GitLab2PROV—Provenance of Software Projects hosted on GitLab

Andreas Schreiber Claas de Boer Lynn von Kurnatowski
German Aerospace Center (DLR)

Abstract

Assertions about quality, reliability, or trustworthiness of software systems are important for many software applications. In addition to typical quality assurance measures, we extract the provenance of software artifacts from source code repository's—especially git-based repository's. Software repository's contain information about source code changes, the software development processes, and team interactions. We focus on the web-based DevOps life-cycle tool GitLab, which provides a git-repository manager and other development tools. We propose a provenance model defined using W3C PROV data model and an implementation: GitLab2PROV.

1 Introduction

Software has conquered many application areas over the past years. In particular safety critical systems are affected, such as aviation and aerospace, where errors can have serious consequences. Furthermore, over the years an increasing complexity within these areas also resulted in the need for more and more complex software solutions. Therefore, for many software applications, ensuring the quality, reliability, and trustworthiness of software systems is a basic requirement; which can be achieved with an automated documentation of the overall process.

Our work aims to automatically collect, store, and evaluate the complete provenance of all process steps of a software development project. Since software repositories contain information about source code, software development processes, and team interactions, we extract the provenance of software artifacts based on these repositories. For this purpose, we defined a provenance model for software development processes using the W3C PROV specification; especially the *PROV data model* (PROV-DM [9]). We focus on the web-based DevOps life-cycle tool GitLab[1], which provides a git-repository manager, issue-tracking, Wiki, and continuous integration and deployment pipelines.

[1]https://about.gitlab.com/features/

Among the many existing code-hosting platforms, GitLab belongs to the most popular ones[2] with $> 30,000,000$ users; used by $> 100,000$ organizations (including the German Aerospace Center). In addition to the public Open Source platform `gitlab.com`, GitLab can be self-hosted within organizations—which many of these use GitLab as their internal platform for *Inner Source* development [3].

Since GitLab is widely used, we contribute the following:

- Background information about provenance of software artifacts and development processes where we briefly summarize our work on a high-level provenance model for software development (Section 2).
- A reasonably comprehensive overview of provenance for `git` services including references to influential work (Section 3).
- A description of GitLab2PROV for extracting provenance graphs from GitLab instances (Section 4).
- An evaluation using an example of an Inner Source project from DLR's GitLab instance (Section 5).

2 Provenance of Software Artifacts

Due to the complexity of today's software many development process models evolved, together with many tools. A typical tool suite consists of an integrated development environment (IDE), a version control system, an issue tracker, a continuous integration framework, and a documentation management system. Many interaction occurs between developers, between the tools they use during the development process, and automatically between different tools.

[2]https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities

In our previous work [17], we developed an high-level *extensible* conceptual provenance model for software development processes using the *Open Provenance Model* (OPM) notation. We updated the model from OPM to PROV. The model covers issue tracking (requirements, bugs), development (planning, design, coding, testing), continuous integration, documentation (developer, user), and release (Figure 1).

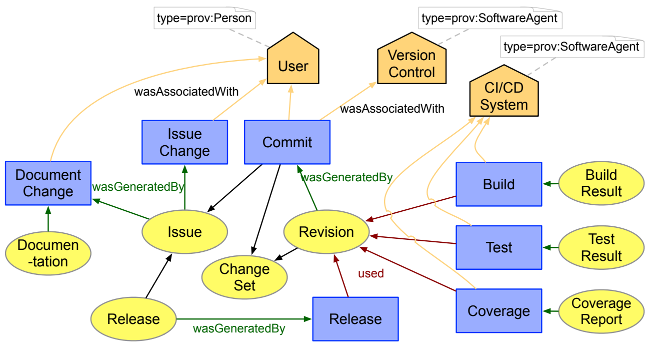


Figure 1: High-level conceptual PROV model for software development processes (excerpt; for clarity, some relation types and most attributes are left out).

The conceptual PROV model can—and should—be extended with further activities such as editing or deployment and further actors such as software bots or software analytics tools. If used for concrete processes, each of the PROV class elements must be defined with specialized class elements—for example:

- The generic role `User` Agent has to be specialized to roles such as `Author` Agent or `Test Manager` Agent. Another way would be to specify the role of an actor by adding a property "role" to the relation, which relates that actor with actions.

- A PROV model, which is more specific for `git`, has specific class elements such as `GitLab` Agent as a specialization of `Version Control` Agent. Also the activities, such as `Commit` Activity or `Issue Change` Activity, have much more details about relations to related activities and related entities.

To get meaningful knowledge and insights from provenance graphs [12], one has to extend toPROV model according to questions of interest. Example questions include questions related to quality assurance (e.g., "*How many releases have been produced this year?*"), process compliance (e.g., "*From which revision was release X built?*"), developer performance (e.g., "*Which developer is most active in contributing documentation?*"), and others [17].

3 Provenance for `git` Services

We generate provenance from the distributed version-control system `git`, which tracks changes in a file system. Nowadays, `git` is used in many developer workflows. Especially Open-Source projects use `git` via hosting services such as BITBUCKET, GITHUB, or GITLAB.

Based on the general PROV model (Section 2), we model all actions that are possible with `git` services with more specialized PROV models. Our work relies on the previous works GIT2PROV by Nies et al. [4] and GITHUB2PROV by Packer et al. [10]. We provide a PROV model for GITLAB and the implementation "GITLAB2PROV" (Section 4). Similar to Packer et al. for GITHUB2PROV, our PROV model extends the model of GIT2PROV with activities that are beyond basic `git` functions (i.e., specific functions of GITLAB such as issue management).

We store the PROV graph in databases such as the PROVSTORE [6] or the graph database NEO4J using additional tools (Figure 2).

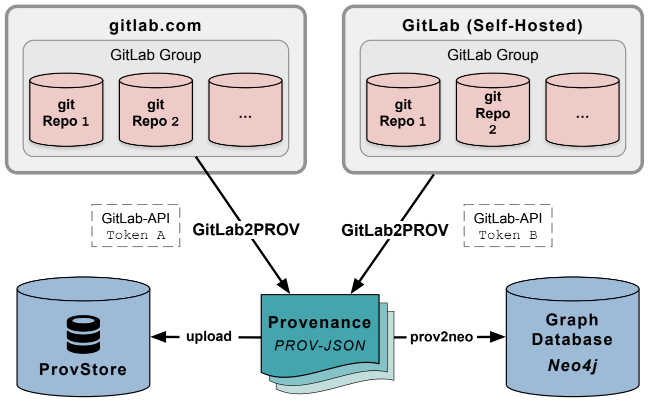


Figure 2: Extracting provenance from `git` repositories. Our tool GITLAB2PROV writes the provenance to a PROV-JSON file, which we upload to the PROVSTORE and import into NEO4J using our tool PROV2NEO (see Section *Availability*).

In NEO4J, performing queries, graph reasoning, or extracting knowledge otherwise is possible by using CYPHER queries or graph algorithms. For example, for analyzing software projects we use CYPHER queries to investigate the following (see Section 5):

- Graph structure information, such as number of nodes and edges, which represent the number of files, commits, and developer activities in total.

- Graph structure changes over time, such as active periods of developers.

- Process-specific questions, such as interactions of developers during curse of the project.

4 GitLab2PROV

GitLab2PROV extracts information from instances of GitLab and stores the PROV graph in a provenance notation file format specified by the W3C PROV specification. We describe GitLab2PROV's provenance model (Section 4.1), give details on its implementation (Section 4.2), and give an example on the extracted provenance (Section 4.3).

4.1 Provenance Model

GitLab2PROV uses PROV models to record actions that can occur within arbitrary GitLab projects[3].

4.1.1 Commits

Three of the employed models are for capturing of different effects that `git` *commits* can have on the status and content of files. The identified effects are the *addition of a new file* (Figure 3a), the *change of a file* (Figure 3b), and the *deletion of a file* (Figure 3c).

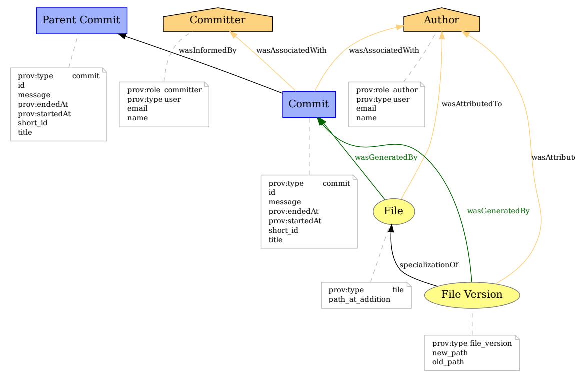
For example, when adding or modifying a file via a commit in the `git` repository, GitLab2PROV records the following information:

- A PROV entity Entity for the `File Version` at the point of addition as well as an entity for the `File` itself. The `File Version` is marked as a specialization of the specific `File`.
- The author and the committer of the git commit as represented by the PROV agents Agent `Author` and `Committer`. The `File` and `File Version` entities are attributed to the `Author` to represent that the `Author` is responsible for their content.
- The commit that adds the file is represented by the specialized PROV activity Activity `Commit` which generates the PROV entities `File` and `File Version`. The commits directly preceding `Commit` are also recorded. The `Author` and the `Committer` are associated with the `Commit` activity, since they are responsible for the commit taking place.

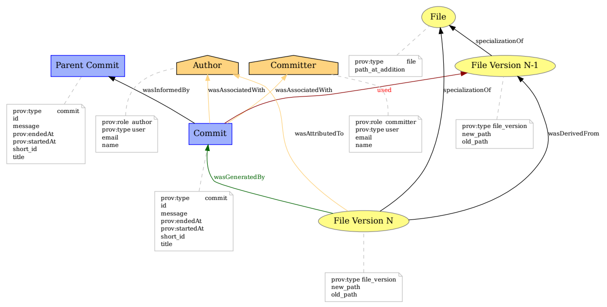
4.1.2 Issue Management and Merge Requests

Two models capture user interactions and events that occur on or with GitLab Web resources such as maintaining GitLab *issues* (Figure 4), managing GitLab *merge requests* (Figure 5), or using the GitLab Web interface for commits. These interactions happen in sequence, one event following the next, without branching the timeline of events. Packer et al. [10] used the term *annotation* for such interactions, as every interaction annotates additional information to the resource itself. An emoji reaction could add a "thumps up" to an issue where previously was none or a comment could be added to the discussion of a code review in the comment section of a merge request. The issue and merge request model capture the chain of consecutive events, which occur on the respective resources.

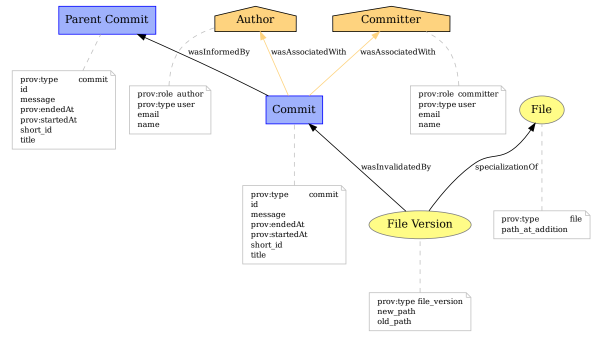
[3]The PROV model and results in this paper are defined and produced using GitLab2PROV version 0.4 (https://doi.org/10.5281/zenodo.4714963)


Parent Commit
Committer
Author
wasInformedBy
wasAssociatedWith
wasAssociatedWith
Commit
wasAttributedTo
wasGeneratedBy
wasAttributedTo
File
wasGeneratedBy
specializationOf
File Version
prov:type commit id message prov:endedAt prov:startedAt short_id title
prov:role committer prov:type user email name
prov:role author prov:type user email name
prov:type file path_at_addition
prov:type file_version new_path old_path

(a) Addition of a file.


File
specializationOf
Parent Commit
Author
Committer
File Version N-1
wasInformedBy
wasAssociatedWith
wasAssociatedWith
used
specializationOf
Commit
wasAttributedTo
wasDerivedFrom
wasGeneratedBy
File Version N
prov:type commit id message prov:endedAt prov:startedAt short_id title
prov:role author prov:type user email name
prov:role committer prov:type user email name
prov:type file path_at_addition
prov:type file_version new_path old_path

(b) Change of a file.


Parent Commit
Author
Committer
wasInformedBy
wasAssociatedWith
wasAssociatedWith
Commit
File
wasInvalidatedBy
specializationOf
File Version
prov:type commit id message prov:endedAt prov:startedAt short_id title
prov:role author prov:type user email name
prov:role committer prov:type user email name
prov:type file path_at_addition
prov:type file_version new_path old_path

(c) Deletion of a file.

Figure 3: PROV sub-models for the different actions on files, which users can perform by `git` commits.

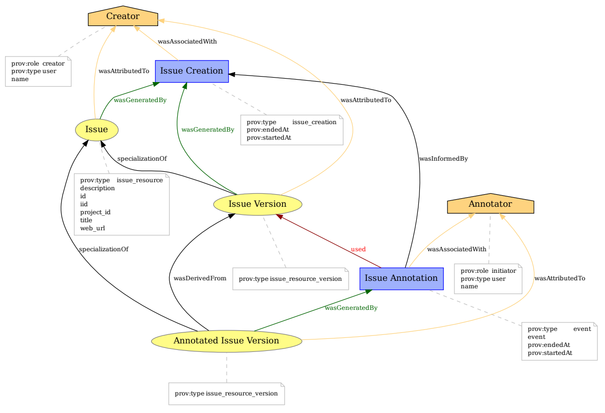


Figure 4: PROV model for maintaining a GITLAB issue using its Web interface.

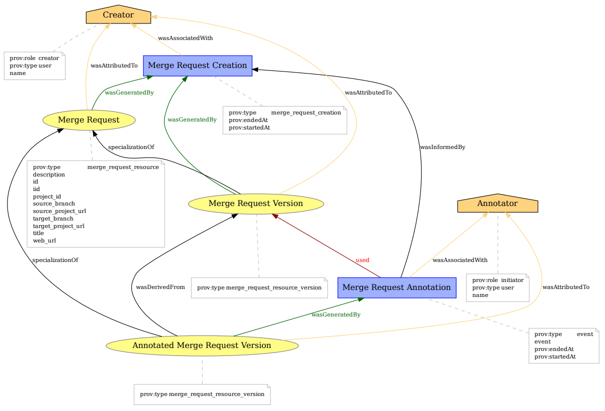


Figure 5: PROV model for creating and annotating a GITLAB merge request.

Apart from replacing the name "Issue" with "Merge Request" both models are equal in their conception. For simplicity, we describe the models as one, exchanging "Issue" and "Merge Request" for "Resource." Both models record the following information:

- A PROV entity Entity for the `Resource Version` at the point of its creation, one entity for the `Resource` itself as well as an entity Annotated Resource Version representing the state of the resource after every new `Resource Annotation`. The `Resource Version` and `Annotated Resource Version` entities are specializations of the `Resource` entity. Each new `Annotated Resource Version` is derived from the entity representing its previous version.

- A PROV agent Agent for the `Creator` of the resource, as well as an agent for every user that carries out an annotation event. The latter agents are called `Annotator` and are responsible for the `Resource Annotation` activity that they triggered by their action. For both issues and merge requests the GitLab user that first opened the issue or request is considered to be its `Creator`. The `Creator` is responsible for the `Resource Creation` activity. The `Resource` entity and the initial `Resource Version` entities are attributed to the `Creator` agent.

- A PROV activity Activity for the `Resource Creation` that generates `Resource` and `Resource Version` entities together with an activity for each `Resource Annotation`. `Resource Annotation` activities use a specific `Resource Version` entity that represents the version of the resource just before the annotation event took place and generates a new version in the form of the `Annotated Resource Version` entity.

4.2 Implementation

GITLAB2PROV is implemented in Python and can be used as a command line tool or as a library for Python to compute the provenance graph of a single or multiple GITLAB projects.

To extract a provenance graph from `git` repositories, the tool GIT2PROV [4] first clones a `git` repository, followed by executing a specific "`git log`" command inside of it, parsing the generated output, and converting the parsed data into a provenance graph. The tool GITHUB2PROV [10] combines the approach of GIT2PROV with the addition of requesting API data for GITHUB; both stored in a tailored provenance model.

In contrast to these implementations, GITLAB2PROV gets its required data solely from the GITLAB REST API and does not use the command line tool "`git`." This reduces the multiplicity of data sources and to avert having to clone a repository to a temporary location for data retrieval. As a side effect, the independence from `git` allows GITLAB2PROV to run on devices on which users lack file permissions or on which `git` is not installed.

The bottleneck of this approach is the generation of the desired PROV graph, as GITLAB2PROV has to wait for all API requests to return, before being able to resume with the computation of the graph itself. Instead of waiting for every single GET request to dispatch the next one, we chose to speed up the retrieval of API data by performing the necessary HTTP requests asynchronously.

Each GITLAB instance defines a *rate limit* for API requests, which confines the speed at which GITLAB2PROV is able to request data (set to 10 requests per second by default). Using the asynchronous HTTP client/server framework "aiohttp,"[4] we implemented a custom, asynchronous Token Bucket API client to do requests in as little time as possible. At the time of implementation, there was no asynchronous GITLAB API

[4] https://github.com/aio-libs/aiohttp

client available. This may change in the future with the addition of support for asynchronous requests by the API client "python-gitlab."[5]

To generate provenance representations, we use the Python package "prov,"[6] a library for W3C PROV, that supports serialization of PROV documents to the text-based representations PROV-O (RDF), PROV-XML, PROV-JSON, and DOT (GraphViz).

A known limitation is, that GITLAB2PROV cannot update previously extracted provenance when new GITLAB events occur; it extracts the entire history again. To overcome this drawback, we plan to use GITLAB "*Webhooks*" to record events immediately when they happen.

4.3 Querying the GitLab Provenance

We show how to query the provenance graph on an example for a single Open Source project from gitlab.com: Flockademic/whereisscihub,[7]

As an example query we choose the workload metric **M7** (*The number of events an agent is associated with*) from Packer et al. [10], which notates in CYPHER as:

```
MATCH
  (user:Agent)-[:wasAssociatedWith]-(event:Activity)
WHERE
  event.`prov:type` = "commit" OR
  event.`prov:type` = "issue" OR
  event.`prov:type` = "merge_request"
RETURN
  user.user_name,
  COUNT(DISTINCT event) as event_count
ORDER BY event_count DESC
```

The result of that query is:

"user.user_name"	"event_count"
"Vincent"	32
"Jon Mountjoy"	9
"GitHub"	7
"Jeremy Morrell"	5
"Hunter Loftis"	2
"scantini"	2
"Jon Byrum"	1

5 Evaluation

We evaluate GITLAB2PROV with an Inner Source project that consists of multiple repositories, which all belong to the same GITLAB group.

[5]https://github.com/python-gitlab/python-gitlab

[6]https://github.com/trungdong/prov

[7]https://gitlab.com/Flockademic/whereisscihub. PROV file at https://openprovenance.org/store/documents/4128.

We selected the software system OPENVOCS [16], which is an open and flexible software for control room communication developed by DLR's German Space Operations Center. We selected three repositories: openvocs/code, openvocs/voice_control, and openvocs/load_tests. For those projects, we are particularly interested in the following questions—its results are provides as charts and diagrams produced using the graphing library PLOT.LY:

(1) How many activities have been conducted and how many files have been produced or changed? ↦ Figure 6

(2) What and how many interactions took place for each of the git projects? ↦ Figure 7

(3) Who contributed to each of the projects? ↦ Figure 8

(4) Who is active during development? ↦ Figure 9

(5) How did the project activities grow over time? ↦ Figure 10

(6) What are developers interactions over time? ↦ Figure 11

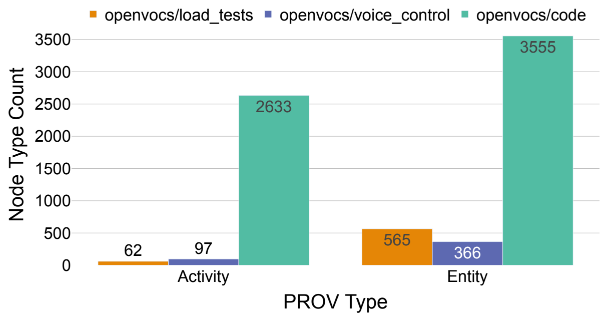


Figure 6: Results for question (1): number of PROV types for each of the OPENVOCS software repositories.

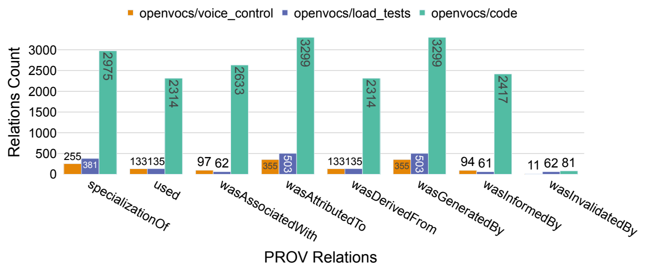


Figure 7: Results for question (2): the number of PROV relations for each of the OPENVOCS software repositories.

Based on the results, we can gain a basic understanding of the project and its development history. In our example, this leads to the following insights:

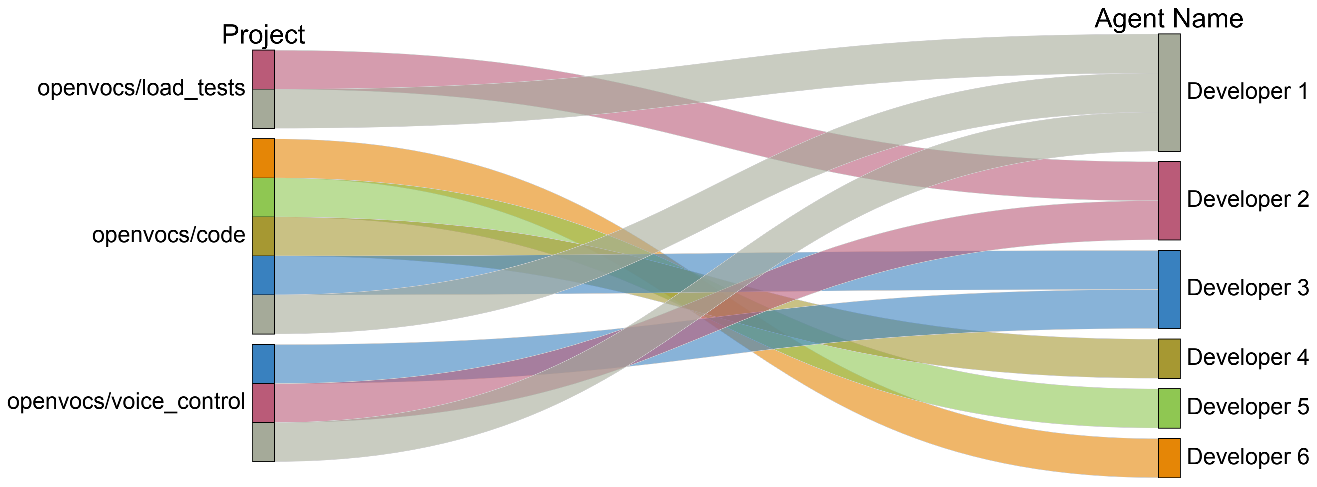


Figure 8: Results for question (3): mapping of developers (PROV agents) to each of the OPENVOCS software repositories. The agents are connected with all the projects they participate in, which is the case, if in the provenance graph an agent is assigned to an activity.

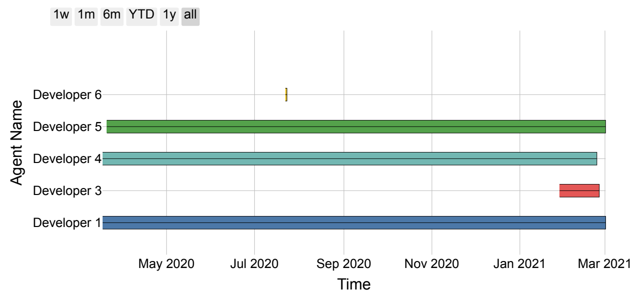


Figure 9: Results for question (4): activity period of the agents involved in the OPENVOCS repository.

- Developer 1 is active in all three repositories (Figure 8). This leads to the assumption that Developer 1 has a central role in the project; a conclusion that is supported by the agent timeline (Figure 9). Developer 1 was active during the entire duration of the project. In addition to Developer 1, Developers 3 and 4 were also active during the entire course of the project. This indicates that these three developers might have a high level of knowledge about the project.

- Based on the evolution of the number of graph nodes (Figure 10), we can identify whether new actions are actively taking place in a GITLAB project. Thus the evolution of the graph is an indicator of the development activity of a project. For our example, in all three repositories the number of entities per activity is high (Figure 10). Therefore, it can be assumed that the actions performed in the project are predominantly commits, which add or modify files, or the creation of issues.

- The event timeline (Figure 11) show how and when the project, and consecutively the developers, were particularly active. As a general conclusion, we find that the project was particularly active at the beginning from March 2020 to June 2020 and from January 2021 to March 2021.

6 Related Work

Our approach combines two major research areas: software repository mining and provenance. The software repository mining community identified early on the benefits of analyzing software artifacts from software repositories, for example Bevan et al. [2] and Dyer et al. [5]. Especially, infrastructures for repository mining at large-scale are available, such as WORLD OF CODE by Ma et al. [8] or SMARTSHARK by Trautsch et al. [15]. However, these repository mining tools do not generate provenance graphs.

Several works focus on `git` only, which—in contrast to our work—do not rely on a standardized graph model such as PROV and do not include knowledge from the hosting service such as issues. Two examples are:

- GITGRAPH by Zhao et al. [18] constructs automatically a knowledge graph associated with a `git` repository. Their knowledge graph contains commits, files, classes, methods, and branches. The graph is stored in graph database and queried using CYPHERfor understanding the content of the repository and for visual exploration.

- GITHRU by Kim et al. [7] focus on visual analytics for understanding the software development history. They use a visual encoding that allows scalable exploration of large `git` commit graphs.

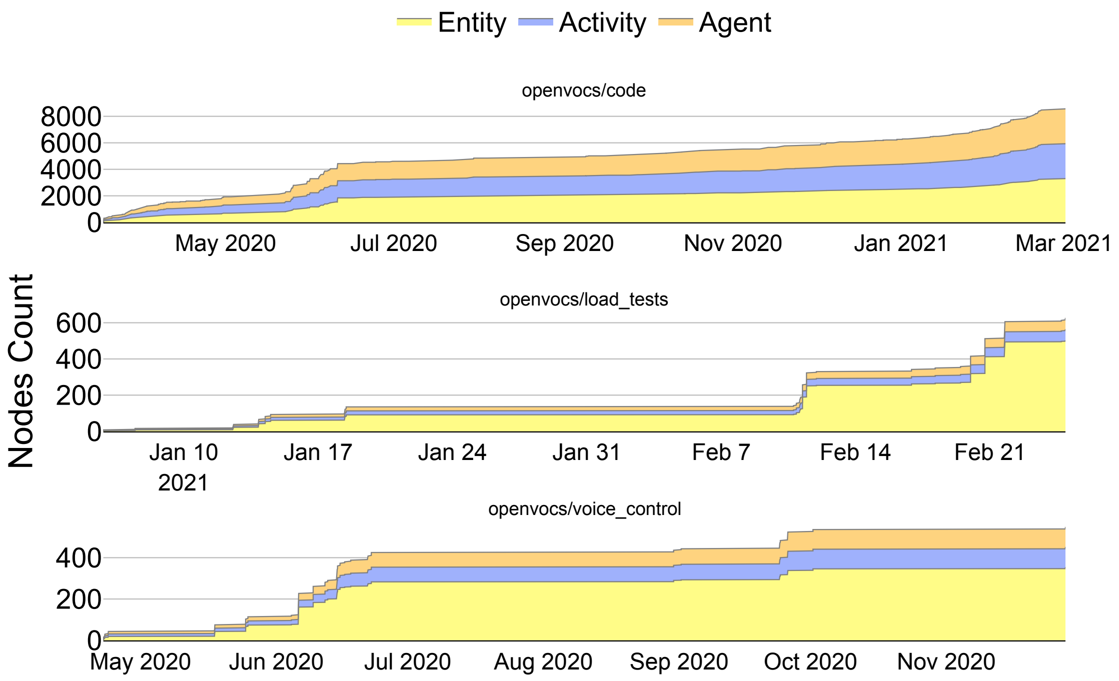


Figure 10: Results for question (5): the number of nodes over project run-time for each PROV class element (*entity*, *activity*, and *agent*) for each of the OPENVOCS software repositories.

Costa and Castro [1] propose an approach called "*iSPuP*" (improving Software Process using Provenance). iSPuP uses provenance to monitor and analyse software processes and provides information about artifacts that can increase new process instances at runtime.

7 Conclusions and Future Work

We presented a provenance model in PROV for software development projects, which are hosted on GITLAB; and the implementation GITLAB2PROV, which extracts and stores provenance graphs from GITLAB instances. With the provenance graphs, we can answers questions on the development process (e.g., for reporting the project's state). However, some of these results can be retrieved via GITLAB's web-interface also. We see more benefit when combining the provenance graph with other data sources, such as analytics results of the source code, text mining results of content (issues, wiki pages, commit messages, etc.), or communication patterns between developers.

Our current work focuses on applications such as:

- Automated, provenance-driven security audits for git-based repositories, which we apply to Germany's Corona-Warn-App [13].
- Visual analysis of contributions to Open Source projects by non-team developers [11].
- Detecting community smell patterns [14] of communication and collaboration in Open Source projects.

As future work, we extend the existing PROV model to support more GITLAB events—based on requirements by provenance questions of applications. For example, we plan to extend the PROV model for release actions, continuous integration and continuous deployment, and documentation changes in the Wiki. Maybe, we model and extract instance-wide security audit events such as (failed) sign-ins, added and removed users, or created or revoked user's personal access token.

Availability

GITLAB2PROV is available as Open Source software under the MIT license: https://github.com/DLR-SC/gitlab2prov.

The tool PROV2NEO is available under the MIT license at https://github.com/DLR-SC/prov2neo.

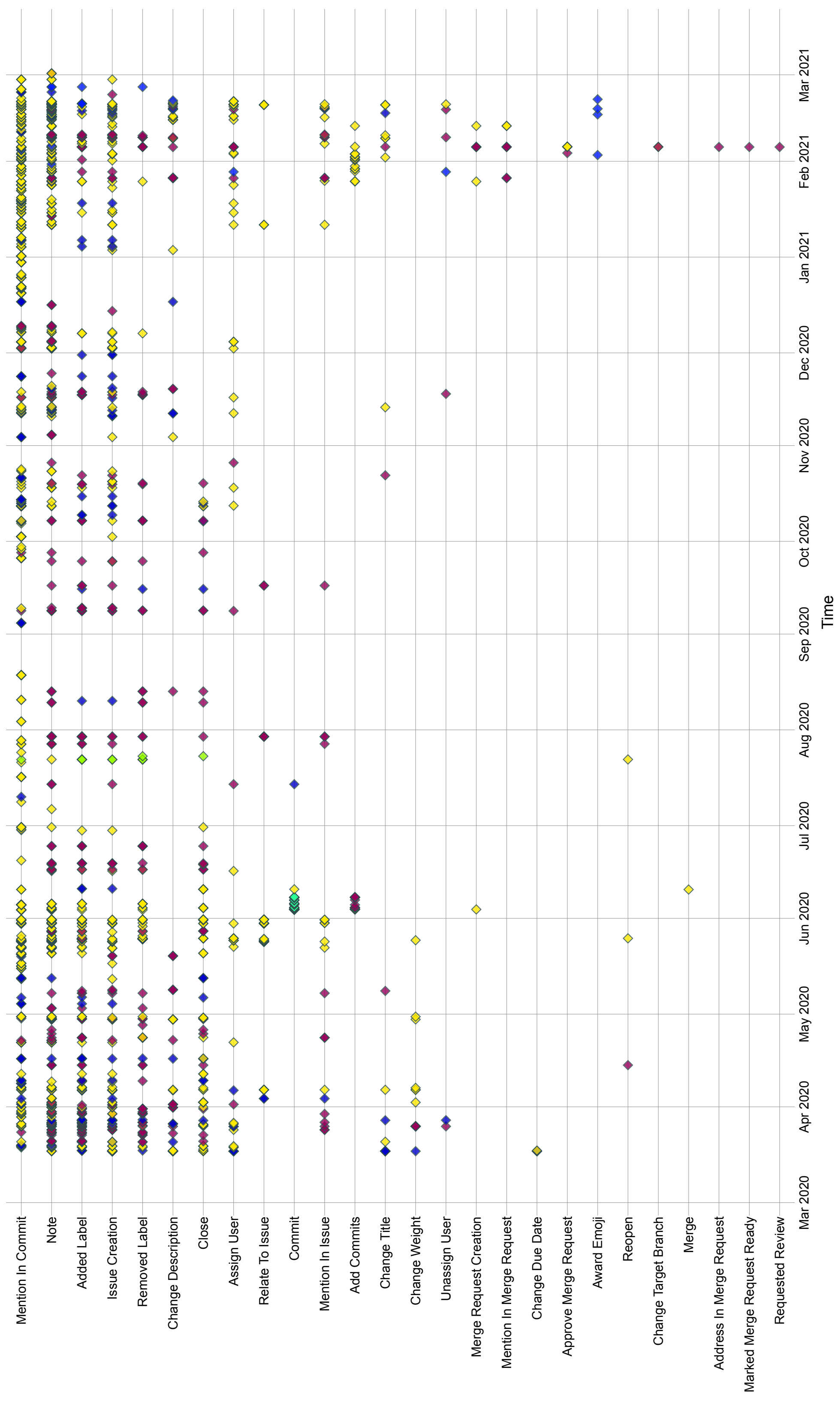


Figure 11: Results for question (6): timeline for each interaction event in the OpenVocs repository. The markers' colors (colored rhombuses) indicate which developer has initiated the corresponding event.

References

[1] Gabriella Castro Barbosa Costa. Using data provenance to improve software process enactment, monitoring and analysis. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 875–878, New York, NY, USA, 2016. ACM.

[2] Jennifer Bevan, E. James Whitehead, Sunghun Kim, and Michael Godfrey. Facilitating software evolution research with kenyon. In *Proceedings of the 10th European Software Engineering Conference*, ESEC/FSE-13, pages 177–186, New York, NY, USA, 2005. ACM.

[3] Maximilian Capraro and Dirk Riehle. Inner source definition, benefits, and challenges. *ACM Comput. Surv.*, 49(4), December 2016.

[4] Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul Groth, Erik Mannens, and Rik Van De Walle. Git2PROV: Exposing version control system content as W3C PROV. In *Proceedings of the 12th International Semantic Web Conference*, volume 1035, pages 125–128. CEUR-WS.org, 2013.

[5] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: Ultra-large-scale software repository and source-code mining. *ACM Trans. Softw. Eng. Methodol.*, 25(1), December 2015.

[6] Trung Dong Huynh and Luc Moreau. ProvStore: A public provenance repository. In *Provenance and Annotation of Data and Processes, IPAW 2014*, volume 8628 of *Lecture Notes in Computer Science*, pages 275–277. Springer, 2015.

[7] Youngtaek Kim, Jaeyoung Kim, Hyeon Jeon, Young-Ho Kim, Hyunjoo Song, Bohyoung Kim, and Jinwook Seo. Githru: Visual analytics for understanding software development history through git metadata analysis.

[8] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. World of code: An infrastructure for mining the universe of open source vcs data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 143–154, 2019.

[9] Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. PROV-DM: The PROV data model, 2013.

[10] Heather S. Packer, Adriane Chapman, and Leslie Carr. GitHub2PROV: Provenance for supporting software project management. In *Proceedings of the 11th USENIX Conference on Theory and Practice of Provenance*, TAPP'19, USA, 2019. USENIX Association.

[11] Andreas Schreiber. Visualization of contributions to open-source projects. In *Proceedings of the 13th International Symposium on Visual Information Communication and Interaction*, New York, NY, USA, 2020. ACM.

[12] Andreas Schreiber and Claas de Boer. Modelling knowledge about software processes using provenance graphs and its application to git-based version control systems. In *Proceedings of the 42nd International Conference on Software Engineering Workshops*, ICSEW'20, pages 358–359, New York, NY, USA, 2020. ACM.

[13] Tim Sonnekalb, Thomas S. Heinze, Lynn von Kurnatowski, Andreas Schreiber, Jesus M. Gonzalez-Barahona, and Heather Packer. Towards automated, provenance-driven security audit for git-based repositories: Applied to Germany's Corona-Warn-App. In *Proceedings of the 3rd International Workshop on Software Security from Design to Deployment (SEAD '20)*, New York, NY, USA, 2020. ACM.

[14] Damian A. A. Tamburri, Fabio Palomba, and Rick Kazman. Exploring community smells in open-source: An automated approach. *IEEE Transactions on Software Engineering*, pages 630–652, 2019.

[15] Fabian Trautsch, Steffen Herbold, Philip Herbold, and Jens Grabowski. Addressing problems with replicability and validity of repository mining studies through a smart data platform. *Empirical Software Engineering*, 23(2):1036–1083, 2017.

[16] Markus Töpfer, Anja Sonnenberg, and Rolf A. Kozlowski. *Open Source based Voice Communication for Mission Control*. American Institute of Aeronautics and Astronautics, Daejeon, Korea, 2016.

[17] Heinrich Wendel, Markus Kunde, and Andreas Schreiber. Provenance of software development processes. In *Provenance and Annotation of Data and Processes, IPAW 2010*, volume 6378 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 2010.

[18] Yanjie Zhao, Haoyu Wang, Lei Ma, Yuxin Liu, Li Li, and John Grundy. Knowledge graphing git repositories: A preliminary study. In *26th International Conference on Software Analysis, Evolution and Reengineering, SANER 2019*, pages 599–603, 2019.