EGAD: A moldable tool for GitHub Action analysis

Pablo Valenzuela-Toledo[1], Alexandre Bergel[2], Timo Kehrer[1], Oscar Nierstrasz[3]
[1]*Software Engineering Group, University of Bern*, Bern, Switzerland
[2]*RelationalAI*, Bern, Switzerland
[3]*feenk GmbH*, Wabern, Switzerland

***Abstract*—GitHub Actions (GA) enjoy increasing popularity in many software development projects as a means to automate repetitive software engineering tasks by enabling programmable event-driven workflows. Researchers typically analyze GA at the raw data level using batch tools to mine and analyze actions, jobs, and steps within GA workflows. Although this approach is widely applicable, it ignores the specific context of the GA workflow domain. Consequently, researchers do not reason directly about the domain abstractions.**

We present our preliminary steps in building EGAD (Explorable GitHub Action Domain Model), a moldable domain-specific tool to depict and analyze detailed GA workflow data. EGAD consists of an explorable domain model of GA workflows augmented with custom, domain-specific views, and live narratives. We illustrate EGAD in action using it to explore "sticky commits" in GitHub repositories.

***Index Terms*—GitHub Actions, software evolution, moldable development**

I. Introduction

GitHub Actions (GA) have been increasingly adopted in software development projects [1]. Although GA was publicly released only in November 2019, it is already the dominant continuous integration service on GitHub [2]. GA allows actions to be triggered automatically based on events such as commits, comments, issues, pull requests, and schedules. For example, GA enables the automation of testing, code reviews, continuous integration, communication, dependency management, and security monitoring.

GA supports automation through *workflows*. Workflows are specified in YAML files that describe the actions to be triggered by specific events (*e.g.*, the automatic treatment of pull requests [3]). However, we observe that developers make frequent mistakes in implementing the workflows and commit multiple changes before running them correctly [4]. The only tool they have for specifying workflows is the text editor, which does not provide features for authoring, analyzing or debugging. Therefore, developers are forced to validate their workflows by pushing multiple versions to the repository.

To conduct studies into GA, researchers use batch (non-interactive) tools to mine and analyze raw data of interest [1], [3], [4] (see section III). Although this approach is widely applicable, it ignores the contextual nature of the mining and does not provide explorable domain models [5].

To support researchers in studying GA workflows, we present **E**xplorable **G**itHub **A**ction **D**omain Model (EGAD), a domain-specific tool to depict and analyze GA workflows and their evolution. EGAD offers an explorable domain model, custom views and live narratives that enable researchers to inspect and analyze workflows.

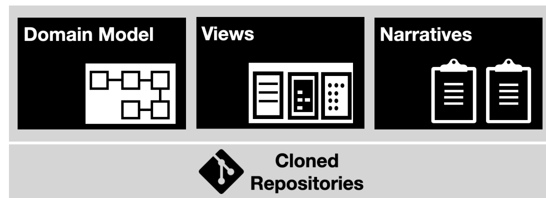

Fig. 1. Architecture overview of EGAD.

EGAD provides an approach to answer specific questions about GA workflows. The procedure consists of four steps, namely: (i) goal definition, (ii) wrapping the GA workflow data in domain model entities, (iii) exploring the domain model, and (iv) interpreting the results.

We give an overview of EGAD's architecture in section II, and illustrate its use by investigating so-called *sticky commits* in GitHub software repositories in section III.

II. Tool Architecture and Implementation

Figure 1 depicts the architecture of EGAD which consists of (i) an explorable domain model, (ii) custom views, and (iii) live narratives that access the raw repository data.

A. Domain model

The domain model wraps the GA workflow data. The model includes: (i) the workflow history, including all the commits associated with a GA workflow, and (ii) the representation of the workflow, including events, jobs and steps.

A class diagram focusing on the most important entities is shown in Figure 2. A `WEHistories` instance contains a collection of `WEHistory` objects, each of which represents the history of workflows of a dedicated project. A `WEFileCommit` object represents a commit revising a dedicated workflow by a new version (`WEWorkflow`), which in turn consists of events, jobs and steps.

As we shall see in the next section, we populate an explorable domain model by extracting data from GA workflow files obtained from cloned software repositories.

B. Custom views

Custom views enable the exploration and navigation of the domain model from multiple perspectives. Instead of presenting the data generically, custom views provide critical insights into the domain model, and make it possible to pose

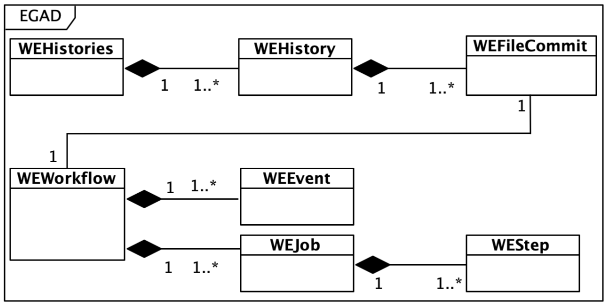

Fig. 2. Representation of GA workflows and their evolution.

questions and explore hypotheses. In order to answer domain-specific questions, the views can be easily extended. This feature allows the tool to be adapted to face new scenarios to explore. We shall see several examples in the next section.

C. Narratives

Narratives link documentation, source code, and running objects to specific questions or hypotheses of the domain model. We use narratives to explain use cases, scenarios, and features of the GA domain. The creation of narratives is accomplished by (i) telling stories, or (ii) following a path of custom views.

D. EGAD implementation

EGAD is developed on top of the Glamorous Toolkit[1] (GT), a moldable environment for building live and explorable domain models using hyperlinked notebooks, live domain objects, and customizable, domain-specific views [6]. GT includes extensive live documentation detailing how to customize it for a given application domain. GT is built on Pharo,[2] a modern, open-source Smalltalk environment.

III. EGAD IN ACTION

In this demonstration, we will work with a dataset comprising selected GitHub repositories that (i) currently use GA and contain at least one workflow file in the `.github/workflow` directory, (ii) have been created after `2019-01-11` (GA official release date) and before `2022-12-14`, and (iii) have at least ten stars and 500 commits. We used the GitHub Search tool to select repositories (excluding forks) meeting these criteria [7]. We cloned the repositories on `2022-12-14`. In this paper, we limit our dataset to the first 50 repositories returned by the query.

For the sake of analysis and exploration, EGAD supports a four-step process: (i) goal definition, (ii) wrapping the GA workflow data in domain model entities, (iii) inspecting the domain model, and (iv) interpreting the results.

We illustrate the EGAD approach in the context of sticky commits. Figure 3 introduces the sticky commits narrative in a single notebook page.

[1]https://gtoolkit.com
[2]https://pharo.org

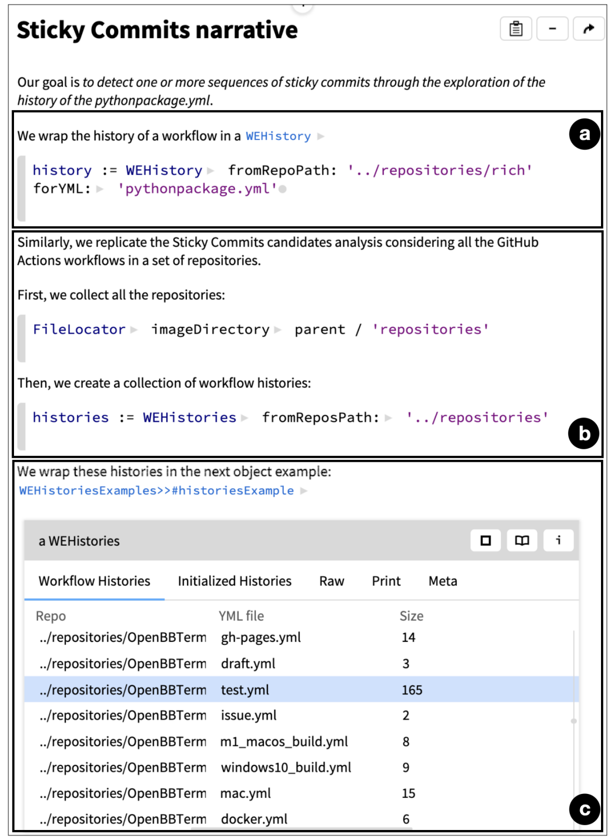

Fig. 3. The sticky commits narrative.

Sticky commits consist of a sequence of commits performed continuously, one after another. The commits are made by the same developer, and are intended to correct errors in workflow specifications made in the previous commit. Sticky commits suggest that the developer is having difficulty correctly implementing some GA behavior. We call them “sticky commits” because they are continuously pushed, as if they were sticky.

As an example of sticky commits, let’s consider the history of the GA workflow file `pythonpackage.yml` [8]. This file is part of the `Rich` GitHub repository [9]. The history of this file consists of seventy commits.

A. Goal definition

Our goal is *to detect one or more sequences of sticky commits through the exploration of the history of the pythonpackage.yml file*.

B. Wrapping the GA workflow data in the domain model

We wrap the GA workflow data in a `WEHistory` object. Figure 3a shows the associated code snippet.

Executing this snippet yields a `WEHistory` object that we can explore.

The *Workflows* view of this object lists the GA workflow files of the `Rich` repository (Figure 4a). Each custom view, such as this one, has been developed in just a few lines of code, to offer a new, dedicated perspective of the domain objects.

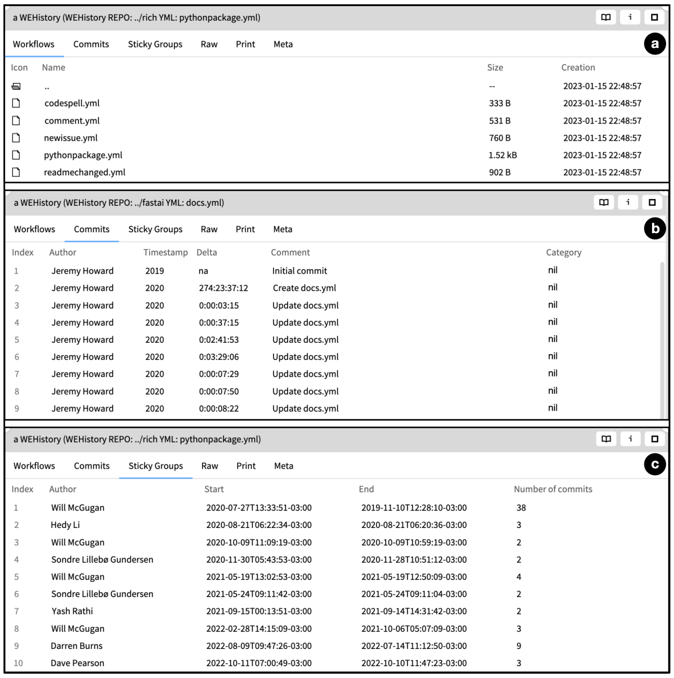

Fig. 4. Different views of a `WEHistory` object: (a) Workflow view, (b) Commits view, (c) Sticky Group view.

The *Commits* view lists all the workflow commits in chronological order (Figure 4b), of which only the first nine commits are shown in the figure.

We now move to the *Sticky Groups* view, which lists groups of sticky commits (Figure 4c). These are computed as a series of commits that are (i) related to GA workflows, (ii) consecutive, and (iii) performed by the same author.

C. Inspecting the domain model

To analyse the presence of sticky commits, we inspect the first group, which has 38 commits (Figure 4c). As a result, we get a `WEStickyGroup` object (Figure 5) listing all the commits within the group.

Inspecting the `WEStickyGroup` object (Figure 5), we notice in the *Duration* column that the time between commits 4 through 9 is always less than 8 minutes. This is of interest since we have a sequence of commits performed repetitively in a short period of time.

To investigate this sequence of commits in depth, we navigate through each commit and review the changes. Each commit is a `WEFileCommit` object (Figure 6). This object shows the code that was modified (the *Diff* view), highlighting added and deleting lines of code in green and red, respectively.

D. Interpreting the results

Reviewing each commit modification, we realize that these commits correspond to minor modifications that could have been handled in a single commit. To record this particular behavior, we use the editable *Category* column, entering "yes", which means there are sticky commit candidates that could be empirically validated (Figure 5). These data can be exported to conduct further analysis.

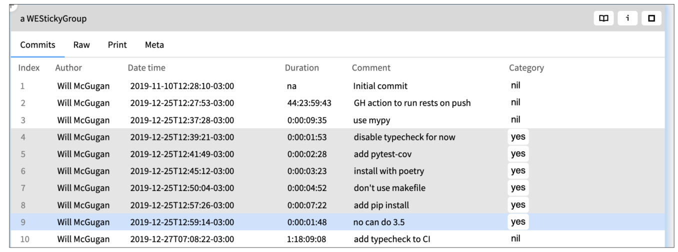

Fig. 5. *Commit* view of the `WEStickyGroup` object.

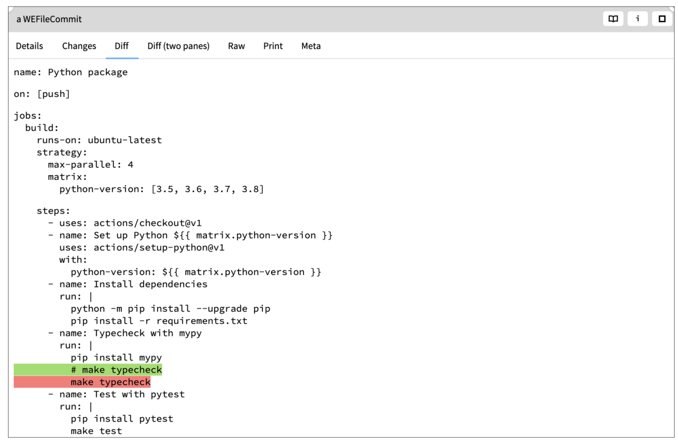

Fig. 6. *Diff* view of the `WEFileCommit` object.

E. What's next?

Are sticky commits pervasive in workflows of other repositories? To answer this question, we replicate the analysis considering all the GA workflows in a set of repositories. A broader view of the presence of sticky commits in GA workflows may guide the refinement of the heuristics for computing sticky groups.

We replicate the analysis by wrapping the histories of GA workflows from multiple repositories in the `WEHistories` object (Figure 3b). To make this behavior easier to access, we create the `historiesExample` example object in our narrative (Figure 3c). The `historiesExample` object lists all the histories from all the workflows of the 50 repositories.

To investigate the presence of sticky commits in these histories we inspect the `test.yml` workflow file from the `OpenBBTerminal` repository. The history of this file has 165 `WEFileCommit` (Figure 3c) and 32 `WEStickyGroup` objects (Figure 7a).

Then, we inspect the `WEStickyGroup` with index 14 (Figure 7a), which contains 11 `WEFileCommit` objects (Figure 7b). We notice in the *Duration* column that the time between commits 3 through 5, and between commits 7 through 11 is always less than 8 minutes. As with our initial exploration, this is of interest since we have a sequence of sticky commit candidates.

To address our finding, we navigate the last five `WEFileCommit` objects of the list using the *Diff* view (Figure 7c). We observe a sequence of commits performing

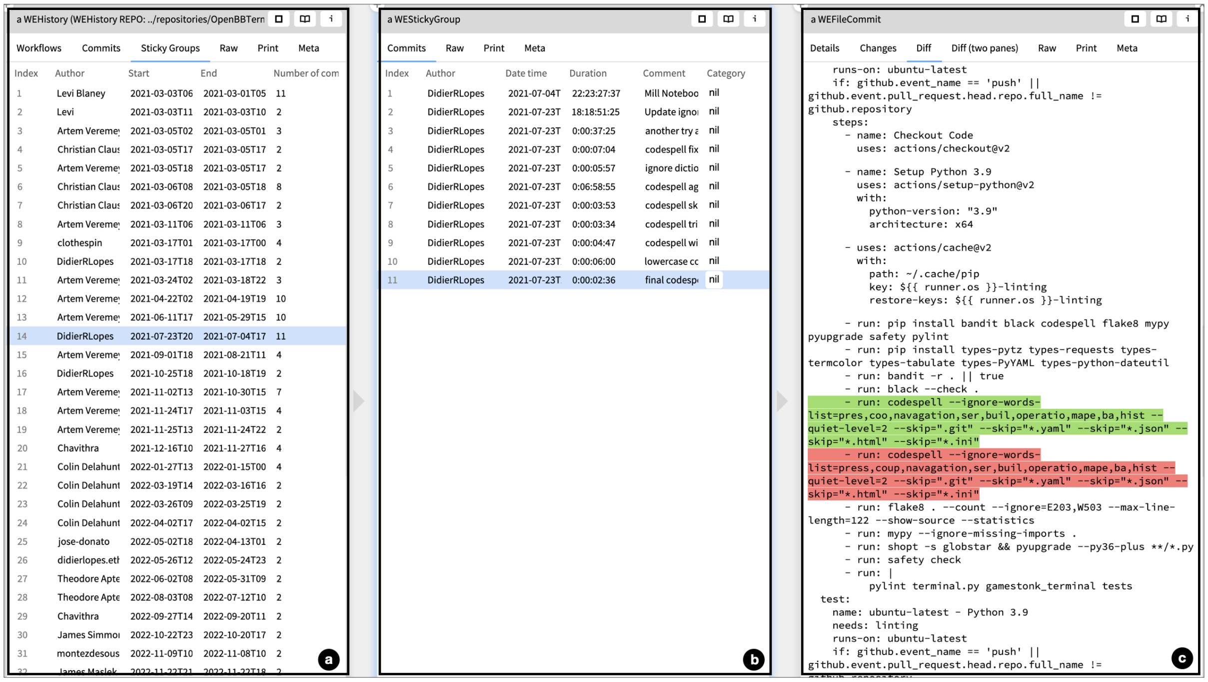

Fig. 7. Following a path of views for `WEStickyGroup` object inspection.

minor modifications one after another, within a short period of time. Although not a confirmation, this finding suggests that the presence of sticky commits is plausible in the context of our narrative.

Then, as a next step, we could be interested in inspecting the workflow structure in detail and, for example, being able to check the implementation of events and jobs in the workflow. This is a feature that will be implemented in the next release of the tool.

We provide a full replication package [10] which includes documentation, and an example dataset for running the tool. A stable version of EGAD is hosted on Zenodo [11].

IV. Related Work

Seminal works from earlier days of mining software repositories relied on artifacts that have been designed to answer specific research questions [12], [13], while the reusability of tools has been discussed largely at an architectural level [14], [15], proposing a blackboard architectural style which we adopt in EGAD.

To date, the well-known GHTorrent [16], [17] and Boa [18] provide queryable offline mirrors to deal with massive amounts of GitHub data. Other tools aim at helping researchers in curating their own datasets [19], providing facilities for sampling [7], filtering [20], or finding similar software projects [21]. Besides, there are initiatives for providing archives of these data, fostering long-term preservation and reproducibility [22], [23]. All of these efforts are orthogonal to ours in the sense that EGAD can be combined with any of them for populating its explorable domain model.

Next to curating large datasets serving as general research platforms, a number of tools aim at supporting specific mining tasks [21], [24], [25], potentially making use of bots to extract the relevant information from software repositories [26]–[28]. The development of most of these tools has been initiated prior to the public release of GA in November 2019, and the analysis of GA workflows has not been considered.

Recently, researchers have started to mine GA using batch tools at the raw data level. Kinsman *et al.* [3] studied how developers use GA, creating a dataset that they later processed to carry out their study. Decan *et al.* [1] investigated trends and adoption patterns of GA through the generation of multiple datasets for answer specific research questions. Similarly, Valenzuela-Toledo and Bergel [4] investigated the evolution of GA workflows, generating a single dataset that they then used to conduct their research. In our work, we shift the focus by introducing a reusable and extensible tool that provides explorable domain models to conduct GA mining studies.

V. Conclusion and Future Work

We have proposed EGAD, a moldable domain-specific tool. We have illustrated our approach by means of the sticky commit narrative. Our tool enables the study of GA through a domain model, and provides an approach to replicate empirical studies on this subject. Our efforts aid in the representation and analysis of GA workflow data, and the creation of an expandable bench for future research.

In future work, we plan to extend EGAD. We envision the visualization of the evolution of GA to enable the discovery of unseen patterns and the automated detection of specific commit groups such as the sticky ones. Knowing the types of commit groups made by developers can guide the development of tools to help developers be more efficient in producing correct workflow specifications.

References

[1] A. Decan, T. Mens, P. R. Mazrae, and M. Golzadeh, "On the use of GitHub actions in software development repositories," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 235–245.

[2] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of CI services in GitHub," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 662–672.

[3] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use GitHub actions to automate their workflows?" in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 420–431.

[4] P. Valenzuela-Toledo and A. Bergel, "Evolution of GitHub action workflows," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 123–127.

[5] A. Chiş, T. Gîrba, J. Kubelka, O. Nierstrasz, S. Reichhart, and A. Syrel, "Moldable tools for object-oriented development," in *Present and Ulterior Software Engineering*. Springer, 2017, pp. 77–101.

[6] O. Nierstrasz and T. Gîrba, "Making systems explainable," in *2022 Working Conference on Software Visualization (VISSOFT)*. IEEE, 2022, pp. 1–4.

[7] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in GitHub for MSR studies," in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 2021, pp. 560–564.

[8] Textualize. (2019, Oct.) pythonpackage.yml file. [Online]. Available: https://github.com/Textualize/rich/blob/master/.github/workflows/pythonpackage.yml

[9] ——. (2020, Oct.) pythonpackage.yml file. [Online]. Available: https://github.com/Textualize/rich

[10] P. Valenzuela-Toledo, A. Bergel, T. Kehrer, and O. Nierstrasz, "EGAD: a Moldable Tool for GitHub Action Analysis," 1 2023. [Online]. Available: https://github.com/pavt/egad

[11] ——, "Egad: A moldable tool for github action analysis," Mar. 2023, tool repository: https://github.com/pavt/egad. [Online]. Available: https://doi.org/10.5281/zenodo.7714219

[12] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM sigsoft software engineering notes*, vol. 30, no. 4, pp. 1–5, 2005.

[13] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 1–1.

[14] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE, 2003, pp. 23–32.

[15] T. Mens, S. Demeyer, M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, "Analysing software repositories to understand software evolution," *Software evolution*, pp. 37–67, 2008.

[16] G. Gousios, "The GHTorent dataset and tool suite," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 233–236.

[17] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean ghtorrent: GitHub data on demand," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 384–387.

[18] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 422–431.

[19] S. Romano, M. Caulo, M. Buompastore, L. Guerra, A. Mounsif, M. Telesca, M. T. Baldassarre, and G. Scanniello, "G-Repo: a tool to support MSR studies on GitHub," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 551–555.

[20] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[21] E. Bogomolov, Y. Golubev, A. Lobanov, V. Kovalenko, and T. Bryksin, "Sosed: a tool for finding similar software projects," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 1316–1320.

[22] R. Di Cosmo and S. Zacchiroli, "Software heritage: Why and how to preserve software source code," in *iPRES 2017-14th International Conference on Digital Preservation*, 2017, pp. 1–10.

[23] V. Markovtsev and W. Long, "Public git archive: a big code dataset for all," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 34–37.

[24] E. Lyulina, A. Birillo, V. Kovalenko, and T. Bryksin, "TaskTracker-tool: A toolkit for tracking of code snapshots and activity data during solution of programming tasks," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 495–501.

[25] N. Sviridov, M. Evtikhiev, and V. Kovalenko, "TNM: A tool for mining of socio-technical data from git repositories," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 295–299.

[26] S. Surana, S. Detroja, and S. Tiwari, "A tool to extract structured data from GitHub," *arXiv preprint arXiv:2012.03453*, 2020.

[27] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillere, "Orion: A software project search engine with integrated diverse software artifacts," in *2013 18th international conference on engineering of complex computer systems*. IEEE, 2013, pp. 242–245.

[28] A. Abdellatif, K. Badran, and E. Shihab, "MSRBot: Using bots to answer questions from software repositories," *Empirical Software Engineering*, vol. 25, no. 3, pp. 1834–1863, 2020.