

REACT: An Approach for Capturing Rationale in Chat Messages

Rana Alkadhi*, Jan Ole Johanssen*, Emitza Guzman[†], and Bernd Bruegge*

*Technical University of Munich
Department of Informatics
Garching b. München, Germany
{alkadhi, jan.johanssen, bruegge}@in.tum.de

[†]University of Zurich
Department of Informatics
Zurich, Switzerland
guzman@ifi.uzh.ch

Abstract—Background: Developers’ chat messages are a rich source of rationale behind development decisions. Rationale comprises valuable knowledge during software evolution for understanding and maintaining the software system. However, developers resist explicit methods for rationale capturing in practice, due to their intrusiveness and cognitive overhead. **Aim:** Our primary goal is to help developers capture rationale in chat messages with low effort. Further, we seek to encourage the collaborative capturing of rationale in development teams. **Method:** In this paper, we present REACT, a lightweight approach for annotating chat messages that contain rationale. To evaluate the feasibility of REACT, we conducted two studies. In the first study, we evaluated the approach with eleven development teams during a short-term design task. In the second study, we evaluated the approach with one development team over a duration of two months. In addition, we distributed a questionnaire to both studies’ participants. **Results:** Our results show that REACT is easily learned and used by developers. Furthermore, it encourages the collaborative capturing of rationale. Remarkably, the majority of participants do not perceive privacy as a barrier when capturing rationale from their informal communication. **Conclusions:** REACT is a first step towards enhancing rationale capturing in developers’ chat messages.

I. INTRODUCTION

Software development is a social and collaborative process [14] during which software developers make various decisions. The justification behind these decisions, other alternatives considered and the argumentation that led to the decisions constitute the software *rationale* [15]. Rationale has many potential uses during software evolution. For example, rationale serves as a documentation of development decisions and improves the understandability of the software system [8]. Furthermore, documented rationale can support maintenance and change impact analysis [4], [8]. However, explicit capturing of rationale rarely happens in practice and rationale remains hidden in developers’ communication and development artifacts [17]. Kruchten et al. [12] argue that the adoption barrier for capturing rationale is the intrusiveness of capturing activities, as they are not fully integrated into current software engineering practices—i.e., the developers must change their context in order to document rationale.

Our ongoing research on rationale capturing revealed that chat messages constitute a rich source of rationale about software systems [2]. In this paper, we present REACT

(RationalE Annotations in ChaT messages), a novel lightweight approach to capture rationale in developers’ chat messages. REACT is designed to be integrated with developers’ messaging platforms and enables developers to annotate the rationale present in their messages with emojis—pictographs commonly used in text-based communications. Developers use REACT to individually annotate their own messages or collaboratively annotate messages posted by other team members. To the best of our knowledge, this work is the first to propose a method for explicit capturing of rationale in developers’ chat messages.

In the context of this paper, we evaluated REACT within Slack, a widely used messaging platform for exchanging chat messages in development teams. Nevertheless, REACT is designed to be easily applicable to other messaging platforms. Our results show that the proposed approach is easy to learn and simple to apply, thus lowering the adoption barrier of rationale capturing by developers. Additionally, our findings highlight the importance of providing developers with incentives for justifying the efforts of capturing rationale.

The contribution of this paper is twofold. First, we present REACT, a lightweight approach for capturing rationale in chat messages. Second, we evaluate the approach feasibility in two studies: a short-term study in a design task and a medium-term study in a software project. Furthermore, we discuss the results of a questionnaire distributed to the study participants.

II. RELATED WORK

To the best of our knowledge, no previous work has investigated the explicit capturing of rationale from developers’ chat messages. However, few annotation approaches were developed to capture rationale within source code. Hesse et al. [10] developed an approach for capturing rationale of implementation decisions using inline code comments in source code. In their approach, text-based annotations are written directly into the code files. Lougher and Rodden [18] proposed a system for capturing maintenance rationale and generating documentation by attaching comments to the source code. Our work differs from Hesse et al. [10] and Lougher and Rodden [18] in that we propose a lightweight approach for capturing rationale in informal developers’ communication through annotating chat messages by developers themselves.

TABLE I: Rationale elements definitions (adapted from [13]).

| Rationale Element | Definition |
|-------------------|--|
| Issue | Problem that needs discussion and negotiation to be solved. Issues typically can not be resolved algorithmically and don't have a single correct solution. |
| Alternative | Possible solution that could address the issue under consideration. |
| Pro-argument | Positive argument supporting an alternative. |
| Con-argument | Negative argument against an alternative. |
| Decision | The alternative selected to resolve an open issue. |






In addition, developers are not required to write further descriptions of rationale when annotating chat messages.

The rapid growth of messaging platforms has attracted the attention of researchers studying their use in software development teams. Lin et al. [16] found that chat messages are mainly used by developers for communication, collaboration and knowledge sharing with team members. There are a few studies investigating the role of instant messages in knowledge management. Dennerlein et al. [7] found that the efficacy of messaging tools for knowledge management is mainly affected by the intended usage of these tools rather than the context of application. Ajjan et al. [1] found that the continuous use of instant messaging positively impacts the creation, retention and transfer of knowledge. In our previous work [2], we explored the nature of rationale in developers' chat messages and the potential of using machine learning techniques for the automatic extraction of rationale. In our current work, developers themselves annotate the rationale in their messages on the spot, whereas in our previous work the extraction is done by a supervised machine learning algorithm. Supervised machine learning algorithms have the disadvantage of needing a truth set—which is usually created manually, requiring a great effort [2]. The creation of such a truth set could dissuade teams from using this type of techniques. In contrast, in this work, we were interested in studying mechanisms for capturing rationale that do not need much effort to be put into practice.

III. REACT APPROACH

REACT is a lightweight approach for capturing rationale in developers' chat messages. The approach 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* [13]. The rationale elements definitions are listed in Table I.

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.

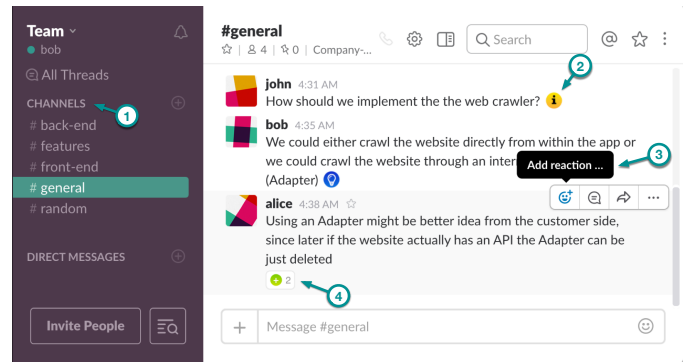




Fig. 1: Example of a conversation in Slack. (1) Team conversations are organized into *channels*, (2) emojis can be included *inline* within messages or as (3) a *reaction* and (4) the added reactions appear under the message.

Our use of emojis as rationale annotations was motivated by three factors. First, emojis are very popular in modern text communication and social media [9]. 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 paper, we implemented our approach within Slack; a real-time messaging platform. Figure 1 shows a screenshot of a conversation in Slack and describes its main components.

We added REACT rationale annotations as custom Slack emojis². 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 [2]. 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 .

IV. REACT EVALUATION

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

The study participants were part of a capstone course at the Technical University of Munich in 2017 [3]. During the course, students are organized in teams and develop mobile applications for industrial clients. Each development team consisted of eight

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

²<https://get.slack.help/hc/en-us/articles/206870177-Create-custom-emoji>

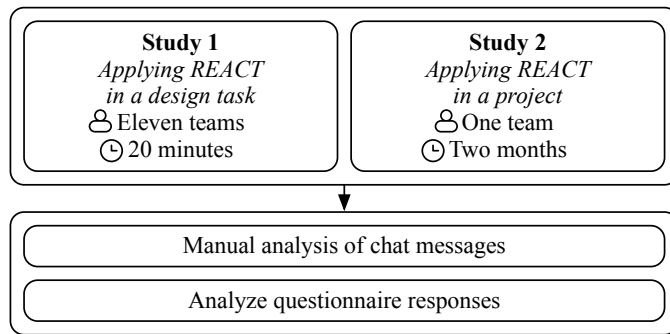


Fig. 2: Evaluation method.

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 to qualitatively investigate **Privacy** and further understand developers’ opinion on this issue after using REACT. Figure 2 shows the evaluation method. Detailed evaluation settings and results are presented in Sections V, VI and VII, respectively.

V. STUDY 1: REACT IN A SHORT-TERM DESIGN TASK

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

A. 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 arrived 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 authors of this paper studied how the eleven development teams applied REACT in chat messages by analyzing the chat messages manually.

B. Study Results

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

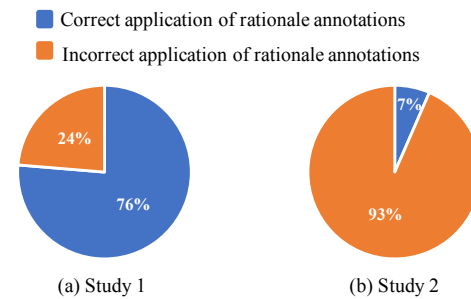


Fig. 3: Correctness analysis.

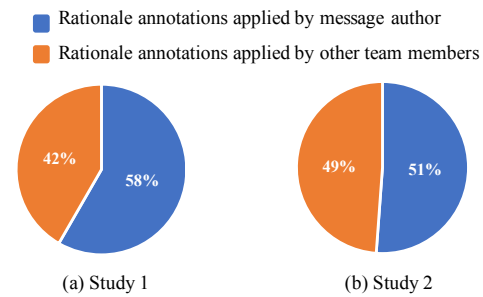







Fig. 4: Collaborativeness analysis.

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 developers annotate 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 3a shows the percentages of correct and incorrect application of annotations in the eleven teams’ messages. In 76% of the annotation applications, developers 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 4a, 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 [11].

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

TABLE II: Relative frequency of REACT rationale annotations and examples of annotated messages.

| Rationale Annotation | Relative 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.

annotation of chat messages by their authors as well as collaboratively by other team members.

We also 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.

VI. STUDY 2: REACT IN A MEDIUM-TERM PROJECT

The aim of this study is 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).

A. 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 V). 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

TABLE III: Chat messages analyzed in the Study 2.

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

of the study, participants were encouraged to ask for help or clarifications when faced with ambiguity while using REACT.

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. Table III 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 V). 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 [19] on the team’s chat messages.

During the content analysis, we developed a coding guide that provided definitions and examples of rationale elements³. Each of the 1,698 messages was annotated by two coders. One of the coders was the project leader of the team which allowed for a more informed assessment of messages containing rationale. We used GATE (General Architecture for Text Engineering) [5], [6] for the manual coding. During the coding task, the messages were shown without the applied REACT annotations, to avoid any bias. The two coders reported an average of 4.3 hours for the task. As a final step, the two coders discussed and resolved their disagreements.

³<https://cloudbruegge.in.tum.de/index.php/s/kz2S18PprNpbwUR>

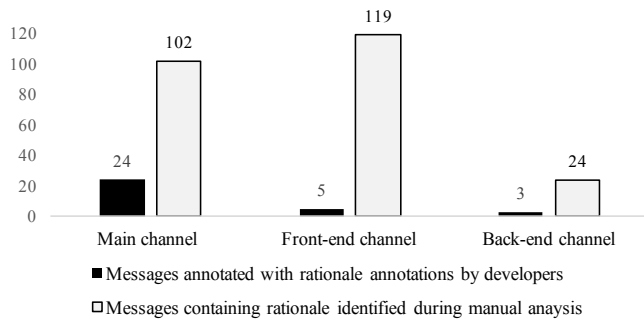


Fig. 5: Rationale in team messages of Study 2.

B. Study Results

Figure 5 compares the number of team messages annotated by developers and the number of messages identified as containing rationale during the manual analysis. 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 (2 months). The relative frequencies and examples of rationale annotations are listed in Table II.

We further found that only 7% of the messages containing rationale were annotated correctly by developers (shown in Figure 3b). A possible explanation for this result may be the lack of incentive for developers to annotate messages, even with a well-integrated capturing tool, if they can not 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 4b, 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%. Furthermore, 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.

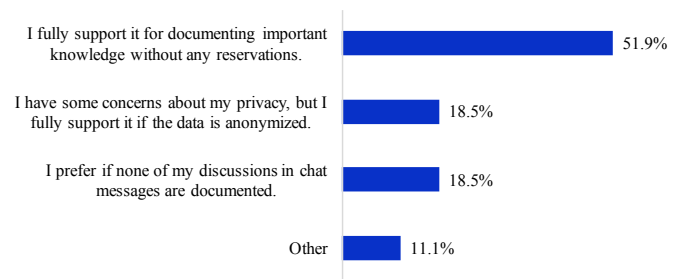


Fig. 6: Privacy analysis.

VII. QUESTIONNAIRE

After the completion of the two studies, an online questionnaire was distributed to the study participants to obtain developers’ opinion after using REACT⁴.

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, 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 finding [2] 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 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 presents participants responses when asked if they have any privacy concerns about annotating their chat

⁴<https://cloudbruegge.in.tum.de/index.php/s/1JdQX7a82pILzgs>

messages. Remarkably, 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.

VIII. DISCUSSION

Implications: Developers' chat messages contain valuable knowledge about development rationale. REACT represents a step towards a more effective rationale capturing in developers' chat messages. The results of our preliminary evaluation show that rationale annotations are easy to learn and simple to apply. Furthermore, REACT can be easily integrated into modern messaging platforms and encourages collaborative capturing of rationale. 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.

Future Work: In our future work, we plan to link conversations annotated as containing rationale to issue trackers and to development artifacts that were attached to chat messages. To reduce the cognitive load of categorizing messages, we plan to implement a hybrid approach that employs machine learning for the automatic detection of messages containing rationale. A machine learning component could prompt developers—the “domain experts”—to approve the suggested categorization of the chat message if rationale is detected. 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.

Threats to Validity: We relied on an error-prone human judgment for categorizing rationale in chat messages. To mitigate this risk, messages were peer-coded by two coders and a coding guide was created. During our evaluation, we used chat messages exchanged during a university course in which students worked closely with industrial clients in settings similar to industrial settings. Moreover, they used Slack which is a common communication tool in industrial projects. We believe that this evaluation gives insights for further replications of the study in practice. However, further studies should be conducted to analyze if our findings hold in industry settings, as well as in longer lasting projects.

IX. CONCLUSION

We present REACT, a lightweight approach for capturing rationale in developers' chat messages through manual annotations. We evaluated REACT in two studies, a short- and a medium-term study. Our preliminary evaluation results show that REACT is easy to learn and simple to apply by developers. Furthermore, developers applied it collaboratively to capture rationale in their messages. REACT is a first step towards

supporting developers in capturing the rationale behind their development decisions.

ACKNOWLEDGMENTS

This work was partially supported by a PhD scholarship provided by King Saud University for Rana Alkadhi and by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES project) for Jan Ole Johanssen.

REFERENCES

- [1] H. Ajjan, R. Hartshorne, Y. Cao, and M. Rodriguez. Continuance Use Intention of Enterprise Instant Messaging: a Knowledge Management Perspective. *Behav. Inf. Technol.*, 33(7):678 – 692, 2014.
- [2] R. Alkadhi, T. Lata, E. Guzman, and B. Bruegge. Rationale in Development Chat Messages: An Exploratory Study. In *Proc. of the 14th Work. Conf. on Min. Softw. Repos.*, MSR'17, pages 436–446, 2017.
- [3] B. Bruegge, S. Krusche, and L. Alperowitz. Software Engineering Project Courses with Industrial Clients. *ACM Transactions on Computing Education*, 15(4):17:1–17:31, 2015.
- [4] J. E. Burge and D. C. Brown. Software Engineering Using RATIONale. *Journal of Systems and Software*, 81(3):395–413, 2008.
- [5] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damjanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters. *Text Processing with GATE (Version 6)*. 2011.
- [6] 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 Comput Biol*, 9(2):e1002854, 2013.
- [7] S. Dennerlein, R. Gutounig, E. Goldgruber, and S. Schweiger. Web 2.0 Messaging Tools for Knowledge Management? Exploring the Potentials of Slack. In *Proc. of the European Conference on Knowledge Management*, page 225, 2016.
- [8] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech. *Rationale Management in Software Engineering*. Springer-Verlag, 2006.
- [9] B. Eisner, T. Rocktäschel, I. Augenstein, M. Bošnjak, and S. Riedel. emoji2vec: Learning Emoji Representations from their Description. In *Proc. of the 4th International Workshop on Natural Language Processing for Social Media*, SocialNLP'16, 2016.
- [10] T.-M. Hesse, A. Kuehlwein, B. Paech, T. Roehm, and B. Bruegge. Documenting Implementation Decisions with Code Annotations. In *Proc. of the 27th International Conference on Software Engineering and Knowledge Engineering*, SEKE'15, pages 152–157, 2015.
- [11] T. M. Hesse, A. Kuehlwein, and T. Roehm. DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally. In *Proc. of the 1st International Workshop on Decision Making in Software Architecture*, MARCH'16, pages 30–37, 2016.
- [12] P. Kruchten, R. Capilla, and J. C. Dueñas. The Decision View's Role in Software Architecture Practice. *IEEE Software*, 26(2):36–42, 2009.
- [13] W. Kunz and H. Rittel. Issues as Elements of Information Systems. Working Paper 131, Institute of Urban and Regional Development, University of California, Berkeley, California, 1970.
- [14] P. Layzell, O. P. Brereton, and A. French. Supporting Collaboration in Distributed Software Engineering Teams. In *Proc. Seventh Asia-Pacific Software Engineering Conference*, APSEC 2000, pages 38–45, 2000.
- [15] J. Lee. Design Rationale Systems: Understanding the Issues. *IEEE Expert*, 12(3):78–85, 1997.
- [16] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik. Why Developers Are Slacking Off : Understanding How Software Teams Use Slack. In *Proc. of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, pages 333–336, 2016.
- [17] 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*, 77(1):66–80, 2012.
- [18] R. Lougher and T. Rodden. Supporting Long-term Collaboration in Software Maintenance. In *Proc. of the Conference on Organizational Computing Systems*, COCS '93, pages 228–238, 1993.
- [19] K. A. Neuendorf. *The Content Analysis Guidebook*. SAGE Publ., 2002.