacts.”

—Dekhtyar et al. [41]

In Chapter 6 and Chapter 7, we presented two methods for capturing rationale in text-based developers’ communications, REACT and A-REACT. In this chapter, we discuss related work in two areas: rationale annotation approaches and automated extraction of rationale.

The chapter is structured as follows: Section 8.1 presents existing rationale annotation approaches in development artifacts. Section 8.2 discusses work related to the automated extraction of rationale.

8.1 RATIONALE ANNOTATION APPROACHES

To the best of our knowledge, no previous work has investigated the annotation of developers’ chat messages to capture rationale during software development. However, there is a relatively small body of literature that has proposed approaches for annotating rationale in textual documents generated during software development.

Kato et al. [91] proposed IDIMS (Integrated Design Information Management System), an approach to capture rationale from developers’ email communications. In the approach, developers annotate their emails to indicate *design issues* and *design decisions*. The annotated issues and decisions can be linked to form a directed graph. Then, the annotated emails are processed to collect annotations and stores them in Issue/Decision Repository. However, their approach requires

the emails to be first converted into XML format to add the annotations, which may limit the usefulness of the approach in practice. Liu et al. [118] introduced an approach for the manual annotation of design rationale in archived design documents, such as patent documents. During the manual annotation, designers highlight the segments of documents relevant to rationale according to an issue, solution, and artifact layer (ISAL) representation model of rationale. Afterwards, the tagged segments are automatically extracted and stored in a design rationale repository. Furthermore, designers can link relevant rationale elements to form a design rationale network. Aman-ul-haq and Babar [3] developed a tool that allows an architect to annotate information related to architectural knowledge in emails and other textual documents using a set of pre-defined tags such as *rationale*, *architecture decisions*, and *design options*. The annotation tool is integrated to the email clients and document editors used during development to minimize the capturing efforts on the architects. The annotated information is then extracted from these documents and stored in a knowledge repository. Similar to our work, these studies aim at capturing rationale from the available communication and documentation records rather than requiring developers to construct a structured rationale. Thus, lowering the capturing costs on developers. However, they differ from the work presented in this dissertation by the type of documents they annotate, and the rationale representation models on which they based their annotations.

Other researchers have proposed approaches in which descriptions of rationale are attached as annotations to development artifacts. Shipman and McCall [174], [175] proposed Hyper-Object Substrate (HOS), an approach for capturing rationale by attaching informal design communications, such as emails, as textual annotations to the artifact design. These captured communications can be converted over time into a more formal representation of rationale. Reeves and Shipman [150] introduced a prototype system, XNETWORK, that implements two main ideas for reducing the efforts required of designers to provide rationale. First, rationale capturing should be part of the design process. Second, capturing rationale in the form of free text. The system allows the integration of design artifacts with designers' communications in the form of textual annotations. The authors argue that adding designers' communications in the form of text annotations reduces the cognitive overhead in providing rationale on de-

signers, and that these recorded communications will serve as the best source for design rationale. Bruehlmann et al. [19] proposed a generic approach to capture human knowledge in the form of annotations during the reverse engineering process. They implemented the approach in a tool, called Metanool, in which annotations can be iteratively defined, refined, and transformed, without requiring a fixed meta-model to be defined in advance. The work presented in this dissertation is complementary, the rationale extracted from developers' text-based communications either by applying REACT annotations or A-REACT for the automated extraction can be linked as textual descriptions to other development artifacts as proposed in these studies.

A few other annotation approaches have been developed to capture rationale within source code and to link existing rationale knowledge with code. Hesse et al. [76] developed an annotation approach for capturing rationale of implementation decisions using inline code comments in the source code. In their approach, text-based annotations are written directly into the code files, which enables developers to document their implementation decisions without the need to switch tools. The rationale annotations are derived from the decision documentation model [78], one annotation for each decision knowledge element¹. The annotations can be used to create a new decision knowledge element or to link to an existing one. The annotations are integrated into a knowledge management tool which allows the linkage of annotations to external knowledge within the tool, e.g., requirements specifications or design diagrams. Lougher and Rodden [120] proposed a system for the documentation of maintenance rationale through annotation. Rationale is captured in the form of comment documents that can be linked to different parts of the source code using hypertext links. The comments can be of three types: free text, graphics, and structured textual forms. However, their work is focused on capturing maintenance rather than design rationale. The authors argued that maintenance rationale is less contentious and more focused on describing the nature of the artifact; while design rationale is more focused on capturing the argumentative process of constructing the artifact.

The lightweight annotation approach presented in this dissertation, REACT, differs from these approaches in that it focuses on capturing the dialectical

¹ The decision documentation model is described in Section 2.2.2.1 in Chapter 2.

reasoning and discussions between developers in the informal communication channels. In addition, developers are not required to write further descriptions of rationale when applying REACT rationale annotations to their chat messages. However, these approaches can be brought together to document the distributed rationale knowledge during software development. An example of such effort is presented in the work of Kleebaum et al. [93]. The authors suggested techniques that trigger developers to capture and use decision knowledge in continuous software engineering. For example, when developers commit code, they are triggered to make the tacit decision knowledge explicit by annotating rationale. This can be done in source code by applying the annotations proposed by Hesse et al. [76] or in chat messages by applying REACT annotations presented in this dissertation. Furthermore, annotations could be applied to other development artifacts such as pull requests and wiki pages. The authors also proposed that explicit annotation of rationale can be combined with machine learning techniques to mine unstructured distributed decision knowledge.

Although only few studies have investigated the annotation of textual documents to capture rationale during software development, the concept of annotating resources using tags is not new to software development. A number of studies have explored the role of tagging as a tool for organizing and sharing knowledge during software development. Treude and Storey [186] conducted two empirical studies to investigate the role of work item, i.e., development task, tagging in software development. Their results showed that different kinds of tags have emerged over the duration of a software project, including architecture, planning, and documentation related tags. Furthermore, they found that the lightweight nature of tagging was one of the main advantages that encourages its adoption by developers. Storey et al. [178], [179] developed TagSEA (Tags for Software Engineering Activities), a source code annotation tool to enhance navigation, coordination, and capture of knowledge relevant to a software development team. Their tool combines waypoints from geographical navigation with social tagging. In TagSEA, a developer can create a waypoint by associating tags with parts of the source code. This is done by typing “@tag” in a comment block, followed by the tag keyword and some descriptive text. The created waypoints can be used to document and share important knowledge about the source code. However, a major difference of our approach to tagging is that

REACT annotations are derived from an argumentative representation model of rationale; while tags usually refer to a freely chosen keywords without requiring a fixed meta-model to be defined in advance [186].

8.2 AUTOMATED EXTRACTION OF RATIONALE

The automated extraction of rationale has received an increasing attention from researchers in recent years. According to Liang et al. [116], the automated extraction of rationale is promising in addressing a number of challenging issues in current design rationale research. First, many design rationale systems require heavy human involvement and designers cannot often afford time and efforts in documenting the rationale behind their decisions. Second, with the increasing number of evolving design documents, the manual handling of rationale included in these documents in a timely manner becomes infeasible. Consequently, large amount of design documentation is often neglected and left intact. Therefore, there is a pressing demand for automated solutions to support the capturing and extraction of rationale from different artifacts generated during software development.

In a recent work, Robillard et al. [156] advocated for a new vision of an on-demand developer documentation (OD₃), which promotes using development artifacts for the automated generation of developer documentation. The authors discussed its opportunities and challenges and viewed it as a promising research direction for software documentation. Sharing the same vision, several researchers have investigated the exploitation of available development artifacts for the automated extraction of rationale. Liang et al. [116] proposed an approach to automatically extract design rationale information from archived design documents. They captured rationale according to ISAL model [118], a three layers rationale representation model consisting of issues, design solutions, and artifacts layers. The approach applies text mining and machine learning techniques for the identification of artifact information, issue summarization, and discovery of solution and reasons pairs from patent documents.

Similar to A-REACT, Rogers et al. [160], [162] applied supervised machine learning techniques to extract rationale from bug reports. In their work, they investigated the use of ontology and linguistic features for training machine

learning classifiers that classify sentences containing rationale into decisions, alternatives, and argumentation. In their recent work, Rogers et al. [161] proposed a system that uses genetic algorithms to evaluate candidate feature sets for identifying rationale in two types of documents: bug reports and design sessions transcripts. Our work differs in that we focus on extracting rationale from text-based developers' communications. This poses different challenges—as developers' communication artifacts are short, informal, and less structured than the previously analyzed documents.

Closely related to our work is the Rationale Extractor (REx) approach proposed by Lucia et al. [39]. REx applies information retrieval, natural language processing, and supervised machine learning techniques to automatically extract design rationale from unstructured communications, particularly email repositories. Given a new email, the email is split into sentences and the designer is presented with a likelihood value for each rationale element, i.e., the probability that the sentence contains that element. The designer can then select the correct rationale elements contained in the email or classify it as non-rationale. Finally, the extracted rationale elements are stored in an issue-base and cross-referenced to the originating documents, i.e., emails. When a new email is processed, relevant emails and their contained rationale elements are presented to provide the designer with the discussion context. Similar to our work, they use the rationale schema proposed by Bruegge and Dutoit [16], which is based on IBIS [102], to classify rationale elements. However, their approach differs from ours in that it still requires the designers intervention to select the correct rational elements from the proposed ones.

Another stream of research has focused mainly on the automatic recovery of architectural design decisions and their rationale. Bhat et al. [11] proposed a two-phase machine learning based approach for detecting architectural design decisions in issue tracking systems and classifying them into structural, behavioral, and ban decisions. Shahbazian et al. [171] presented a technique, named RecovAr, for automatically recovering architectural design decisions from issue tracking systems and code repositories. RecovAr first identifies how the architecture of the system has changed, then maps the code commits to issues to detect decisions affecting the system architecture and their rationale. The technique is based on the assumption that the issues related to the changed architectural

entities contain the rationale behind the change decisions. López et al. [119] proposed an ontology-driven approach to extract knowledge units relevant to architecture rationale from plain-text documents. The recovered rationale need to be validated by a software architect before it is stored in a software architecture and rationale knowledge base to enable further manipulation. They argued that rationale recovery from existing documents can be decomposed into three smaller problems: automatic extraction of rationale from these documents, formalization of the extracted rationale, and manipulation of the formalized information for further reuse.

Baysal and Malton [9] described an approach to find correlation between discussions in email archives and source code changes. The approach uses natural language processing techniques to compare the vocabulary of the changed code in a release history—i.e., identifiers of classes, methods, and comments—with the vocabulary of the email discussions preceding that release to find discussions relevant to the code changes. Brunet et al. [20] applied supervised machine learning techniques for the automatic identification of structural design discussions in issues, commits, and pull requests. Li and Ramani [115] proposed ontology-based design document analysis and retrieval tool (ODART), an approach that combines natural language processing techniques and domain-specific ontology for the extraction and retrieval of design information from unstructured, textual design documents, such as technical reports, proposals, and drawing notes.

Williams and Rainer [193] investigated the automatic identification of arguments in software practitioners blog posts by identifying specific words in the blog post text. They reported a preliminary evaluation of the approach on posts from one software practitioner's blog. Sorbo et al. [177] proposed DECA (Development Emails Content Analyzer), an approach that applies natural language parsing to classify the content of development emails according to their purpose. The approach classifies email sentences into six different categories: feature request, opinion asking, problem discovery, solution proposal, information seeking, and information giving. Pascarella and Bacchelli [143] proposed a taxonomy for classifying code comments among which is implementation rationale. The authors applied machine learning algorithms for the automatic classification of code comments into the proposed taxonomy.

Taken together, these approaches are related to our work in that they use automated techniques to extract design related knowledge from developments artifacts, but differ widely in the applied techniques, the type of artifacts they analyze, and the type of knowledge they recover.

A number of researchers have attempted to lessen the documentation burden of rationale by automatically generating rationale through monitoring the design process. Myers et al. [134] designed Rationale Construction Framework (RCF) to automatically construct rationale information for the detailed design process. RCF records designers interactions in a CAD (computer-aided design) tool during detailed design and produces a rich design history. This design history is used to provide a series of hierarchical abstractions about *what* the designer did and *when*. In addition, the rationale that explains *why* the designer performed a particular action is extracted according to a set of design metaphors. Garcia and Howard [61] implemented Augmenting Design Documentation (ADD), an intelligent apprentice to the designer. ADD learns about the features that make a design action different from the standard ones, and whenever a designer makes an action that differs from the ADD's expectations, it asks the designer for a justification. Later, queries about rationale is constructed using a combination of the system's domain knowledge and the justifications supplied by the designers. ADD is an example of the generative design rationale approach proposed by Gruber and Russel [64]. The authors argued for a paradigm in which rationale is generated, in response to information request, from background knowledge and information captured earlier during the design process. They affirmed that it is more important to capture the data that can be used to infer answers to question at later stages, rather than trying to anticipate possible questions and formulate answers in advance. Although the generative approach allows for more computational power, it requires the capturing of more operational information. A system proposed by Sung et al. [182] and Rea et al. [148] automatically logs the designer's actions while using a CAD system, extracts the design rationale by automatically parsing these logs, and presents the extracted rationale in an understandable format. Our work differs in that we extract rationale automatically from free-form text communications to capture the deliberation and discussions related to rationale.

Few studies have investigated the automatic detection of decisions and their rationale from the transcripts of audio or video recordings of meetings. McCall and Mistrik [127] proposed an approach that applies natural language processing techniques for identifying requirements and their rationale from transcripts of participatory design sessions. Their overall goal was to reduce the number of utterances a human analyst needs to look at for identifying requirements-related rationale. The authors argued that requirements are typically found in the users' responses to proposed features of a system, i.e., in the arguments to the proposed features. Therefore, the technique uses semantic grammars to detect *proposals* and *arguments* in reaction to these proposals based on a set of introductory wording, e.g., proposals may be preceded by "What if we...", while arguments may be preceded by "I like the idea. It will...". Similar to this dissertation, their proposed techniques structure the captured rationale in a form closely related to IBIS.

Hsueh and Moore [82] developed supervised machine learning classification models for the automatic detection of decisions in meeting recordings. The classification models detect decisions on two levels of granularity: detecting decision-making dialogue acts (DM DAs) and detecting decision-making topic segments (DM Segments), i.e., topic segments that contain one or more decision-making dialogue acts. They evaluated their classification models on the extractive summaries of a subset of the AMI meeting corpus [130]. In their follow-up work [83], Hsueh and Moore applied the developed models in developing AMI DecisionDetector, a system that performs automatic decision detection in meeting speech. In the same manner, AMI DecisionDetector detects decisions on two levels of granularity: dialogue acts and topic segments. First, the system detects topic segments in meeting speech in which decisions have been made. Second, the decision-related dialogue acts that are reflective of the decision discussion are detected. Furthermore, the system provides visual aids for reviewing decisions made during the meeting. They evaluated the classification performance of the system when detecting decisions directly from the complete meeting transcripts rather than their extractive summaries. Fernández et al. [55] proposed a hierarchical approach for automatically detecting decisions sub-dialogues, i.e., regions of dialogue where decisions were made, in transcripts of multi-party meetings. The hierarchical approach involves two steps: *sub-classifiers* that first classify ut-

terances into three decision dialogue act (DDA) classes: issue, resolution, and agreement; then *super-classifiers* that detect decision sub-dialogues. Detecting the decision-making topic segments or sub-dialogues provide important contextual information about the discussion topics where decisions have been made.

Rodeghero et al. [158] presented an approach for automatically extracting information relevant to user stories from recorded conversations between developers and customers. The approach consists of machine learning classifiers that were trained to recognize turns in speech containing function and rationale information behind user stories. The approaches proposed in these studies are not practical for real-world development as they require accurate transcripts of the analyzed meetings. However, an automatic generation of these transcripts using speech recognition is still not completely feasible [127]. Nevertheless, these approaches could be complemented with the automated approach presented in this dissertation. This provides automated rationale extraction from both face-to-face and text-based communication channels, which are used as a complementary communication and knowledge sharing channels during software development [181], [196].

A different perspective on studying rationale is presented in the work of Kurtanović and Maalej [103], [104] in which the authors studied how users denote rationale in online reviews. They applied grounded theory approach to identify rationale concepts in user reviews. The identified concepts consist of issue, alternative, criteria, decision, and justification. Furthermore, the authors investigated various supervised machine learning algorithms and feature combinations to automatically detect these rationale concepts in user reviews. However, in this dissertation, we focus on studying rationale from the developers' perspective.

Overall, our work adds to the growing body of research on the automated extraction of rationale from development artifacts in general, and provides a deeper insight for future research into the automated extraction of rationale from text-based developers' communications.

Part IV

Conclusion

Conclusion and Future Work

“A software organization’s main asset is its intellectual capital [...] The major problem with intellectual capital is that it has legs and walks home every day.”

—Rus and Lindvall [165]

Capturing rationale has long been recognized as a central problem during software development. However, developers often resist capturing rationale due to the intrusiveness and labor-intensive activities of capturing approaches. Many rationale capturing approaches require developers to manually write up their rationale according to a pre-defined template. Furthermore, many of these approaches are not fully integrated into software development activities, i.e., they require developers to change tools to document their rationale. As a consequence, these approaches represent additional overhead on developers and interruption to the development activities; thus, increasing their adoption barriers by developers.

The overarching goal of this dissertation was to advance the rationale capturing during software development by studying how developers discuss rationale over text-based communication channels and by developing methods integrated into these channels to help developers in capturing the discussed rationale. Studying written communications of developers as a source of rationale was motivated by the fact that decisions made during software development result from the deliberation of issues and discussion of the pros and cons of different alternatives among developers. As Shipman and McCall stated: “designers seem to spontaneously produce relatively well-structured arguments in natural discussion” [174], which makes developers’ communications a rich source of information about the software system and the development process.

We followed a representation of rationale adapted from Kunz and Rittel's IBIS issue model [102], that captures issues, alternatives, pro-arguments, con-arguments, and decisions as natural language text written by developers while communicating with each other. This representation provided the basis for analyzing how developers discuss rationale in text-based communication channels.

Our long-term vision is to minimize the additional overhead of the manual capturing of rationale during software development by developing automated techniques to support developers in the capturing and linking of rationale across different development artifacts. And eventually making the captured rationale available to use during different maintenance and evolution tasks.

In the following, we summarize the contributions of this dissertation in Section 9.1, and outline future work direction in Section 9.2.

9.1 CONTRIBUTIONS

In this dissertation, we made the following contributions:

1. *Analyzing Rationale in Written Developers' Communications*

We presented three empirical studies to understand how developers discuss rationale over two text-based communication channels: chat messages and issue tracking systems, in co-located as well as distributed development teams. Our focus on these two channels was motivated by their increasing popularity and significance during software development [4], [90].

The first empirical study aimed at understanding how software developers discuss rationale in chat messages of co-located development teams. In particular, we investigated the frequency and completeness of rationale present in chat messages of three co-located teams by applying content analysis techniques [138]. Despite the dominance of face-to-face communications and the absence of geographical and time zone differences in co-located teams, we found that 9% of their chat messages contained rationale. Due to the high volume of chat messages and the sparsity of the messages containing rationale, the manual extraction and classification of rationale is a tedious and time-consuming process. Our findings emphasize the importance of linking rationale elements found in chat

messages with the rationale elements extracted from other development and communication artifacts for a more complete capture of rationale.

In the second empirical study, we investigated how developers discuss rationale in chat messages across distributed development teams. We collected IRC logs from three OSS projects developed and maintained by globally distributed teams. We manually analyzed a sample of 7,500 IRC messages by applying content analysis techniques [138], and found that an average of 25% of the analyzed messages contained rationale. Moreover, we found that IRC authors who were committing to the project code repository contributed an average of 54% of the messages containing rationale. However, we did not find a strong correlation between the development activities and the rationale contribution.

We performed the third empirical study to analyze how distributed developers discuss rationale in the comments of issue tracking systems. We collected issues and their comments of the three OSS projects and applied content analysis techniques [138] on a stratified sample of 3,007 comments from 300 issues. We found that an average of 69% of the analyzed comments contain rationale. However, only 27% of the rationale contributors were identified as committing to the software code repository and they contributed an average of 38% of the comments identified as containing rationale among the three projects.

The investigation of how developers discuss rationale in these two text-based communication channels has shown that: (i) developers discuss related rationale elements in a sequence of short messages and comments, (ii) rationale is fragmented across different communication and development artifacts, and (iii) the manual extraction and classification of rationale from written communications is a tedious and time-consuming process due to the high volume of communication artifacts.

Overall, the findings of these studies provide empirical evidence that text-based developers' communications are valuable sources of rationale during software development. Furthermore, they give deeper insights about the nature of rationale found in written communications that can aid future research in exploiting these communications as a source of

rationale and for building effective rationale capturing tools from these communication channels.

2. *REACT: A Method for Capturing Rationale in Developers' Chat Messages*

We presented REACT (Rationale ExtrAction from Communication arTifacts), a novel lightweight method to capture rationale in developers' chat messages. REACT is designed to be easily integrated into developers' messaging platforms, and it can be used by developers to (i) individually annotate their own messages containing rationale, and (ii) collaboratively annotate messages posted by other team members. We evaluated REACT in two studies: a short-term design task and a medium-term project. Our evaluation shows that REACT is easy to learn, simple to apply, and applicable to capture rationale in chat messages.

However, it is worth noting that the effectiveness of REACT is highly dependent on providing immediate benefit for developers to justify the efforts of capturing rationale. While REACT annotations are simple to apply, the need to think about the message contents with respect to rationale presents additional cognitive overload and disruption to the communication flow for developers.

Contrary to expectations, our results showed that privacy is not the main concern for developers when capturing rationale in their chat messages. Communication distraction resulting from applying these annotations and the lack of immediate benefits motivating the rationale capturing are more pressing concerns experienced by most developers in our evaluation.

3. *A-REACT: An Automated Rationale Extraction Method*

To reduce the cognitive overload of categorizing rationale elements and to deal with communication noise, we developed A-REACT (Automated Rationale ExtrAction from Communication arTifacts), an automated method that applies supervised machine learning techniques for classifying rationale in developers' written communications on two granularity levels: binary and fine-grained classification. The binary classifier detects communication artifacts containing rationale and filters out communication artifacts without rationale, and the fine-grained classifier subsequently classifies

the communication artifacts containing rationale into different rationale elements: issues, alternatives, pro-arguments, con-arguments, and decisions.

We evaluated the feasibility of A-REACT on three communication artifacts: chat messages of co-located teams, chat messages of distributed teams, and comments in issue tracking systems of distributed teams. Our evaluation revealed that A-REACT can detect rationale with a recall up to 0.99 and a precision up to 0.92, and classifies the detected rationale into different rationale elements with a recall up to 0.99 and a precision up to 0.95. The fine-grained classification performance varies according to the rationale element frequency in the communication artifacts used for training the classifier.

The results of the project cross validation of the generated classifiers suggest that it is advisable to use communication artifacts from the same project to train the classifiers, rather than using generic rationale classifiers. This is due to the fact that development teams use different terminologies and jargons in their communications; thus, the generated classification features might be different across projects. However, the truth sets used for building and evaluating the classification models of A-REACT can be used by other researchers to replicate our results¹.

The results of our experiments can provide guidelines for researchers about the techniques and configurations that yield the most accurate results.

Even though a complete automated approach for accurately extracting well-structured rationale is still unrealized, our results of the automatic detection and extraction of rationale from developers' written communications will pave the way to bring this vision closer to reality.

9.2 FUTURE WORK

Studying rationale in text-based communications of developers opens up new opportunities for incorporating and taking advantage of informal communication channels as valuable sources of rationale during software development.

¹ Except for the truth set of the chat messages of co-located teams which could not be made available due to privacy issues.

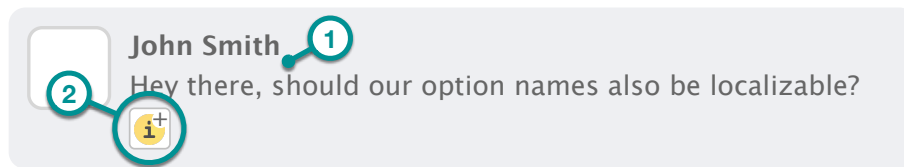


Figure 9.1: Extension mockup: based on detected rationale elements in the message (1), rationale annotations are suggested (2).

However, this comes with a number of challenges due to the informal nature of these communications. These communications are written in natural language, lack a clear structure, and contain a large volume of noise and irrelevant information. Furthermore, these communications contain mixed content, e.g., they may contain code snippets or stack traces. The research presented in this dissertation opens up several directions for future work, which are outlined below.

IMPROVEMENTS ON REACT Our evaluation has shown that the effectiveness of REACT is highly dependent on providing immediate benefit for developers to justify the efforts of capturing rationale. Future research therefore should explore which immediate benefits can be offered to motivate developers to annotate rationale in their communication artifacts. For example, different techniques can be explored to export conversations annotated as containing rationale in the chat messages to issue trackers and to other development artifacts that were attached to the chat messages, such as design models or code snippets.

To reduce the cognitive load of categorizing messages on developers, a hybrid approach could be implemented that employs machine learning for the automatic detection of messages containing rationale. The involvement of developers and their domain knowledge guarantees accurate categorization of rationale even in situations in which machine learning might fail. A machine learning component could prompt developers—the “domain experts”—to approve the suggested categorization of the chat message if rationale is detected. Utilizing a chat bot might be a first step towards such an implementation. Figure 9.1 illustrates the idea.

REACT could be further improved by making use of message threads in chat channels, a feature that has been recently introduced by Slack². This feature helps in keeping the discussion focused by grouping messages and relating replies that discuss the same topic into sub-conversations. Further research could make use of this feature by creating a new thread when a message is annotated using REACT as containing an issue. In this way, the replying messages discussing different alternatives to address the issue are grouped into a single conversation. Finally, message threads could be extracted to capture the discussed rationale rather than single messages to preserve the context of the conversations.

Another direction for future work is studying how REACT annotations support effective team communication, e.g., promptness of response to an annotated message from other developers.

IMPROVEMENT ON A-REACT Future work could investigate different features and feature combinations for improving the classification performance. For example, adding classification features on whether the neighbor messages or comments contain rationale. We hypothesize that this will improve the classification accuracy as developers tend to discuss rationale in a sequence of consecutive messages. Furthermore, considering additional features of the communication artifacts such as linguistic features and metadata, e.g., the message or comment authors and the creation time, could improve the classification performance.

Further research could also be conducted to experiment with additional machine learning algorithms and different optimization parameters of these learning algorithms.

A more balanced training set with a larger number of manually identified rationale elements would help in addressing the sparsity of rationale in text-based communications of developers. Another technique to address the rationale sparsity is by applying data augmentation approaches for text classification [163].

² <https://slackhq.com/threaded-messaging-comes-to-slack-417ffb054bd>

One drawback for applying supervised machine learning algorithms is the need for annotated training dataset—which is usually created manually, requiring a great effort. The creation of such a truth set could dissuade practitioners from using this type of techniques. One interesting avenue for future work is running REACT for some time in a project, and then use the chat messages annotated by the developers as a truth set for training supervised machine learning classifiers, as in A-REACT, that can be applied for the automated extraction of rationale.

A-REACT could be extended to allow the manual validation and refinement of the automatically extracted rationale. However, further research should be carried out to compare the effectiveness of the semi-automatic capturing approaches, i.e., an automatic extraction of rationale followed by a manual validation, against the complete manual capturing of rationale.

LINKAGE OF RATIONALE Developers use various communication channels to share knowledge and coordinate their development activities. In addition, various artifacts and documents are generated during software development. As a result, the rationale is fragmented across multiple sources and no single source can provide a complete picture of the rationale. Future research is required to explore additional sources of rationale, and to develop methods and tools to systematically extract and aggregate rationale from its identified sources.

Classifying single communication artifacts, e.g., chat messages, containing rationale is not enough for extracting useful rationale information, as these artifacts are highly context dependent. The linkage of the extracted rationale need to be performed at two levels. First, the communication artifacts identified as containing rationale should be linked with other artifacts discussing related rationale elements within the single channel. However, it is important to note that communication artifacts discussing related rationale elements may not be exchanged in a sequential order, as discussion topics in these informal communication channels are often intertwined and may also be discussed at different times. Second, communication artifacts discussing related rationale elements across multiple communication

channels should be linked. For example, the same issue might be discussed in the chat messages as well as in the comments of issue tracking systems.

Another compelling direction for future work is the linkage of the extracted rationale with the parts of the source code that it describes. Establishing traceability links between the source code and these discussions allows to join decisions and their rationale with their implementation [7]. A low-hanging fruit for such linkage is capturing rationale from commit messages. Commit messages may include important information about the committed changes and their rationale. However, as discussed by Kleebaum [93], to capture useful rationale from the commit messages, it is important that developers commit *atomic* changes that address one single issue instead of *composite* changes that address multiple development issues in a single commit [184].

Another future work direction for such linkage between communication artifacts containing rationale and the source code is through exploiting the code fragments included in these communications.

Part V

Appendices

Coding Guide: For Annotating Rationale Elements in Developers' Chat Messages of Co-located Teams

Our study aims at studying what type of rationale elements are present in developers' chat messages.

As part of the coding task, you will read chat messages of three development teams that were part of a multi-project course with industrial partners at the Technical University of Munich in 2015 and 2016. During the course, students were asked to develop a mobile application. Your task is to classify the parts of the chat messages that contain rationale elements.

This guide describes the instructions, which you should follow carefully in order to successfully conduct the task. You will use GATE¹ for completing your task. We recommend using this guide as a reference during the coding task.

Your task is to read the chat messages assigned to you. For each message:

- If the message contains rationale, identify sentences of the message that contain rationale elements by highlighting them and classifying them accordingly. We explain each rationale element later in this guide.

Please make sure that you read about the mobile application (from which you are coding the messages) in order to make sure that you understand its main functionality. You can read a description of each mobile application on the iPraktikum² results website.

In case any questions arise during the coding, please contact the other project members.

CLASSIFYING RATIONALE IN THE CHAT MESSAGE: Each chat message consists of one or more sentences. If the message contains a rationale element, high-

¹ <https://gate.ac.uk>

² https://www1.in.tum.de/lehrstuhl_1/component/content/article/733

light only the sentence(s) of the message containing the rationale element and indicate the type of the rationale element. The highlighted part can be one sentence, multiple sentences in the message or the complete message.

The smaller unit you can classify, i.e. highlight and assign a rationale element to, is a sentence. The maximum is the complete message. This means if only part of a sentence contains rationale, you should classify the complete sentence. A sentence ends with one of four punctuation marks: a period (i.e. "."), an exclamation mark (i.e. "!"), a question mark (i.e. "?") or a new line (i.e. "/n"). Given the informal nature of chat messages, some messages consist of sentences without a proper ending punctuation. In that case, the ending of the message is considered as the end mark of the sentence.

If a rationale element spans multiple messages, you should classify each message separately. Moreover, if two or more sentences of a message are related to one type of rationale and some of these sentences are related to another type of rationale, then label the complete set of sentences as both types of rationale to avoid substring labeling.

You can assign more than one rationale element to a chat message. The rationale elements are:

ISSUE A problem to be solved. Issues are typically resolved through discussions and negotiation. Issues are usually phrased as questions. Examples:

- *"I think you had a good question about difficulty level—whether it's for all recipes or for particular ones."*
- *"How soon should a dispatcher be notified of a train delay?"*
- *"How should persistent data be stored?"*
- *"Which technology presents the most risk?"*

ALTERNATIVE A possible solution that could address the issue under consideration. Examples:

- *"Just my thoughts, but what do you think of making the "cost" be for the whole dish, instead of next to each individual ingredient?"*
- *"The interface for the dispatcher could be realized with a point-and-click interface."*

- *“The display used by the dispatcher can be a text-only display with graphic characters to represent track segments.”*

PRO-ARGUMENT Reasons supporting an alternative. Pro-argument is usually phrased as a positive statement. Example:

- *“Text-based interfaces are easier to implement and test than Point-and-click interfaces.”*

When a developer supports to an alternative with a subjective opinion it should be marked as a pro-argument. Example:

- *“I like it.”*
- *“I totally agree with you!”*

CON-ARGUMENT Reasons against an alternative. Con-argument is usually phrased as a negative statement. Examples:

- *“This screen has no real useful functionality.”*
- *“Point-and-click interfaces are much more complex to implement than text-based interfaces.”*
- *“The point-and-click interface risks introducing fatal errors in the system that would offset any usability benefit the interface would provide.”*

DECISION A decision that were made to resolve an open issue. Examples:

- *“We decided that we will rank our recipes based on frequency of use.”*
- *“We select a text-based display and a keyboard input for the traffic control user interface.”*

WHAT NOT TO CLASSIFY?

MANAGEMENT ISSUES Issues that are not discussing the mobile application being developed. Example:

- *“Should I put the user stories in Confluence and JIRA?”*

GENERAL QUESTIONS Questions that are not related to the mobile application being developed. Examples:

- *“How can I log in to confluence?”*
- *“How can I see the project events on the calendar?”*
- *“How do I create a new storyboard in Xcode?”*
- *“How to sort any object array by dates on Swift?”*

COURSE REQUIREMENTS During the course project, students were required to deliver some materials for evaluating and presenting their work, such as a trailer and a presentation to be represented in the middle of the course duration (known as *Design Review*) and at the end of the course (known as *Client Acceptance Test*). Messages about the trailers preparations should not be marked as rationale related.

SOCIAL EVENTS Messages that discuss social aspects of the development teams.

Examples:

- Arrangement for an icebreaker event.
- Discussing the time and place for a team gathering.

DISTRIBUTION OF WORKLOAD Example:

- *“I would do it myself but I’m too afraid to break something. I would be very happy if someone with more knowledge maybe can reset things and create the structure we need.”*

NON-ENGLISH MESSAGES.

Coding Guide: For Annotating Rationale Elements in Developers' Chat Messages of Distributed Teams

Our study aims at studying what type of rationale elements are present in developers' IRC messages of open source projects.

As part of the coding task, you will read IRC messages exchanged between the developers of three open source projects: Apache Lucene¹, Mozilla Thunderbird², and Ubuntu³. Your task is to classify IRC messages that contain rationale elements.

This guide describes the instructions, which you should follow carefully in order to successfully conduct the task. You will use GATE⁴ for completing your task. We recommend to use this guide as a reference during the coding task.

Your task is to read the IRC messages assigned to you. For each message, you will:

- Identify rationale elements contained in the messages by highlighting the message and classifying it accordingly. We explain each rationale element later in this guide.

In case any questions arise during the coding, please contact the other project members.

CLASSIFYING THE IRC MESSAGE: If the message contains a rationale element, highlight the complete message and indicate the type of the rationale element.

You can assign more than one rationale element to each message. The rationale elements are:

¹ <https://lucene.apache.org/>

² <https://www.mozilla.org/thunderbird/>

³ <https://www.ubuntu.com/>

⁴ <https://gate.ac.uk>

ISSUE A problem to be solved. Issues are typically resolved through discussions and negotiation. Issues are usually phrased as questions. Examples:

- *“I think you had a good question about difficulty level—whether it’s for all recipes or for particular ones.”*
- *“How soon should a dispatcher be notified of a train delay?”*
- *“How should persistent data be stored?”*
- *“Which technology presents the most risk?”*

ALTERNATIVE A possible solution that could address the issue under consideration. Examples:

- *“Just my thoughts, but what do you think of making the “cost” be for the whole dish, instead of next to each individual ingredient?”*
- *“The interface for the dispatcher could be realized with a point-and-click interface.”*
- *“The display used by the dispatcher can be a text-only display with graphic characters to represent track segments.”*

PRO-ARGUMENT Reasons supporting an alternative. Pro-argument is usually phrased as a positive statement. Example:

- *“Text-based interfaces are easier to implement and test than Point-and-click interfaces.”*

When a developer supports to an alternative with a subjective opinion it should be marked as a pro-argument. Example:

- *“I like it.”*
- *“I totally agree with you!”*

CON-ARGUMENT Reasons against an alternative. Con-argument is usually phrased as a negative statement. Examples:

- *“This screen has no real useful functionality.”*
- *“Point-and-click interfaces are much more complex to implement than text-based interfaces.”*

- *“The point-and-click interface risks introducing fatal errors in the system that would offset any usability benefit the interface would provide.”*

DECISION A decision that were made to resolve an open issue. Examples:

- *“We decided that we will rank our recipes based on frequency of use.”*
- *“We select a text-based display and a keyboard input for the traffic control user interface.”*

WHAT NOT TO CLASSIFY?

MANAGEMENT ISSUES Issues that are not discussing the software system being developed. Example:

- *“Should I put the user stories in Confluence and JIRA?”*

GENERAL QUESTIONS AND ISSUES that are not related to the software system being developed. For example, questions about the development environment, development processes, programming languages, APIs or software libraries. Examples:

- *“How can I log in to confluence?”*
- *“How can I see the project events on the calendar?”*
- *“How do I create a new storyboard in Xcode?”*
- *“How to sort any object array by dates on Swift?”*

SOCIAL EVENTS Messages that discuss social aspects of the development teams. Example:

- Arrangement for an icebreaker event.
- Discussing the time and place for a team gathering.

DISTRIBUTION OF WORKLOAD Example:

- *“I would do it myself but I’m too afraid to break something. I would be very happy if someone with more knowledge maybe can reset things and create the structure we need.”*

NON-ENGLISH MESSAGES.

Coding Guide: For Annotating Rationale Elements in Issue Tracking Systems

Our study aims at studying what type of rationale elements are present in Issue Tracking Systems of open source projects.

As part of the annotation task, you will read issues and their comments of three open source projects: Apache Lucene¹, Mozilla Thunderbird² and Ubuntu³. Your task is to classify issue comments that contain rationale elements.

This guide describes the instructions, which you should follow carefully in order to successfully conduct the task. Together with this coding guide, you will receive a spreadsheet with the issues' comments to be coded. The spreadsheet contains for each issue to be coded: the *Issue Title*, *Issue Description*, *Issue Reporter*, *Issue Assignee*, and *Issue Comments*. A comment to an issue might consist of more than a sentence. For each comment, the *Comment author* and *Comment Sentences* are displayed. Each sentence is displayed in a single spreadsheet row.

Your task is to read the issue comments assigned to you. For each sentence in the comment, you will:

- Identify rationale elements contained in the sentence by assigning an X in the cell forming the intersection of the row containing the sentence and the column representing the rationale element. We explain each rationale element later in this guide.

We recommend to use this guide as a reference during the whole coding task. In case any questions arise during the coding, please contact the other project members.

¹ <https://lucene.apache.org/>

² <https://www.mozilla.org/thunderbird/>

³ <https://www.ubuntu.com/>

CLASSIFYING THE SENTENCES OF ISSUE COMMENTS: You can assign more than one rationale element (code) to each sentence. The rationale elements are:

ISSUE A problem to be solved. Issues are typically resolved through discussions and negotiation. Issues are usually phrased as questions.

ALTERNATIVE A possible solution that could address the issue under consideration.

PRO-ARGUMENT Reasons supporting an alternative. Pro argument is usually phrased as a positive statement. When a developer supports to an alternative with a subjective opinion it should be marked as a pro-argument.

CON-ARGUMENT Reasons against an alternative. Con argument is usually phrased as a negative statement.

DECISION A decision that were made to resolve an open issue.

WHAT NOT TO CLASSIFY?

MANAGEMENT ISSUES that are not discussing the app being developed.

GENERAL QUESTIONS AND ISSUES that are not related to the software system being developed. For example, questions about the development environment, development processes, programming languages, APIs or software libraries.

SOCIAL EVENTS: Messages that discuss social aspects of the development teams.

DISTRIBUTION OF WORKLOAD.

AUTOMATICALLY-GENERATED COMMENTS: including comments generated by bots. For example, jira-bot, githubbot and commit-tag-bot in Apache Lucene.

NON-ENGLISH MESSAGES.

Questionnaire for Evaluating REACT

This is a questionnaire about the use of rationale annotations (emojis) in Slack chat messages.

*** Required**

Q1. How do you rate your agreement with the following statements? *

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Slack rationale emojis are easy to learn.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Slack rationale emojis are simple to apply.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I enjoy using Slack rationale emojis to capture important knowledge.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Applying Slack rationale emojis helps in documenting the rationale behind the decisions we make in our team chat messages.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Applying Slack rationale emojis encourages all team members to participate in the ongoing discussion.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Q2. When applying annotations in Slack chat messages: *

- I fully support it for documenting important knowledge without any reservations.
- I have some concerns about my privacy, but I fully support it if the data is anonymized.
- I prefer if none of my discussions in chat messages are documented.
- Other: _____

Q3. Would you like to add any other comment?

Bibliography

- [1] J. Agg, "Harvesting Versus Creating: Effective Web Design Rationale," in *Proceedings of the 17th Australia Conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future (OZCHI)*, 2005, pp. 1–4.
- [2] H. Ajjan, R. Hartshorne, Y. Cao, and M. Rodriguez, "Continuance Use Intention of Enterprise Instant Messaging: a Knowledge Management Perspective," *Behaviour & information technology*, vol. 33, no. 7, pp. 678–692, 2014.
- [3] Aman-ul-haq and M. A. Babar, "Tool Support for Automating Architectural Knowledge Extraction," in *Proceedings of the ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, 2009, pp. 49–56.
- [4] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is It a Bug or an Enhancement?: A Text-based Approach to Classify Change Requests," in *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON)*, 2008, pp. 304–318.
- [5] J. Anvik, L. Hiew, and G. C. Murphy, "Who Should Fix This Bug?" In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, 2006, pp. 361–370.
- [6] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza, "Content Classification of Development Emails," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 375–385.
- [7] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes, "Benchmarking Lightweight Techniques to Link E-mails and Source Code," in *Proceedings of the 16th Working Conference on Reverse Engineering (WCRE)*, 2009, pp. 205–214.

- [8] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-Mails and Source Code Artifacts," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering (ICSE)*, 2010, pp. 375–384.
- [9] O. Baysal and A. J. Malton, "Correlating Social Interactions to Release History During Software Evolution," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 7–14.
- [10] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2010, pp. 291–300.
- [11] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, "Automatic Extraction of Design Decisions from Issue Management Systems: a Machine Learning Based Approach," in *Proceedings of the European Conference on Software Architecture (ECSA)*, 2017, pp. 138–154.
- [12] T. Bingham and M. Conner, *The New Social Learning: A Guide to Transforming Organizations Through Social Media*. Berrett-Koehler Publishers, 2010.
- [13] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining Email Social Networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR)*, 2006, pp. 137–143.
- [14] G. Breach, *I'm Not Chatting, I'm Innovating! Locating Lead Users in Open Source Software Communities*, University of Technology, Sydney School of Management, Working Paper Series, 2008.
- [15] H. Brücher, G. Knolmayer, and M.-A. Mittermayer, *Document Classification Methods for Organizing Explicit Knowledge*, Institute of Information Systems, 2002.
- [16] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 3rd Edition. Prentice Hall Press, 2009.
- [17] B. Bruegge, S. Krusche, and L. Alperowitz, "Software Engineering Project Courses with Industrial Clients," *ACM Transactions on Computing Education*, vol. 15, no. 4, pp. 1–31, 2015.

- [18] B. Bruegge, S. Krusche, and M. Wagner, "Teaching Tornado: From Communication Models to Releases," in *Proceedings of the 8th Edition of the Educators' Symposium (EduSymp)*, 2012, pp. 5–12.
- [19] A. Brühlmann, T. Gîrba, O. Greevy, and O. Nierstrasz, "Enriching reverse engineering with annotations," in *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2008, pp. 660–674.
- [20] J. Brunet, G. C. Murphy, R. Terra, J. Figueiredo, and D. Serey, "Do Developers Discuss Design?" In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 340–343.
- [21] J. E. Burge, "Software Engineering Using RATIONALE," PhD thesis, Worcester Polytechnic Institute, 2005.
- [22] J. E. Burge and D. C. Brown, "An Integrated Approach for Software Design Checking Using Design Rationale," in *Proceedings of the Conference on Design Computing and Cognition*, 2004, pp. 557–575.
- [23] J. E. Burge and D. C. Brown, "Rationale-Based Support for Software Maintenance," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds. Springer Berlin Heidelberg, 2006, pp. 273–296.
- [24] J. E. Burge and D. C. Brown, "Software Engineering Using RATIONALE," *Journal of Systems and Software*, vol. 81, no. 3, pp. 395–413, 2008.
- [25] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrík, *Rationale-Based Software Engineering*. Springer-Verlag, 2008.
- [26] K. C. Burgess Yakemovic and E. J. Conklin, "Report on a Development Project Use of an Issue-based Information System," in *Proceedings of the ACM Conference on Computer-supported Cooperative Work (CSCW)*, 1990, pp. 105–118.
- [27] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is Going to Mentor Newcomers in Open Source Projects?" In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 1–11.

- [28] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M. A. Babar, "10 Years of Software Architecture Knowledge Management," *Journal of Systems and Software*, vol. 116, pp. 191–205, 2016.
- [29] C. E. Carroll, *The Handbook of Communication and Corporate Reputation*. John Wiley & Sons, 2015.
- [30] M. Cataldo and J. D. Herbsleb, "Communication Networks in Geographically Distributed Software Development," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2008, pp. 579–588.
- [31] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.
- [32] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special Issue on Learning from Imbalanced Data Sets," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 1–6, 2004.
- [33] S. A. Chowdhury and A. Hindle, "Mining StackOverflow to Filter out Off-topic IRC Discussion," in *Proceedings of the 12th International Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 422–425.
- [34] E. J. Conklin and K. C. B. Yakemovic, "A Process-oriented Approach to Design Rationale," *Human-Computer Interaction*, vol. 6, no. 3, pp. 357–391, 1991.
- [35] J. Conklin and M. L. Begeman, "gIBIS: A Hypertext Tool for Team Design Deliberation," in *Proceedings of the ACM Conference on Hypertext (HYPERTEXT)*, 1987, pp. 247–251.
- [36] J. Conklin and M. L. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," in *Proceedings of the ACM Conference on Computer-supported Cooperative Work (CSCW)*, 1988, pp. 140–152.
- [37] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [38] H. Cunningham, V. Tablan, A. Roberts, and K. Bontcheva, "Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics," *PLoS Computational Biology*, vol. 9, no. 2, 2013.

- [39] A. De Lucia, F. Fasano, C. Grieco, and G. Tortora, "Recovering Design Rationale from Email Repositories," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, 2009, pp. 543–546.
- [40] P. DeGrace and L. H. Stahl, *Wicked Problems, Righteous Solutions: a Catalogue of Modern Software Engineering Paradigms*. Englewood Cliffs, N.J: Yourdon Press, 1990.
- [41] A. Dekhtyar, J. H. Hayes, and T. Menzies, "Text is Software Too," in *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR)*, 2004, pp. 22–26.
- [42] S. Dennerlein, R. Gutounig, E. Goldgruber, and S. Schweiger, "Web 2.0 Messaging Tools for Knowledge Management? Exploring the Potentials of Slack," in *Proceedings of the European Conference on Knowledge Management*, 2016, pp. 225–232.
- [43] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. Gall, "DECA: Development Emails Content Analyzer," in *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE)*, 2016, pp. 641–644.
- [44] T. G. Dietterich, "Ensemble Methods in Machine Learning," in *Proceedings of the International Workshop on Multiple Classifier Systems*, 2000, pp. 1–15.
- [45] Y. Dittrich and R. Giuffrida, "Exploring the Role of Instant Messaging in a Global Software Development Project," in *Proceedings of the 6th IEEE International Conference on Global Software Engineering (ICGSE)*, 2011, pp. 103–112.
- [46] C. Drummond and R. C. Holte, "C4. 5, Class Imbalance, and Cost Sensitivity: Why Under-sampling Beats Over-sampling," in *Proceedings of the Workshop on Learning from Imbalanced Datasets II*, 2003, pp. 1–8.
- [47] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, *Rationale Management in Software Engineering*. Springer-Verlag, 2006.
- [48] A. H. Dutoit and B. Paech, "Rationale Management in Software Engineering," in *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing Company, 2000, pp. 787–815.

- [49] B. Eisner, T. Rocktäschel, I. Augenstein, M. Bošnjak, and S. Riedel, “emoji2vec: Learning Emoji Representations from their Description,” in *Proceedings of the 4th International Workshop on Natural Language Processing for Social Media (SocialNLP)*, 2016, pp. 48–54.
- [50] M. Elliott and W. Scacchi, “Communicating and Mitigating Conflict in Open Source Software Development Projects,” *Projects & Profits*, pp. 25–41, 2002.
- [51] M. S. Elliott, “The Virtual Organizational Culture of a Free Software Development Community,” in *Proceedings of the 3rd Workshop on Open Source Software Engineering*, 2003, pp. 45–49.
- [52] M. S. Elliott and W. Scacchi, “Free Software Developers As an Occupational Community: Resolving Conflicts and Fostering Collaboration,” in *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP)*, 2003, pp. 21–30.
- [53] T. Erickson and W. A. Kellogg, “Social Translucence: An Approach to Designing Systems That Support Social Processes,” *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 1, pp. 59–83, 2000.
- [54] R. Feldman and J. Sanger, *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. New York, NY, USA: Cambridge University Press, 2006.
- [55] R. Fernández, M. Frampton, P. Ehlen, M. Purver, and S. Peters, “Modelling and Detecting Decisions in Multi-party Dialogue,” in *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue (SIGdial)*, 2008, pp. 156–163.
- [56] G. Fischer, R. McCall, and A. Morch, “JANUS: Integrating Hypertext with a Knowledge-based Design Environment,” in *Proceedings of the Second Annual ACM Conference on Hypertext (HYPERTEXT)*, 1989, pp. 105–117.
- [57] C. Francalanci and F. Merlo, “Empirical Analysis of the Bug Fixing Process in Open Source Projects,” in *Proceedings of the IFIP International Conference on Open Source Systems, Open Source Development, Communities and Quality*, 2008, pp. 187–196.

- [58] E. Frank and R. R. Bouckaert, "Naive Bayes for Text Classification with Unbalanced Classes," in *Proceedings of the 10th European Conference on Principle and Practice of Knowledge Discovery in Databases (PKDD)*, 2006, pp. 503–510.
- [59] B. Franklin, *The Way to Wealth*. 1758.
- [60] M. Friendly, "Corrgrams: Exploratory Displays for Correlation Matrices," *The American Statistician*, vol. 56, no. 4, pp. 316–324, 2002.
- [61] A. C. B. Garcia and H. C. Howard, "Acquiring Design Knowledge Through Design Decision Justification," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 6, no. 1, pp. 59–71, 1992.
- [62] S. Gill and E. V. Munson, "A Version-Aware Tool for Design Rationale," in *Proceedings of the 12th Brazilian Symposium on Multimedia and the Web*, ACM, 2006, pp. 20–26.
- [63] R. Giuffrida and Y. Dittrich, "Empirical Studies on the Use of Social Software in Global Software Development - A Systematic Mapping Study," *Information and Software Technology*, vol. 55, no. 7, pp. 1143–1164, 2013.
- [64] T. R. Gruber and D. M. Russell, "Generative Design Rationale: Beyond the Record and Replay Paradigm," in *Design Rationale*, T. P. Moran and J. M. Carroll, Eds., L. Erlbaum Associates Inc., 1996, pp. 323–349.
- [65] C. Gutwin, R. Penner, and K. Schneider, "Group Awareness in Distributed Software Development," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2004, pp. 72–81.
- [66] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in Open Source Software Development Mailing Lists," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 277–286.
- [67] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

- [68] C. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg, "Designing Task Visualizations to Support the Coordination of Work in Software Development," in *Proceedings of Computer-Supported Cooperative Work and Social Computing (CSCW)*, 2006, pp. 39–48.
- [69] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd. Morgan Kaufmann Publishers Inc., 2011.
- [70] M. Handel and J. D. Herbsleb, "What is Chat Doing in the Workplace?" In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2002, pp. 1–10.
- [71] A. E. Hassan, "The Road Ahead for Mining Software Repositories," in *Proceedings of Frontiers of Software Maintenance (FoSM)*, 2008, pp. 48–57.
- [72] T. M. Hesse, A. Kuehlwein, and T. Roehm, "DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally," in *Proceedings of the 1st International Workshop on Decision Making in Software ARCHitecture (MARCH)*, 2016, pp. 30–37.
- [73] T. M. Hesse, B. Paech, T. Roehm, and B. Bruegge, "How to improve decision documentation in software evolution?" In *Proceedings of the First Collaborative Workshop on Evolution and Maintenance of Long-Living Systems (EMLS)*, 2014, pp. 14–15.
- [74] T.-M. Hesse, S. Gartner, T. Roehm, B. Paech, K. Schneider, and B. Bruegge, "Semiautomatic Security Requirements Engineering and Evolution Using Decision Documentation, Heuristics, and User Monitoring," in *Proceedings of the IEEE 1st Workshop on Evolving Security and Privacy Requirements Engineering (ESPRES)*, 2014, pp. 1–6.
- [75] T.-M. Hesse, C. Kücherer, and B. Paech, "Experiences with Supporting the Distributed Responsibility for Requirements through Decision Documentation," in *Proceedings of the GI-Fachgruppen-Treffen Requirements Engineering (FGRE)*, 2014.
- [76] T.-M. Hesse, A. Kuehlwein, B. Paech, T. Roehm, and B. Bruegge, "Documenting Implementation Decisions with Code Annotations," in *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2015, pp. 152–157.

- [77] T.-M. Hesse, V. Lerche, M. Seiler, K. Knoess, and B. Paech, "Documented Decision-making Strategies and Decision Knowledge in Open Source Projects," *Information and Software Technology*, vol. 79, pp. 36–51, 2016.
- [78] T.-M. Hesse and B. Paech, "Supporting the Collaborative Development of Requirements and Architecture Documentation," in *Proceedings of the 3rd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, 2013, pp. 22–26.
- [79] T.-M. Hesse and B. Paech, "Documenting Relations Between Requirements and Design Decisions: A Case Study on Design Session Transcripts," in *Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2016, pp. 188–204.
- [80] B. Hicks, S. Culley, R. Allen, and G. Mullineux, "A Framework for The Requirements of Capturing, Storing and Reusing Information and Knowledge in Engineering Design," *International Journal of Information Management*, vol. 22, no. 4, pp. 263–280, 2002.
- [81] T. K. Ho, "Random Decision Forests," in *Proceedings of the 3rd International Conference on Document Analysis and Recognition (ICDAR)*, 1995, pp. 278–282.
- [82] P.-Y. Hsueh and J. D. Moore, "What Decisions Have You Made?: Automatic Decision Detection in Meeting Conversations," in *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2007, pp. 25–32.
- [83] P.-Y. Hsueh and J. D. Moore, "Automatic Decision Detection in Meeting Speech," in *Proceedings of the International Workshop on Machine Learning for Multimodal Interaction*, 2008, pp. 168–179.
- [84] P. Hübner and B. Paech, "Using Interaction Data for Continuous Creation of Trace Links Between Source Code and Requirements in Issue Tracking Systems," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, P. Grünbacher and A. Perini, Eds., 2017, pp. 291–307.

- [85] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text Classification Using Machine Learning Techniques," *WSEAS Transactions on Computers*, vol. 4, no. 8, pp. 966–974, 2005.
- [86] A. Jarczyk, P. Loffler, and F. Shipmann, "Design Rationale for Software Engineering: A Survey," in *Proceedings of the 25th Hawaii International Conference on System Sciences*, 1992, pp. 577–586.
- [87] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "Towards the Visualization of Usage and Decision Knowledge in Continuous Software Engineering," in *Proceedings of the IEEE Working Conference on Software Visualization (VISOFT)*, 2017, pp. 104–108.
- [88] A. Johri, "Look Ma, No Email!: Blogs and IRC As Primary and Preferred Communication Tools in a Distributed Firm," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2011, pp. 305–308.
- [89] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3rd ed. draft. 2017.
- [90] V. Käfer, D. Graziotin, I. Bogicevic, S. Wagner, and J. Ramadani, "Communication in Open-Source Projects—End of the E-mail Era?" In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018.
- [91] Y. Kato, K. Hori, and K. Taketa, "Capturing Design Rationale by Annotating E-mails," in *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, 2002, pp. 278–282.
- [92] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, and G. Valetto, "Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 220–260, 2016.
- [93] A. Kleebaum, J. O. Johanssen, B. Paech, R. Alkadhi, and B. Bruegge, "Decision Knowledge Triggers in Continuous Software Engineering," in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, 2018, pp. 23–26.

- [94] A. Kleebaum, J. O. Johanssen, B. Paech, and B. Bruegge, "Tool Support for Decision and Usage Knowledge in Continuous Software Engineering," in *Proceedings of the 3rd Workshop on Continuous Software Engineering (CSE)*, 2018, pp. 74–77.
- [95] A. J. Ko, "A Three-year Participant Observation of Software Startup Software Evolution," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2017, pp. 3–12.
- [96] A. J. Ko and P. K. Chilana, "Design, Discussion, and Dissent in Open Bug Reports," in *Proceedings of the iConference*, 2011, pp. 106–113.
- [97] A. J. Ko, R. DeLine, and G. Venolia, "Information Needs in Collocated Software Development Teams," in *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, 2007, pp. 344–353.
- [98] P. Kruchten, R. Capilla, and J. C. Dueñas, "The Decision View's Role in Software Architecture Practice," *IEEE Software*, vol. 26, no. 2, pp. 36–42, 2009.
- [99] P. Kruchten, "An Ontology of Architectural Design Decisions in Software Intensive Systems," in *Proceedings of the 2nd Groningen Workshop on Software Variability*, 2004, pp. 54–61.
- [100] S. Krusche and L. Alperowitz, "Introduction of Continuous Delivery in Multi-customer Project Courses," in *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion)*, 2014, pp. 335–343.
- [101] S. Krusche, L. Alperowitz, B. Bruegge, and M. O. Wagner, "Rugby: An Agile Process Model Based on Continuous Delivery," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE)*, 2014, pp. 42–50.
- [102] W. Kunz and H. Rittel, "Issues as Elements of Information Systems," Institute of Urban and Regional Development, University of California, Berkeley, California, Working Paper 131, 1970.

- [103] Z. Kurtanović and W. Maalej, "Mining User Rationale from Software Reviews," in *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*, 2017, pp. 61–70.
- [104] Z. Kurtanović and W. Maalej, "On User Rationale in Software Engineering," *Requirements Engineering*, pp. 1–23, 2018.
- [105] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining Mental Models: A Study of Developer Work Habits," in *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, 2006, pp. 492–501.
- [106] J. Lee Rodgers and W. A. Nicewander, "Thirteen Ways to Look at the Correlation Coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.
- [107] J. Lee, "Decision Representation Language (DRL) and Its Support Environment," MIT Artificial Intelligence Laboratory, Working Paper, 1989.
- [108] J. Lee, "Artificial Intelligence at MIT Expanding Frontiers," in P. H. Winston and S. A. Shellard, Eds., MIT Press, 1990, ch. SIBYL: A Qualitative Decision Management System, pp. 104–133.
- [109] J. Lee, "Extending the Potts and Bruns Model for Recording Design Rationale," in *Proceedings of the 13th International Conference on Software Engineering (ICSE)*, 1991, pp. 114–125.
- [110] J. Lee, "Design Rationale Systems: Understanding the Issues," *IEEE Expert*, vol. 12, no. 3, pp. 78–85, 1997.
- [111] J. Lee and K.-Y. Lai, "What's in Design Rationale?" *Human-Computer Interaction*, vol. 6, no. 3, pp. 251–280, 1991.
- [112] T. C. Lethbridge, J. Singer, and A. Forward, "How Software Engineers Use Documentation: The State of the Practice," *IEEE Software*, vol. 20, no. 6, pp. 35–39, 2003.
- [113] C. Lewis, J. Rieman, and B. Bell, "Problem-centered Design for Expressiveness," in *Design Rationale: Concepts, Techniques, and Use*, T. P. Moran and J. M. Carroll, Eds., L. Erlbaum Associates Inc., 1996, pp. 147–184.

- [114] D. D. Lewis, "Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval," in *Proceedings of the 10th European Conference on Machine Learning (ECML)*, 1998, pp. 4–15.
- [115] Z. Li and K. Ramani, "Ontology-Based Design Information Extraction and Retrieval," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 21, pp. 137–154, 2007.
- [116] Y. Liang, Y. Liu, C. K. Kwong, and W. B. Lee, "Learning the "Whys": Discovering Design Rationale Using Text Mining - An Algorithm Perspective," *Computer-Aided Design*, vol. 44, no. 10, pp. 916–930, 2012.
- [117] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik, "Why Developers Are Slacking Off : Understanding How Software Teams Use Slack," in *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion (CSCW Companion)*, 2016, pp. 333–336.
- [118] Y. Liu, Y. Liang, C. K. Kwong, and W. B. Lee, "A New Design Rationale Representation Model for Rationale Mining," *Journal of Computing and Information Science in Engineering*, vol. 10, no. 3, pp. 1–10, 2010.
- [119] C. López, V. Codocedo, H. Astudillo, and L. M. Cysneiros, "Bridging the Gap Between Software Architecture Rationale Formalisms and Actual Architecture Documents: An Ontology-driven Approach," *Science of Computer Programming*, vol. 77, no. 1, pp. 66–80, 2012.
- [120] R. Lougher and T. Rodden, "Supporting Long-term Collaboration in Software Maintenance," in *Proceedings of the Conference on Organizational Computing Systems (COCS)*, 1993, pp. 228–238.
- [121] A. MacLean, R. M. Young, V. M. E. Bellotti, and T. P. Moran, "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human-Computer Interaction*, vol. 6, no. 3, pp. 201–250, 1991.
- [122] R. McCall, I. Mistrik, and W. Schuler, "An Integrated Information and Communication System for Problem Solving," in *Proceedings of the 7th International CODATA Conference*, 1981, pp. 107–115.

- [123] R. McCall, "PHIBIS: Procedurally Hierarchical Issue-based Information Systems," in *Proceedings of the International Congress on Planning and Design Theory*, 1987, pp. 17–22.
- [124] R. J. McCall, "MIKROPLIS: A Hypertext System for Design," *Design studies*, vol. 10, no. 4, pp. 228–238, 1989.
- [125] R. J. McCall, "PHI: A Conceptual Foundation for Design Hypermedia," *Design studies*, vol. 12, no. 1, pp. 30–41, 1991.
- [126] R. McCall, P. R. Bennett, P. S. d'Oronzio, J. L. Ostwald, F. M. Shipman III, and N. F. Wallace, "PHIDIAS: Integrating CAD Graphics into Dynamic Hypertext," in *Proceedings of the European Conference on Hypertext (ECHT)*, 1990, pp. 152–165.
- [127] R. McCall and I. Mistrik, "Capture of Software Requirements and Rationale through Collaborative Software Sevelopment," in *Requirements Engineering for Sociotechnical Systems*, A. S. José Luis Maté, Ed. Information Science Publishing, 2005, pp. 303–317.
- [128] A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," in *Proceedings of the AAAI/ICML Workshop on Learning for Text Categorization*, 1998, pp. 41–48.
- [129] M. McCandless and O. Gospodnetic, *Lucene in Action*. Manning Publications Co., 2005.
- [130] I. McCowan, J. Carletta, W. Kraaij, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, M. Kronenthal, G. Lathoud, M. Lincoln, A. Lisowska, W. Post, D. Reidsma, and P. Wellner, "The AMI Meeting Corpus," in *Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research*, 2005, pp. 28–39.
- [131] S. Menaka and N. Radha, "Text Classification Using Keyword Extraction Technique," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 12, 2013.
- [132] C. Miesbauer and R. Weinreich, "Classification of Design Decisions: An Expert Survey in Practice," in *Proceedings of the 7th European Conference on Software Architecture (ECSA)*, 2013, pp. 130–145.

- [133] T. P. Moran and J. M. Carroll, Eds., *Design Rationale: Concepts, Techniques, and Use*. L. Erlbaum Associates Inc., 1996.
- [134] K. L. Myers, N. B. Zumel, and P. Garcia, "Automated Capture of Rationale for the Detailed Design Process," in *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI)*, 1999, pp. 876–883.
- [135] M. Nagel, "The QUARC Metamodel: A Communication-Based Generic Project Model," PhD thesis, Technische Universität München, 2012.
- [136] G. Navarro, "A Guided Tour to Approximate String Matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.
- [137] R. Neches, W. R. Swartout, and J. D. Moore, "Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development," *IEEE Transactions on Software Engineering - Special Issue on Artificial Intelligence and Software Engineering*, vol. 11, no. 11, pp. 1337–1351, 1985.
- [138] K. A. Neuendorf, *The Content Analysis Guidebook*. SAGE Publications, 2002.
- [139] R. Nordquist. (2018). Noise and Interference in Various Types of Communication, [Online]. Available: thoughtco.com/noise-communication-term-1691349 (visited on 05/03/2018).
- [140] D. Pagano and W. Maalej, "How Do Developers Blog?: An Exploratory Study," in *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR)*, 2011, pp. 123–132.
- [141] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora, "Mining Source Code Descriptions from Developer Communications," in *Proceedings of the IEEE 20th International Conference on Program Comprehension (ICPC)*, 2012, pp. 63–72.
- [142] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, and G. Antoniol, "How Developers' Collaborations Identified from Different Sources Tell Us About Code Changes," in *Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 251–260.

- [143] L. Pascarella and A. Bacchelli, "Classifying Code Comments in Java Open-source Software Systems," in *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 227–237.
- [144] F. Peña-Mora, D. Sriram, and R. Logcher, "Design Rationale for Computer-Supported Conflict Mitigation," *Journal of Computing in Civil Engineering*, vol. 9, no. 1, pp. 57–72, 1995.
- [145] C.-Y. J. Peng, K. L. Lee, and G. M. Ingersoll, "An Introduction to Logistic Regression Analysis and Reporting," *The Journal of Educational Research*, vol. 96, no. 1, pp. 3–14, 2002.
- [146] M. Poppendieck, "Wicked Problems," *Software Development Magazine*, 2002.
- [147] M. Pushpa and S. Karpagavalli, "Multi-label Classification: Problem Transformation methods in Tamil Phoneme classification," *Procedia Computer Science*, vol. 115, pp. 572–579, 2017.
- [148] H. J. Rea, J. R. Corney, J. M. Ritchie, R. Sung, and C. Salamon, "Automating Digital Capture of Engineering Knowledge," in *Proceedings of the 4th International CIRP-sponsored conference (DET)*, 2007, pp. 19–21.
- [149] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, "MEKA: A Multi-label/Multi-target Extension to Weka," *Journal of Machine Learning Research*, vol. 17, no. 21, pp. 1–5, 2016.
- [150] B. Reeves and F. Shipman, "Making It Easy for Designers to Provide Design Rationale," in *Proceedings of the AAAI Workshop on Design Rationale Capture and Use*, 1992, pp. 1–7.
- [151] W. C. Regli, X. Hu, M. Atwood, and W. Sun, "A Survey of Design Rationale Systems: Approaches, Representation, Capture and Retrieval," *Engineering with Computers*, vol. 16, no. 3, pp. 209–235, 2000.
- [152] G. L. Rein and C. A. Ellis, "rIBIS: a Real-time Group Hypertext System," *International Journal of Man-Machine Studies*, vol. 34, no. 3, pp. 349–367, 1991.
- [153] H. W. J. Rittel, "On the Planning Crisis: Systems Analysis of the 'First and Second Generations'," *Bedriftsøkonomen*, vol. 8, pp. 390–398, 1972.

- [154] H. W. J. Rittel, "Second Generation Design Methods," *Interview in: Design Methods Group 5th Anniversary Report: DMG Occasional Paper*, vol. 1, pp. 5–10, 1972, Interviewed by Donald P. Grant and Jean-Pierre Protzen, *Developments in Design Methodology* (reprinted).
- [155] H. W. J. Rittel and M. M. Webber, "Dilemmas in a General Theory of Planning," *Policy Sciences*, vol. 4, no. 2, pp. 155–169, 1973.
- [156] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Morenox, D. Shepherdxi, and E. Wong, "On-Demand Developer Documentation," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 479–483.
- [157] G. Robles and J. M. Gonzalez-Barahona, "Developer Identification Methods for Integrated Data from Various Sources," in *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, 2005, pp. 1–5.
- [158] P. Rodeghero, S. Jiang, A. Armaly, and C. McMillan, "Detecting User Story Information in Developer-client Conversations to Generate Extractive Summaries," in *Proceedings of the 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 49–59.
- [159] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How Do Professional Developers Comprehend Software?" In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 255–265.
- [160] B. Rogers, J. Gung, Y. Qiao, and J. E. Burge, "Exploring Techniques for Rationale Extraction from Existing Documents," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1313–1316.
- [161] B. Rogers, C. Justice, T. Mathur, and J. E. Burge, "Generalizability of Document Features for Identifying Rationale," in *Proceedings of the 7th International Conference on Design Computing and Cognition (DCC)*, 2016, pp. 633–651.
- [162] B. Rogers, Y. Qiao, J. Gung, T. Mathur, and J. E. Burge, "Using Text Mining Techniques to Extract Rationale from Existing Documentation," in *Proceedings of the 6th International Conference on Design Computing and Cognition (DCC)*, 2014, pp. 457–474.

- [163] R. R. Rosario, "A Data Augmentation Approach to Short Text Classification," PhD thesis, University of California, Los Angeles, 2017.
- [164] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Publishing, 2012.
- [165] I. Rus and M. Lindvall, "Guest Editors' Introduction: Knowledge Management in Software Engineering," *IEEE Software*, vol. 19, no. 3, pp. 26–38, 2002.
- [166] J. Sagoo, A. Tiwari, and J. Alcock, "Reviewing the State-of-the-Art Design Rationale Definitions, Representations and Capabilities," *International Journal of Design Engineering*, vol. 5, no. 3, pp. 211–231, 2014.
- [167] I. Salman, A. T. Misirli, and N. Juristo, "Are Students Representatives of Professionals in Software Engineering Experiments?" In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015, pp. 666–676.
- [168] K. Schneider, "Rationale as a By-Product," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, Eds., Springer Berlin Heidelberg, 2006, pp. 91–109.
- [169] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.
- [170] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.
- [171] A. Shahbazian, Y. K. Lee, D. Le, Y. Brun, and N. Medvidovic, "Recovering Architectural Design Decisions," in *Proceedings of the IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 95–104.
- [172] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying The Use of Developer IRC Meetings in Open Source Projects," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, 2009, pp. 147–156.
- [173] E. Shihab, Z. M. Jiang, and A. E. Hassan, "On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project," in *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories (MSR)*, 2009, pp. 107–110.

- [174] F. M. Shipman and R. J. McCall, "Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 11, no. 2, pp. 141–154, 1997.
- [175] F. M. Shipman III, "Supporting Knowledge-base Evolution with Incremental Formalization," PhD thesis, Boulder, CO, USA, 1993.
- [176] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An Examination of Software Engineering Work Practices," in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, 1997, pp. 1–15.
- [177] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall, "Development Emails Content Analyzer: Intention Mining in Developer Discussions," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 12–23.
- [178] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby, "Shared Waypoints and Social Tagging to Support Collaboration in Software Development," in *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW)*, 2006, pp. 195–198.
- [179] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. Muller, "How Software Developers Use Tagging to Support Reminding and Re-finding," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 470–483, 2009.
- [180] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (R) Evolution of Social Media in Software Engineering," in *Proceedings of the on Future of Software Engineering (FOSE)*, 2014, pp. 100–116.
- [181] M.-A. Storey, A. Zagalsky, F. Figueira Filho, L. Singer, and D. M. German, "How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 185–204, 2017.
- [182] R. Sung, J. M. Ritchie, H. J. Rea, and J. Corney, "Automated Design Knowledge Capture and Representation in Single-User CAD Environments," *Journal of Engineering Design*, vol. 22, no. 7, pp. 487–503, 2011.

- [183] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A Survey of Architecture Design Rationale," *Journal of Systems and Software*, vol. 79, no. 12, pp. 1792–1804, 2006.
- [184] Y. Tao and S. Kim, "Partitioning Composite Code Changes to Facilitate Code Review," in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, 2015, pp. 180–190.
- [185] S. E. Toulmin, *The Uses of Argument*. UK: Cambridge University Press, 1958.
- [186] C. Treude and M.-A. Storey, "Work Item Tagging: Communicating Concerns in Collaborative Software Development," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 19–34, 2012.
- [187] G. Tsoumakas and I. Katakis, "Multi-label Classification: An Overview," *International Journal of Data Warehousing and Mining*, vol. 3, pp. 1–13, 2007.
- [188] E. Ukkonen, "Algorithms for Approximate String Matching," *Information and Control*, vol. 64, no. 1-3, pp. 100–118, 1985.
- [189] G. Venolia, "Textual Allusions to Artifacts in Software-Related Repositories," in *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, 2006, pp. 151–154.
- [190] S. Vijayarani, M. J. Ilamathi, and M. Nithya, "Preprocessing Techniques for Text Mining- an overview," *International Journal of Computer Science & Communication Networks*, vol. 5, no. 1, pp. 7–16, 2015.
- [191] D. Vyas, T. Capel, D. Tank, and D. Shepherd, "Understanding the Use of a Bug Tracking System in a Global Software Development Setup," in *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction (OzCHI)*, 2015, pp. 222–226.
- [192] H. Wang, A. L. Johnson, and R. H. Bracewell, "The Retrieval of Structured Design Rationale for the Re-use of Design Knowledge with an Integrated Representation," *Advanced Engineering Informatics*, vol. 26, no. 2, pp. 251–266, 2012.
- [193] A. Williams and A. Rainer, "Identifying Practitioners' Arguments and Evidence in Blogs: Insights from a Pilot Study," in *Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016, pp. 345–348.

- [194] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016.
- [195] L. Yu, S. Ramaswamy, A. Mishra, and D. Mishra, "Communications in Global Software Development: An Empirical Study Using GTK+ OSS Repository," in *Proceedings of the Confederated International Conference on On the Move to Meaningful Internet Systems (OTM)*, 2011, pp. 218–227.
- [196] Y. C. Yuan, X. Zhao, Q. Liao, and C. Chi, "The Use of Different Information and Communication Technologies to Support Knowledge Sharing in Organizations: from E-mail to Micro-blogging," *Journal of the American Society for Information Science and Technology*, vol. 64, no. 8, pp. 1659–1670, 2013.
- [197] M.-L. Zhang and Z.-H. Zhou, "A Review on Multi-Label Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837, 2014.