DiscOrDance: Visualizing Software Developers Communities on Discord

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

***Abstract*—New communication platforms have emerged to support developers in finding and creating the knowledge they need for program comprehension, maintenance, and evolution. Instant messaging applications are supplanting developer mailing lists in collaborative development toolchains. These applications provide a new medium, supporting faster and richer communication (*e.g.,* embedded previews, images, files, videos). Research so far focused on extracting information from these platforms, but there is a lack of tools to visually and interactively explore them.**

We present DISCORDANCE, a tool for the interactive visual exploration of the complete message history of a Discord server. We show how three categories of views elicit insights on aspects of the structure, members, and software related content of a Discord server. We demonstrate use cases of DISCORDANCE to support software maintenance and evolution activities on an active software developer community, the Pharo Discord server.

Demo video: `https://youtu.be/eYCLGWwM9HY`

Tool homepage: `https://discordance.si.usi.ch`

***Index Terms*—software communities, collaborative development, visualization, Discord**

I. INTRODUCTION

In the context of collaborative development, toolchains fill the role of supporting inter-developer communication with different software solutions [1], [2]. Instant messaging communication platforms have replaced traditional media (*e.g.,* developer mailing lists). Gitter [3]–[6], Slack [6]–[9], and Discord [10]–[12] are examples of platforms where developers discuss project- or language-related issues, ask for support with software design or code comprehension, request reviews, and share source code snippets.

All these activities can be seen as a source of *informal* documentation where, above all, design, explanatory, and structural information is discussed [8], and therefore represent new possibilities for maintenance and evolution activities. Documentation generated on instant messaging platforms share many commonalities with the one present in social networks [13], and forum-like Q&A websites (*e.g.,* StackOverflow [14]), in particular, with respect to crowd-sourcing aspects [15].

What sets instant messaging apart is the throughput of the channel and the volatility of its content. This is even more evident in the lack of tools to grasp the structure of a Discord server or to retrieve specific information. Discord, for example, supports word-based search in the whole history of a channel, with filtering criteria such as date, sender, and presence of embedded links or images. Although this might work for a quick search in a low-traffic channel, it does not scale up to generic information retrieval on large servers. Dealing with high throughput is essentially left to manual labor by users.

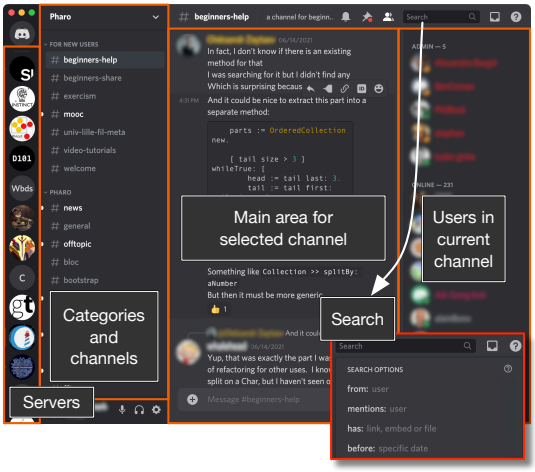

Fig. 1: A Screenshot of the Discord Desktop Application

Fig. 1 shows the user interface (UI) of the Discord desktop application. Discord is available for desktop (*i.e.,* Linux, Windows, macOS) and smartphones (*i.e.,* Android and iOS).

The desktop application is organized in columns containing the servers list, categories and channels, the messages of the selected channel, and the member list. The mobile UI features the same components but with a layout more suitable for small screens. Discord servers may contain tens of thousands of users and reach a throughput of several messages per second.

We developed DISCORDANCE, a tool to overcome issues with high throughput and volatility of developer communications on Discord. DISCORDANCE enables visualization and live exploration of the complete message history of a Discord server. Its domain model is built around the source code and related conversations to support software maintenance and evolution activities. The model is represented as a graph, it allows to extract and persist information in a form that better suits analysis and retrieval needs. Besides six pre-defined views, DISCORDANCE can be easily extended with custom view specifications to elicit further insights. Metrics of nodes and edges (*i.e.,* entities) can be mapped to visual properties of the glyphs representing them (*e.g.,* rectangles, circles, lines).

Structure of the Paper. In Section II we present the user interface and the architecture of DISCORDANCE and we explain how to define custom view specs. Section III illustrates the usage of DISCORDANCE on a specific software community Discord server. Section IV outlines the related work and Section V concludes our paper.

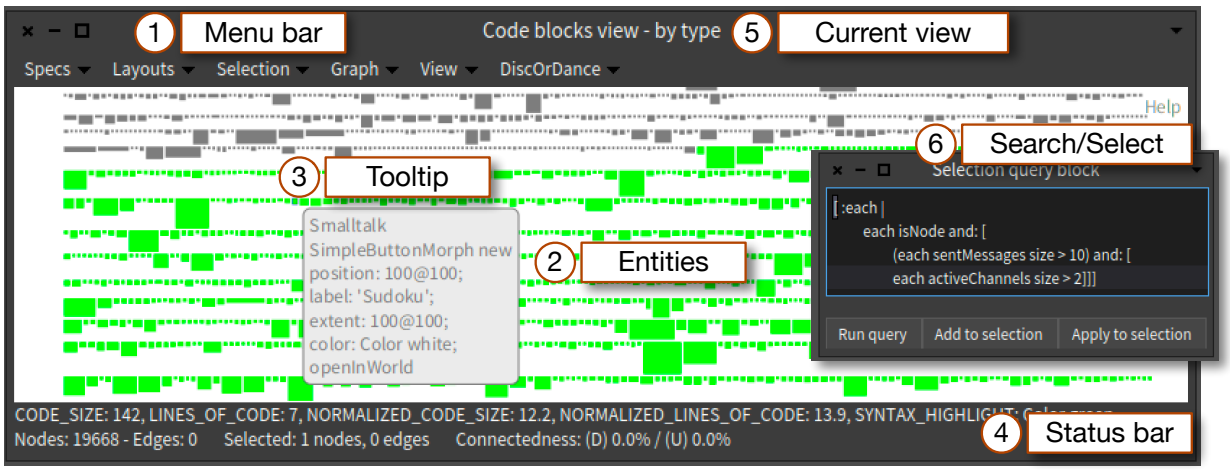

Fig. 2: The User Interface of DiscOrDance

II. DiscOrDance

We developed DiscOrDance to mine the message history of a Discord[1] server. DiscOrDance is implemented in Pharo[2] and it is composed of a scraper bot, an object-oriented domain model of a Discord server, and a user interface for interactive visualization and exploration of a server instance.

Fig. 2 shows a screenshot of the user interface of DiscOrDance. The menu bar ① allows to select pre-defined view specs, change the layout of the current view, select nodes and edges and spawn a new view from the current selection, inspect the current graph, and handle visibility of specific elements. An entry also collects diverse custom operations related to the model (*e.g.,* cleaning, saving), metrics, and metric normalization. Entities ② are shown in the main canvas and can support on-hover tooltips ③ for displaying relevant information about them. DiscOrDance also supports contextual menu interactions on the entities (*e.g.,* right click on a message node → *Open in Discord*). A status bar ④ reports information about hovered entities and statistics for the graph in the current view ⑤. Finally, a selection window ⑥ allows to filter entities and select them programmatically.

DiscOrDance does not need any configuration and works out-of-the-box, but it can be extended by implementing new metrics and view specs (see Section II-B and Section II-C).

A. Architecture

Fig. 3 summarizes the architecture of DiscOrDance.

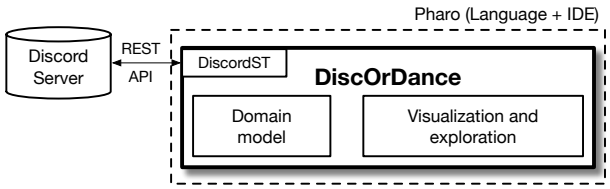

Fig. 3: The Architecture of DiscOrDance

[1]See **https://discord.com** [accessed August 5, 2022]
[2]See **https://pharo.org** [accessed August 5, 2022]

DiscordST [16] is a client for Discord that includes a subset of the public Discord REST API. The bot must be added by the server administrators with reading permissions (*read_message_history*). Then it can scrape the structure and content of a server and store it locally. It retrieves channels and their messages, and builds the model (*e.g.,* adds links for mentions between authors).

The scraping process attempts to retrieve all the entities of interest (*e.g.,* authors, messages, attachments, code) and populates the internal domain model accordingly (*i.e.,* nodes for messages, edges for mentions). After the scraping is complete, the graph of all nodes and edges is created and the resulting instance can be explored in a new window, according to a selected pre-defined view specification (Fig. 2).

B. Metrics

DiscOrDance implements two types of metrics: Color and numeric. Color metrics map categorical attributes of the entity (*e.g.,* activity status) to an output color. Numeric metrics correspond to numeric attributes of the entity (*e.g.,* LOC of a code snippet). Table I shows the implemented metrics and an example of their meaning. Every numeric metric has a normalized version where the minimum and maximum values for the normalization can be provided.

TABLE I: Metrics, Types, and Example Values

Metric	Type	Example(s)
Active Authors	Numeric	# authors active in a channel
Active Channels	Numeric	# channels in which author is active
Active Status	Color	Server membership status
Activity Span	Numeric	# days of activity in a channel
Channel Type	Color	Voice or Text
Class References	Numeric	# references of a class
Code Size	Numeric	Size of code in bytes
Contained Messages	Numeric	# messages contained in a category
LOC	Numeric	# lines of code
Max Daily Messages	Numeric	Maximum # sent messages per day
Mentions	Numeric	# mentions of other users
Sent Messages	Numeric	# messages sent in a channel
Syntax Highlight	Color	Language syntax highlighting

C. View Specs

Views are the way entities of the graph are displayed in DISCORDANCE. Views are created following templates called ***view specs***. A view spec is a specification of entities, a layout, glyph mappings, metric mappings, a sorting block, and a filtering block (all attributes are optional).

Entities: Limiting the scope to specific entity types allows to filter out all nodes and edges except those of interest.

Layout: Takes care of positioning entities on the canvas (unless a metric is mapped to the glyph position).

Glyph Mappings: Define associations between entity types and their visual representation. Custom glyphs are defined for nodes and edges. Subtype mapping is supported.

Metric Mappings: Each entity type can have associations between metrics of the entity (*e.g.,* # of references of a class) and visual attributes of the glyph (*e.g.,* size, color, position).

Sorting Block: A code block that provides the ordering of entities based on their properties. Layouts considering ordering (*e.g.,* flow layout) will position glyphs in the resulting order.

Filtering Block: A code block that filters specific entities based on their properties instead of their type. For example, a view considering only recent messages can show all message entities and filter according to sending date.

Although custom view spec definitions are possible (Fig. 4), DISCORDANCE offers six pre-defined views to explore different aspects of a Discord server: server structure (*Channel Activity View* and *Channel Activity Timeline View*), message authors (*Author Activity Status View* and *Author Activity Sparkline View*), and source code shared among members of the community (*Code Blocks View* and *Class References View*). These views are also implemented through the view specs mechanism and described in detail in our previous work [10].

```
self name: 'Class links view';
   layoutClass: RSRadialTreeLayout;
   entityClasses:
      { DDClassElement . DDClassNode . DDCodeReferenceEdge } asSet;
   glyphMappings: {
      DDClassElement -> DDSmallGlyph .
      DDClassNode -> DDNodeGlyph .
      DDCodeReferenceEdge -> VZEdgeGlyph }.
```

Fig. 4: Example of a View Spec Definition.

D. Manual Inspection

The Pharo IDE provides an inspector for objects, supporting live navigation of object instances. Through the inspector we can explore the object-oriented model by navigating the properties and content of entities in the graph. The inspector also supports custom code execution in a playground to quickly try ideas and test hypotheses directly on the live model.

In Fig. 5, we show an example of a code node glyph with its instance variables. The glyph is mapped to a code entity. Navigating the link to the container message allows us to see where that code has been referenced in a conversation and retrieve the actual message entity for further exploration. The Pharo code at the bottom can be evaluated in the context of the current object, for example allowing to open the original Discord message in the browser.

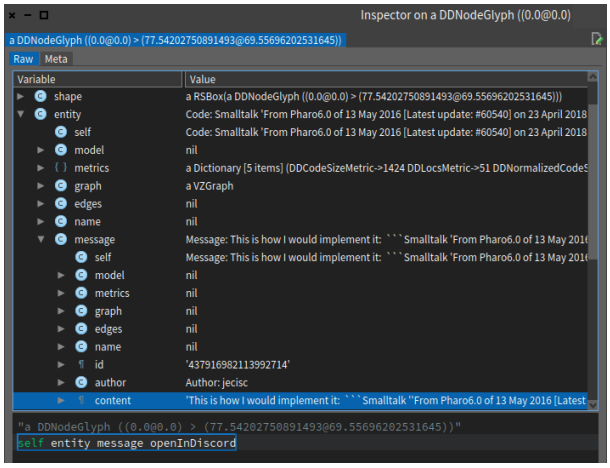

Fig. 5: Pharo Inspector and an Example Code Node Glyph

III. PHARO DISCORD SERVER

We present insights from the Pharo community Discord server elicited by channel, author, and source code views.

Channel Views. This view provides a structural overview of a server. Categories and channels are depicted with shapes. Edges between channels and categories represent containment relationships. The size of glyphs is proportional to the number of messages, pinpointing the most active channels. Fig. 6 shows the Channel Activity View of the *"Libraries"* category.

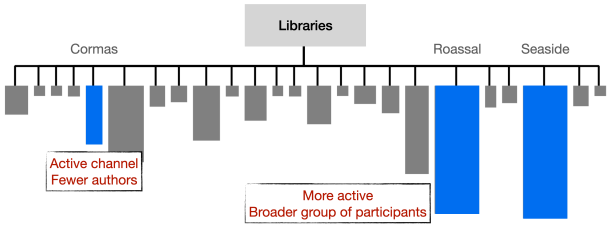

Fig. 6: Channel Activity View of the "Libraries" Category

Its channels are different in terms of activity and the number of members actively participating. By hovering over the *"Cormas"* channel, we can see in the status bar that it has 30 active authors who contributed about 3,100 messages. Channels *"Roassal"* and *"Seaside"* have a broader number of participants (*i.e.,* 154 and 153 respectively) with a higher overall activity of more than 7,500 messages each.

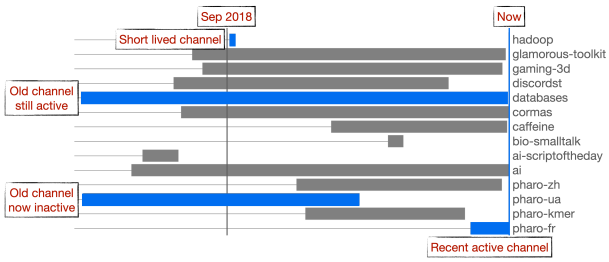

Fig. 7: Channel Activity Timeline View of 14 Channels

Fig. 7 shows the Channel Activity Timeline View of fourteen channels, highlighting their different types. Either by manual inspection (Section II-D) or with the information provided in the status bar (Fig. 2 – ④), we can see that *"hadoop"* has been a short-lived channel while *"databases"* is old and still active (∼4,200 messages since Apr 2017). The Ukrainian Pharo community has been active between Apr 2017 and Dec 2019. French-speaking Pharo users received a dedicated channel in 2021: *"Pharo en français c'est ici ☺"*.

Author Views. Author views map message authors to box-shaped glyphs. Edges connect authors based on mentioning tags (*i.e.,* @*username*). Mapping the number of messages sent or the number of active channels to the glyph size allows for a visualization of the most (or least) active authors. An example view spec in this category is the *Author Activity Status View*.

By combining the selection query support with the Author Activity Status View we can quickly select 242 authors who posted a single message to the server, as depicted in Fig. 8.

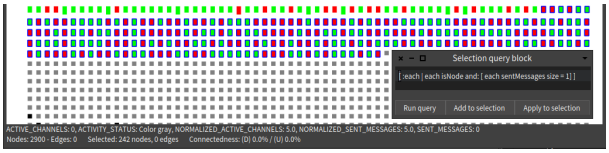

Fig. 8: Selection Query on Author Activity View

Half of them are not part of the Pharo Discord community anymore (*i.e.,* they quit the server, highlighted in red). It remains to be investigated when they have been active and what is the nature of their only message. Answering these questions could help better address newcomers in the community. By selecting ex-members with a low number of messages, and manually inspecting their latest conversations, we could better understand why they abandoned the server. All these operations are already supported by DISCORDANCE.

Source Code Views. *Code Blocks View* and *Class References View* are examples of source code views focusing on domain-specific content (*e.g.,* mentioned classes) to provide insights on language features discussed in the community.

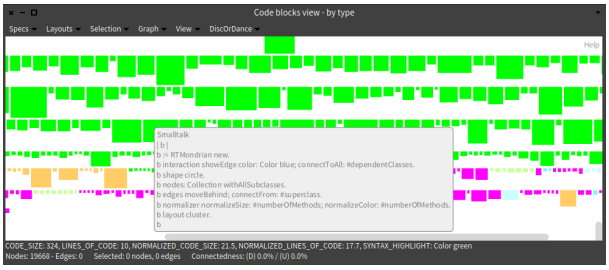

Fig. 9: Code Blocks View Showing Code with Tooltips

In the Code Blocks View, such as the one depicted in Fig. 9, the size of a code block allows distinguishing between short snippets and longer code. We can hover over different blocks to see their content. For example, one can find unformatted long lines of copy-pasted code by looking at the "wide" blocks, as these blocks have many characters in a few lines. In this view spec, the *syntax highlight* metric is mapped to the *color* attribute of code block entities. This allows visually estimating the ratio of source code snippets for each language. Length and language are only two examples, other source code metrics can be implemented and used in a Code Blocks View.

Class References View is an example of a feature-specific view that has been implemented by extending the domain model. Code blocks are parsed to find class instances and new nodes are added to the graph for each class mention we find. Nodes are connected to the container message with code reference edges. The resulting view spec maps the number of mentions to the glyph size and uses simple labeled node glyphs. It can help us pinpoint the most discussed classes and find where their documentation could be improved.

IV. RELATED WORK

Stephany *et al.* visually analyzed mailing lists and source code repositories of medium to large developer communities with their tool, MAISPION [17]. In the domain of synchronous communication, Mutton presented visualizations of IRC channels and their users [18]. Shihab *et al.* mined IRC meeting logs of the GNOME GTK+ developers community [19]. More recently, Foundjem and Adams mined IRC chat logs of the team responsible for OpenStack ecosystem releases [20]. Rich media platforms have also been investigated. Developer discussions on Gitter have been successfully mined and used for knowledge extraction [3]–[5], [21]. In particular, the tight coupling with the GitHub cloud-based repository allowed linking instant messaging discussions to projects' issues. The use of Slack by developers has been investigated with respect to workflows and purpose [7], as a possible source for Q&A structured documentation [8], and as a coordination tool to reduce communication delays [9]. Parra *et al.* compared developer communications on Slack and Gitter [6]. Discord, which recently gained popularity among developers, has been analyzed in a case study on the Pharo community [10], to extract conversations for program comprehension [11], and as a data source to be mined for research in Software Engineering [12].

V. CONCLUSION

DISCORDANCE allows to explore the history of a Discord server. We described its user interface, architecture, and key implementation concepts. We defined view specs and metrics we used to provide insightful visualization of a server. Finally, we showed how we used it to investigate the Pharo Discord server and extract knowledge about its structure and contents. DISCORDANCE gives access to source code and conversations surrounding it. It is another step towards supporting developers in maintenance and evolution tasks by leveraging yet unexplored documentation in instant messages.

ACKNOWLEDGMENT

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

REFERENCES

[1] J. A. Teixeira and H. Karsten, "Managing to release early, often and on time in the OpenStack software ecosystem," *Journal of Internet Services and Applications*, vol. 10, no. 1, p. 7, 2019.

[2] G. Poo-Caamaño, E. Knauss, L. Singer, and D. M. German, "Herding cats in a FOSS ecosystem: A tale of communication and coordination for release management," *Journal of Internet Services and Applications*, vol. 8, no. 1, pp. 1–24, 2017.

[3] E. Parra, A. Ellis, and S. Haiduc, "GitterCom: A dataset of Open Source developer communications in Gitter," in *Proceedings of MSR 2020 (International Conference on Mining Software Repositories)*. ACM, 2020, pp. 563–567.

[4] O. Ehsan, S. Hassan, M. E. Mezouar, and Y. Zou, "An empirical study of developer discussions in the Gitter platform," *Transactions on Software Engineering and Methodology*, vol. 30, no. 1, pp. 1–39, 2020.

[5] L. Shi, X. Chen, Y. Yang, H. Jiang, Z. Jiang, N. Niu, and Q. Wang, "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, 2021, pp. 391–403.

[6] E. Parra, M. Alahmadi, A. Ellis, and S. Haiduc, "A comparative study and analysis of developer communications on Slack and Gitter," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–33, 2022.

[7] B. Lin, A. Zagalsky, M.-A. Storey, and A. Serebrenik, "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, 2016, pp. 333–336.

[8] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N. A. Kraft, "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, 2019, pp. 490–501.

[9] V. Stray and N. B. Moe, "Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack," *Journal of Systems and Software*, vol. 170, p. 110717, 2020.

[10] M. Raglianti, R. Minelli, C. Nagy, and M. Lanza, "Visualizing Discord servers," in *Proceedings of VISSOFT 2021 (Working Conference on Software Visualization)*. IEEE, 2021, pp. 150–154.

[11] M. Raglianti, C. Nagy, R. Minelli, and M. Lanza, "Using Discord conversations as program comprehension aid," in *Proceedings of ICPC 2022 (International Conference on Program Comprehension)*. ACM, 2022.

[12] K. M. Subash, L. P. Kumar, S. L. Vadlamani, P. Chatterjee, and O. Baysal, "DISCO: A dataset of Discord chat conversations for software engineering research," in *Proceedings of MSR 2022 (International Conference on Mining Software Repositories)*. ACM, 2022.

[13] Y. Tian, P. Achananuparp, I. N. Lubis, D. Lo, and E.-P. Lim, "What does software engineering community microblog about?" in *Proceedings of MSR 2012 (Working Conference on Mining Software Repositories)*. IEEE, 2012, pp. 247–250.

[14] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "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, 2014, pp. 102–111.

[15] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," Georgia Institute of Technology, Tech. Rep., 2012.

[16] J. Cerezo, J. Kubelka, R. Robbes, and A. Bergel, "Building an expert recommender chatbot," in *Proceedings of BotSE 2019 (International Workshop on Bots in Software Engineering)*. IEEE/ACM, 2019, pp. 59–63.

[17] F. Stephany, T. Mens, and T. Gîrba, "Maispion: A tool for analysing and visualising open source software developer communities," in *Proceedings of IWST 2009 (International Workshop on Smalltalk Technologies)*. ACM, 2009, pp. 50–57.

[18] P. Mutton, "Inferring and visualizing social networks on internet relay chat," in *Proceedings of IV 2004 (International Conference on Information Visualisation)*. IEEE Computer Society, 2004, pp. 35–43.

[19] 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 MSR 2009 (International Working Conference on Mining Software Repositories)*. IEEE, 2009, pp. 107–110.

[20] A. Foundjem and B. Adams, "Release synchronization in software ecosystems," *Empirical Software Engineering*, vol. 26, no. 3, p. 34, 2021.

[21] H. Sahar, A. Hindle, and C.-P. Bezemer, "How are issue reports discussed in Gitter chat rooms?" *Journal of Systems and Software*, vol. 172, p. 110852, 2021.