Using Discord Conversations as Program Comprehension Aid

Marco Raglianti, Csaba Nagy, Roberto Minelli, Michele Lanza
REVEAL @ Software Institute – USI, Lugano, Switzerland

ABSTRACT

Modern communication platforms used in software development host daily conversations among developers and users about a wide range of topics pertaining to software systems, such as language features, APIs, code artifacts like classes and methods, design patterns, usage examples, code reviews, bug reporting and fixing. Discord servers are one of these virtual community hubs that have seen a steep rise in popularity, as coordination and aggregation means for communities of developers. Although Discord supports filter-based search functionalities, the sheer volume, velocity, and small granularity of single messages make it hard to find useful results, let alone complete discussions revolving around particular themes. One reason is that the concept of a discussion, which we call a *conversation*, does not exist as an explicit concept. We argue that extracting and analyzing such conversations can be used fruitfully to aid program comprehension.

We present an approach that reconstructs the conversations that take place on a software community Discord server, focusing on software-related conversations: Our approach binds the conversations to the discussed artifacts. Leveraging our approach, we built a tool that enables the interactive exploration of the conversations' contents. We illustrate its usefulness through a number of examples that highlight how the insights obtained serve as an additional form of software documentation and program comprehension aid.

CCS CONCEPTS

• **Software and its engineering → Collaboration**.

KEYWORDS

conversations, Discord, visualization

ACM Reference Format:
Marco Raglianti, Csaba Nagy, Roberto Minelli, Michele Lanza. 2022. Using Discord Conversations as Program Comprehension Aid. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3524610.3528388

1 INTRODUCTION

More than half a century ago, McLuhan's seminal book "Understanding Media" started on the premise *"the medium is the message"*: A communication medium itself, and not [only] the messages it carries, should be the primary focus of study [15].

Indeed, the very nature of a communication medium shapes the contents it supports. In recent years the many types of media that have been used by software communities have been complemented, if not replaced, by a new one: rich-media instant messaging platforms, such as Slack and Discord. These platforms are high-throughput/high-volatility virtual hubs where developers discuss daily about software. Gitter, Slack, and Discord are currently some of the most used instant messaging platforms, and have also been studied as possible data mining sources for software-related information [7, 11, 14, 19, 22, 24, 26]. They all lack semantic-based searches in the documentation corpora they provide and they offer at most basic functionalities to group messages in semantically coherent chunks, such as Q&A threads. The message granularity level is too fine-grained to efficiently help a developer quickly discriminate between the content of interest and noise. Searching answers for a specific task amounts to defining filtering criteria (*e.g.,* keywords or date intervals), retrieving messages, and manually exploring them one by one. This puts the burden of even finding the limits of the region of interest on the developer, without any form of summary to speed up the discrimination process in order to find what is needed to accomplish a task. Many GitHub projects adopted Discord as the primary communication tool among development team members and their community. Unsurprisingly, concerns about the long-term persistency of information shared on Discord are already emerging:

> *"I can't wait for the day Discord starts to cull old content [from their content delivery networks] to save server space and for so much information to just disappear.*[1]"

As developers use these platforms to share knowledge and coordinate projects [13], this integral part of the documentation landscape [21] constitutes an important form of crowd-sourced documentation for many languages, frameworks, and software projects in general [8]. Besides preserving history, an imminent need is to access and mine this documentation source. The volatility of information is in the very nature of instant messaging applications, for example, in the fact that only a handful of messages is visible on the screen simultaneously, and that older ones quickly disappear from a user's point of view. This can be a matter of seconds or minutes in a high-traffic channel. Countering the volatility could help in the fruition of information in real-time and on-demand [23].

We propose an approach to mine knowledge in Discord servers' crowd-sourced documentation to aid in program comprehension, while providing an extra layer of persistence. We reconstruct conversations and elevate them to first-class concepts. We show how discussed source code artifacts can be analyzed to extract knowledge about them as a form of ad-hoc documentation. We provide a visual representation of conversations that can be leveraged as a form of summarization to gauge important aspects that could play a role in exploration strategies. Finally, we outline the links between source code and the natural language part of a conversation.

[1] See **https://knockout.chat/thread/33251/1#post-1176126** [acc. March 30, 2022]

ICPC '22, May 16–17, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9298-3/22/05…$15.00
https://doi.org/10.1145/3524610.3528388

2 BACKGROUND

Discord – Discord is a rich-media instant messaging, *Voice over Internet Protocol* (*i.e.,* VoIP), and digital distribution platform.

Users can communicate with messages containing text, images, videos, files, embedded links, and emojis. It also supports streaming, voice chat, and video conferencing.

A *Discord server* is the basic functional unit encapsulating the concept of a community. Users in a Discord server can share messages in *text channels* and talk in *voice channels*. Channels are organized into *categories*. Many software development communities have a public Discord server, with a permanent invite link (*i.e.,* published on the main website or GitHub project page) that allows users to join the server and participate in activities.

Problem – Discord servers can host tens of thousands of users and reach throughputs of several messages per second.[2] Nevertheless, the Discord client for desktop can fit up to twenty one-line messages on an average screen, which drops to just a handful on mobile devices. People can easily miss longer conversations while they are offline: Why and when did a conversation start? Hard to tell at a glance. One needs to scroll to see all messages, and maybe realize that nothing important happened.

Solution – Reconstructing summaries of conversations to show appropriately chunked pieces of information to users. The first step in this direction is to aggregate messages and reconstruct conversations, adding meaningful information about the content (*i.e.,* discussed topics) and its context (*e.g.,* involved authors, conversation length). This higher-level representation can help discriminate conversations of interest and easily overview their messages. It can also be used for archival and retrieval.

Table 1: Statistics on the Pharo Discord server

Snapshot Date	Feb 7 2022
Activity Span	5 years 153 days
# Sent Messages	197,009
# Members	3,176
# Active Authors	1,568

Case Study – We demonstrate our approach and present interesting insights that emerged from the analysis of the Pharo Discord server, the main communication hub for daily interactions of the Pharo[3] developer community. It has more than five years of history with ~200k messages and ~1.6k message authors (see Table 1). The server has 67 channels organized into seven categories. The following section presents examples from this server, including conversations on the *roassal* channel in the *LIBRARIES* category.[4]

Table 2: Conversations on the Pharo Discord server

# Conversations	26,306
Average Conversation Span	49.8 minutes
Average Messages per Conversation	7.5
Longest Conversation # Messages	532

[2]The Programmer's Hangout Discord server has more than 110,000 users with ca. 17,000 active users a day. See **https://disboard.org/server/244230771232079873** [acc. March 30, 2022].
[3]See **https://pharo.org** [acc. March 30, 2022]
[4]Roassal is an agile interactive visualization framework for Pharo [5].

3 CONVERSATIONS

In most instant messaging applications, including Discord, the minimum unit of exchanged information is a message. We group messages that are temporally related to one another into **conversations**. The boundaries of a conversation can be defined in various ways. In our study, we use inter-message time intervals. Figure 1 shows these intervals on the Pharo Discord server.

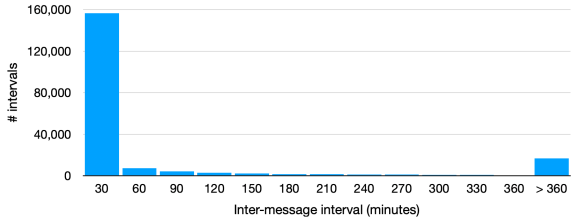


Figure 1: Time intervals between messages in all channels

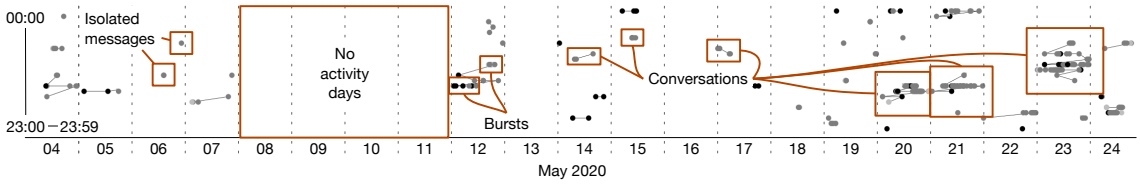
We use a two hours threshold, retaining 87% of the intervals, to reconstruct *quasi-real-time conversations*. This provides more context, so a human observer could still disambiguate and manually split without losing potentially related information if needed. The impact of the chosen threshold on the accuracy of recommended conversations about a topic remains to be evaluated.

On the Pharo Discord server, we found 26,306 conversations with an average duration of 49.8 minutes and 7.5 messages (see Table 2). Considering those with more than one message (16,482), we have an average of 11.3 messages per conversation.

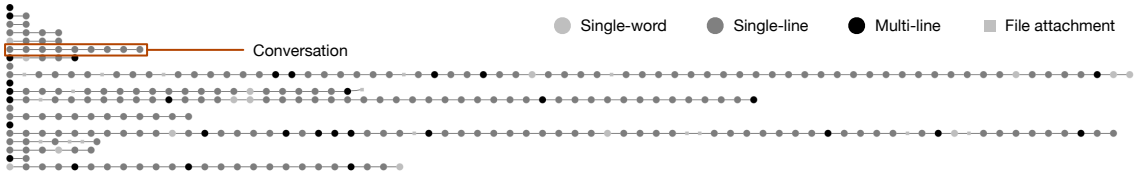
Messages Timeline and Conversation Patterns – Conversations can follow one another, they sometimes consist of sporadic isolated messages, exist as short intensive bursts or longer discussions, possibly with pauses between bursts. We can see such patterns in Figure 2a, where messages are placed in a 2D space according to their timestamp. Each day is a column in the view from left to right and each row represents an hour of the day. Messages are represented as dots and connected when they fall within the two-hour threshold. It is easy to spot clusters of overlapping messages, which represent quasi real-time conversations, with seconds or minutes between them. When messages are linked over longer distances, they indicate a semi-synchronous interaction, *i.e.,* when the inter-message interval is still below the threshold. Horizontal gaps show inactivity periods.

Linearized Conversations and Message Types – A text message can be a single emoji, a word, or sentences on multiple lines. We represent the conversations as a linear sequence of messages to provide a clearer view of the types that alternate in a typical flow.

In Figure 2b, we show examples of alternation patterns in conversation sequences of varying lengths. 89.5% of messages (176,356) are composed of a single line. This majority of one-line messages is not surprising if we consider the typical interaction mode of Discord. On a newline keystroke (*i.e.,* Return) a message is sent. Only a specific combination (*i.e.,* Shift-Return) generates a multi-line message. While the average conversation length is low (7.5 messages), there are many significant outliers (SD 19.7). For example, the longest conversation counts 532 messages. Four authors help each other with coding exercises and repository management issues.



(a) Partial message timeline where days are represented as columns and rows represent hours of the day



(b) Linearized conversations and message types (May 19–24, 2020)

Figure 2: Two visualizations of message sequences in conversations for the *roassal* channel

Further work is needed to ensure that conversations in low-traffic channels are not split. For example, timezone differences may result in responses exceeding the given threshold. In high-traffic channels, disentanglement of interleaving messages should provide better accuracy in reconstructing minimal subsets about the same topic [7]. Isolation or longer intervals between messages could indicate questions that did not receive a timely answer. These occurrences should be investigated separately. Our main focus is on (quasi) real-time conversations and the source code they discuss.

4 CONVERSATIONS ABOUT SOURCE CODE

Figure 3 shows an example of a conversation between two authors involving source code (fair usage consent has been explicitly granted by authors whose real names or pictures appear in the following examples). There are 1,485 conversations on the Pharo Discord server about source code artifacts. While this specific example is short, interestingly, the average number of messages per conversation containing code is 29, about four times the overall average (7.5). This indicates more activity around source code in the Pharo Discord server.

To investigate discussions revolving around source code snippets, we developed a custom view that shows the relevant contextual information in a condensed representation. Figure 4 shows a conversation consisting of 19 messages between three authors.

Serge Stinckwich sends a single comment suggesting what could be implemented. The conversation revolves around four code snippets (*i.e.,* green rectangles in the center). We highlighted one to show its content in a tooltip. The discussion focuses on how classes could be used to create charts: *RSChart*, *RSScatterPlot*, and *RSLinePlot* (*i.e.,* outer circle). Various methods are also visible (*i.e.,* inner circle). In particular *addPlot:* and *addDecoration:*, two methods of *RSChart*. The visualization shows how this information can grasp the topics of a conversation. This work should be extended to automate information extraction and provide a compact, meaningful representation, *e.g.,* with text summarization.

How? – We differentiate natural language and source code based on the code blocks in the messages, as they can be marked by single or triple back-ticks, like in Markdown.

In most cases, the language of a code block is specified for the syntax highlighting, and we rely on it to extract the relevant source code elements. Next, we tokenize the extracted code blocks.

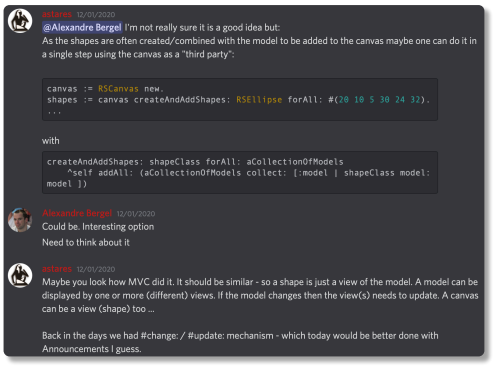


Figure 3: Example of a conversation with source code

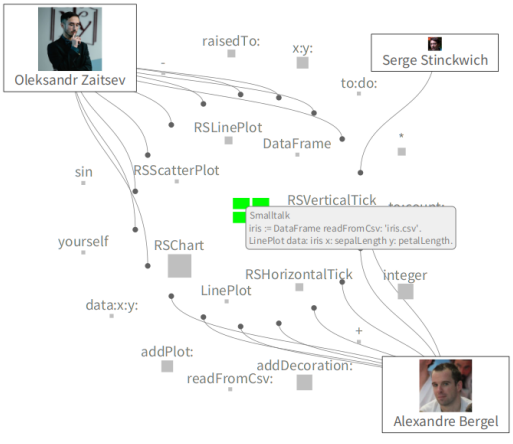
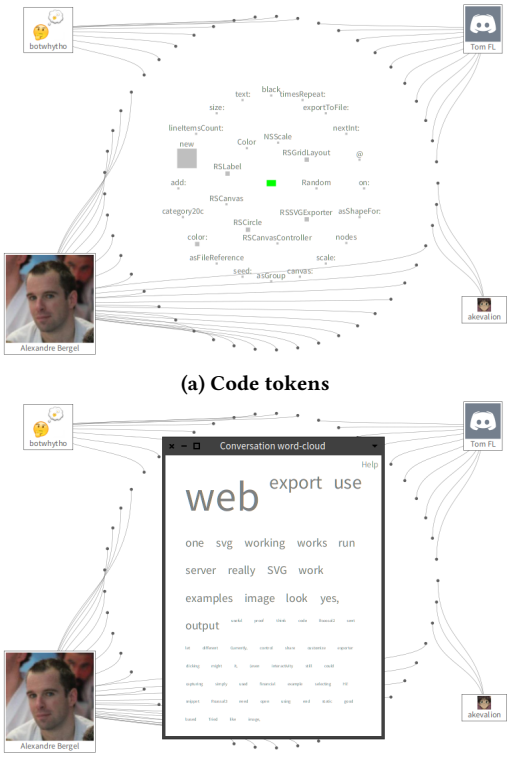


Figure 4: Complex representation of a conversation with authors, messages, code, referenced classes and methods



(a) Code tokens

(b) Natural language word-cloud

Figure 5: Example of a conversation between four authors with code and natural language related features

In the case of Smalltalk, we do a full parsing to get an abstract syntax tree. For other languages, we implemented a fallback solution based on *ANTLR* parsers. When we cannot parse the source according to a given grammar, we rely only on the tokens gathered through tokenization. Finally, we use heuristics, for example capitalized first letter, to extract class names from tokens (Figure 5a).

Why? – Our main focus is to aid program comprehension by retrieving conversations about specific language features, APIs, constructs, methods, and classes. Splitting natural language and code allows us to treat the two differently. While for the natural language content we provide word clouds to grasp the most important terms in the conversation (Figure 5b), source code classes and methods are mapped to the relevant conversations. We can retrieve, for example, all the conversations containing code about the *RSSVGExporter* class and find the conversation in Figure 5 whose main topic is about *exporting* the Roassal canvas to be *used* in the *web* (top-keywords in the word-cloud).

5 RELATED WORK

Mutton and Shihab *et al.* have investigated IRC as an instant messaging platform, where the only supported medium is text messages [16, 25]. Alkadhi *et al.* mined Atlassian HipChat [2] and IRC [3] to extract development decisions rationale from developer discussions. Ehsan *et al.* extracted discussion threads from Gitter [11] and analyzed them for development problems resolution.

Parra *et al.* manually labeled a dataset of 10,000 Gitter messages from 10 developer communities to foster future research in this field [19]. They analyzed the performances of machine and deep learning algorithms for intent prediction on their dataset [18]. Lin *et al.* explored the role of Slack in software developer teams [14]. Chatterjee *et al.* mined and disentangled conversations in Slack [7] providing a curated dataset for further research in this direction [6]. Their approach aimed at comparing the result to Q&A websites like StackOverflow, trying to exploit previous research on these platforms for a different medium. Stray *et al.* investigated the use of Slack for coordination and communication in *agile development* teams [27]. Overall, Discord remains unexplored [22].

While most of the related works focus on semantically related discussion threads, our work is more about (quasi-)real-time interactions. Conversations lasting over a day are still meaningful, but they are handled better by asynchronous media (*e.g.,* mailing lists, Q&A websites, forums), which have been extensively studied, for example by Abreu and Premraj [1], Bacchelli *et al.* [4], Guzzi *et al.* [12], and Di Sorbo *et al.* [9] for e-mails, Parnin *et al.* [17] and Ponzanelli *et al.* [20] for StackOverflow, and Di Sorbo *et al.* [10] for mobile app reviews. They dealt with similar issues in extracting information from developer communications. The different velocity, granularity, and features of modern instant messaging platforms require building higher-level, source-independent concepts.

6 CONCLUSIONS AND FUTURE WORK

Instant messaging applications are sources with high throughput and volatility. Massive adoption by developer communities makes them precious and fragile containers of ad-hoc crowd-sourced software documentation. Mining and persisting them is fundamental.

In this paper, we reconstructed conversations in Discord as a first class concept, and presented an approach based on word-clouds and source code parsing to deal with their twofold nature. The approach could be adapted to other instant messaging platforms such as Slack and Gitter, *e.g.,* by supporting different APIs for extracting messages and content. There are also limitations in the available history of some communities (*e.g.,* Slack's free tier limit to the latest 10k messages). Different platforms could provide ways to retrieve more code blocks, even outside triple back-ticks or without explicit syntax highlighting. Nevertheless, we plan to explore regular expressions and machine learning techniques to pre-process messages and identify source code fragments that could be linked to conversations. We also plan to improve disentanglement of conversations. For example, we could reduce the probability of mixing topics and unrelated features in high traffic channels by splitting and merging conversations based on topic similarity.

Overall, the resulting semantic search capabilities of our approach, augmented by the contextualized visual summarization, help in finding conversations of interest about a specific topic. This is the first step towards exploiting the potential of mining Discord for program comprehension.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the Swiss National Science Foundation and the Fonds de la Recherche Scientifique for the joint Lead Agency project "INSTINCT" (SNF Project No. 190113).

REFERENCES

[1] Roberto Abreu and Rahul Premraj. 2009. How Developer Communication Frequency Relates to Bug Introducing Changes. In *Proceedings of IWPSE-EVOL 2009 (ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution)*. ACM, 153–158.

[2] Rana Alkadhi, Teodora Lata, Emitza Guzmany, and Bernd Bruegge. 2017. Rationale in Development Chat Messages: An Exploratory Study. In *Proceedings of MSR 2017 (International Conference on Mining Software Repositories)*. IEEE/ACM, 436–446.

[3] Rana Alkadhi, Manuel Nonnenmacher, Emitza Guzman, and Bernd Bruegge. 2018. How do Developers Discuss Rationale?. In *Proceedings of SANER 2018 (International Conference on Software Analysis, Evolution and Reengineering)*. IEEE, 357–369.

[4] Alberto Bacchelli, Tommaso Dal Sasso, Marco D'Ambros, and Michele Lanza. 2012. Content Classification of Development Emails. In *Proceedings of ICSE 2012 (International Conference on Software Engineering)*. IEEE, 375–385.

[5] Alexandre Bergel. 2022. *Agile Visualization with Pharo – Crafting Interactive Visual Support Using Roassal*. Apress, Berkeley, CA.

[6] Preetha Chatterjee, Kostadin Damevski, Nicholas A. Kraft, and Lori Pollock. 2020. Software-Related Slack Chats with Disentangled Conversations. In *Proceedings of MSR 2020 (International Conference on Mining Software Repositories)*. ACM, 588–592.

[7] Preetha Chatterjee, Kostadin Damevski, Lori Pollock, Vinay Augustine, and Nicholas A Kraft. 2019. Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools. In *Proceedings of MSR 2019 (International Conference on Mining Software Repositories)*. IEEE/ACM, 490–501.

[8] Camila Mariane Costa Silva. 2020. Reusing Software Engineering Knowledge from Developer Communication. In *Proceedings of ESEC/FSE (European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 1682–1685.

[9] Andrea Di Sorbo, Sebastiano Panichella, Corrado A. Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2015. Development Emails Content Analyzer: Intention Mining in Developer Discussions. In *Proceedings of ASE 2015 (International Conference on Automated Software Engineering)*. IEEE, 12–23.

[10] Andrea Di Sorbo, Sebastiano Panichella, Corrado A. Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2021. Exploiting Natural Language Structures in Software Informal Documentation. *IEEE Transactions on Software Engineering* 47, 8 (2021), 1587–1604.

[11] Osama Ehsan, Safwat Hassan, Mariam El Mezouar, and Ying Zou. 2020. An Empirical Study of Developer Discussions in the Gitter Platform. *Transactions on Software Engineering and Methodology* 30, 1 (2020), 1–39.

[12] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie Van Deursen. 2013. Communication in Open Source Software Development Mailing Lists. In *Proceedings of MSR 2013 (Working Conference on Mining Software Repositories)*. IEEE, 277–286.

[13] Tuomas Jaanu, Maria Paasivaara, and Casper Lassenius. 2012. Near-synchronicity and Distance: Instant Messaging as a Medium for Global Software Engineering. In *Proceedings of GSE 2012 (International Conference on Global Software Engineering)*. IEEE, 149–153.

[14] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In *Proceedings of CSCW/SCC 2016 (Conference on Computer Supported Cooperative Work and Social Computing Companion)*. ACM, 333–336.

[15] Marshall McLuhan. 1964. *Understanding Media*. Gingko Press.

[16] Paul Mutton. 2004. Inferring and Visualizing Social Networks on Internet Relay Chat. In *Proceedings of IV 2004 (International Conference on Information Visualisation)*. IEEE Computer Society, 35–43.

[17] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. *Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow*. Technical Report. Georgia Institute of Technology.

[18] Esteban Parra, Mohammad Alahmadi, Ashley Ellis, and Sonia Haiduc. 2022. A Comparative Study and Analysis of Developer Communications on Slack and Gitter. *Empirical Software Engineering* 27, 2 (2022), 1–33.

[19] Esteban Parra, Ashley Ellis, and Sonia Haiduc. 2020. GitterCom: A Dataset of Open Source Developer Communications in Gitter. In *Proceedings of MSR 2020 (International Conference on Mining Software Repositories)*. ACM, 563–567.

[20] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter. In *Proceedings of MSR 2014 (Working Conference on Mining Software Repositories)*. IEEE/ACM, 102–111.

[21] Marco Raglianti. 2022. Topology of the Documentation Landscape. In *Proceedings of ICSE 2022 Companion (International Conference on Software Engineering Companion)*. ACM.

[22] Marco Raglianti, Roberto Minelli, Csaba Nagy, and Michele Lanza. 2021. Visualizing Discord Servers. In *Proceedings of VISSOFT 2021 (Working Conference on Software Visualization)*. IEEE, 150–154.

[23] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. 2017. On-demand Developer Documentation. In *Proceedings of ICSME 2017 (International Conference on Software Maintenance and Evolution)*. IEEE, 479–483.

[24] Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyou Jiang, Nan Niu, and Qing Wang. 2021. A First Look at Developers' Live Chat on Gitter. In *Proceedings of ESEC/FSE 2021 (European Software Engineering Conference and Symposium on the Foundations of Software Engineering)*. ACM, 391–403.

[25] Emad Shihab, Zhen Ming Jiang, and Ahmed E Hassan. 2009. On the Use of Internet Relay Chat (IRC) Meetings by Developers of the GNOME GTK+ Project. In *Proceedings of MSR 2009 (International Working Conference on Mining Software Repositories)*. IEEE, 107–110.

[26] Viktoria Stray and Nils Brede Moe. 2020. Understanding Coordination in Global Software Engineering: A Mixed-Methods Study on the use of Meetings and Slack. *Journal of Systems and Software* 170 (2020), 110717.

[27] Viktoria Stray, Nils Brede Moe, and Mehdi Noroozi. 2019. Slack Me If You Can! Using Enterprise Social Networking Tools in Virtual Agile Teams. In *Proceedings of ICGSE 2019 (International Conference on Global Software Engineering)*. ACM/IEEE, 111–121.