PatchScope – A Modular Tool for Annotating and Analyzing Contributions

Piotr Przymus*, Jakub Narębski†, Mikołaj Fejzer‡, and Krzysztof Stencel§
Nicolaus Copernicus University in Toruń, Toruń, Poland,
**University of Warsaw*, Warsaw, Poland
Email: *piotr.przymus, †jakub.narebski, ‡mfejzer, @mat.umk.pl §, stencel@mimuw.edu.pl

***Abstract*—PatchScope is a modular framework for analyzing software contributions, providing insights beyond traditional metrics. Its customizable modules enable granular analysis of contribution dynamics and project evolution.**

At its core, a flexible automatic code annotator labels source code lines using customizable rules, categorizing changes such as documentation, testing, or code updates. Based on that PatchScope produces reports with actionable insights for team evaluation, expertise identification, and longitudinal studies.

With applications in annotation automation, bug localization, patch dataset analysis, and project monitoring, PatchScope is a versatile tool for researchers, project managers, and developers, emphasizing flexibility and ease of use.

***Index Terms*—source code annotation, commit annotation, code analysis, contributor profiling, project insights**

I. Introduction

In today's collaborative software development landscape, understanding and evaluating developer contributions is critical for project success. While code contributions are tracked, the details surrounding what developers contribute—whether code, documentation, translation, or project configuration—can reveal valuable insights into each developer's role and expertise within a project. This paper introduces PatchScope,a new tool that provides a systematic approach to annotating, filtering, and reporting of developer contributions in Git repositories.

PatchScope comprises three core components that work together to offer a flexible comprehensive analysis framework.

First, patch-selecting component allows targeted commit filtering for the annotation process. PatchScope filters select patches by specific users, commits between defined versions, or commits affecting given directory. Filtered sets of patches are then processed by the annotator.

Second, the code annotator enables granular labeling of changed source code lines, based on the file name, its place in the project structure, and tokens of lexed contents of the file. It comes with sane and verified defaults, but also allows defining custom annotation rules. This customization lets users focus on elements most relevant to their analyses.

Third, the reporting module offers flexible output options for presenting findings from the annotated commits. It gathers insights from selected annotations into a single JSON data file. Users can also generate web-based dashboard, with analytical plots comparable to an advanced version of GitHub's Insights view. The starting page of the dashboard provides detailed profiles of user contributions by type.

PatchScope is designed with modularity in mind, allowing each component to be used independently, or substituted with custom tools for tailored analysis workflows.

PatchScope has versatile applications, including manual annotation workflows, bug localization, project monitoring, research on patch-based datasets, behavioral analysis and security investigations.

The resulting tool and example annotations are publicly available at https://figshare.com/s/d78ffecdee987f2ed55d, the source code at GitHub https://github.com/ncusi/PatchScope, and a demo at https://patchscope-9d05e7f15fec.herokuapp.com.

II. Tool description

PatchScope is a modular tool for analyzing and reporting on code changes. The tool (see Fig. 1) integrates three key components: (1) extracting patches from version control systems or user inputs, (2) applying predefined annotation rules, with verified defaults for Python and Java, and (3) generating configurable reports with advanced visualizations. We describe each module in detail below.

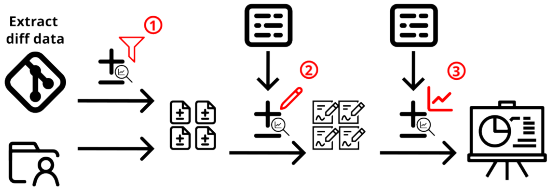


Fig. 1: Overview of tool components

A. Patches selection

The first phase of the process is Patch Selection Phase; this phase can be integrated with the next phase. The tool supports three types of patches sources: individual patch files, patch datasets (directories with patches), and Git repositories.

When annotating changes in a Git repository, users can define commit of interest through custom lists, and through Git's native revision filtering options. The latter allows specifying, among others, commits within selected version range, commits authored by a specific user, or changes on a selected branch.

This patch selection mechanism helps with ad-hoc analysis and reduces computational costs, particularly for large repositories.

B. Annotation Tool

The second step supported by PatchScope, the Annotation Phase, is designed to produce detailed line-by-line annotations that capture the unique characteristics of each changed line within a patch file or a commit. For each line, three pieces of data are generated, containing (1) the line type, determined by analyzing lexer output (by default we use Pygments [1]), (2) file-specific information derived from GitHub Linguist [2], and (3) the file path within the project structure.

This data allows creating customizable annotations that can be tailored to the specifics of users' project or organization. For example, in a large organization, users can define rules aligned with common project structures, while in language-specific environments, rules can adapt to typical organizational patterns in a given programming language.

The tool includes a set of predefined general rules compatible with any file format supported by Pygments. Additionally, it offers specialized rule sets for Python and Java projects, curated and validated through comparisons with manual annotations by experienced programmers on publicly available annotated datasets (Java [3], Python [4] with annotations from [5]). This ensures end users have a clear understanding of the tool's accuracy.

New annotation rules can be easily introduced as shown by the following example of configuration code. Here the purpose of a file is defined based on its path. Moreover, a line callback function determines line types. Empty lines are marked as such. Comments and tests are categorized as documentation.

```
PURPOSE['test/**'] = 'test'
PURPOSE['po{,4a}/**'] = 'translation'

def callback(file_purpose, tokens):
    if line_is_empty(tokens): return 'empty'
    if file_purpose in {'test', 'code'}
        and line_is_comment(tokens):
        return 'documentation'
    return file_purpose
```

In addition to annotating changed lines, this phase also extracts commit metadata, e.g. the time a commit was authored. It also computes quantitative and qualitative properties regarding each patch [6], e.g. (1) the numbers of source code lines that were added, removed or modified by a patch, (2) the number of modified files, (3) the number of change groups, i.e. sequences of continuous changes in a file, and (4) spreadings of change groups (chunks), etc.

C. Aggregation and reporting

The aggregation and reporting component forms the final phase of PatchScope. It provides a flexible framework for deriving insights from annotated commit data. It enables users to generate customizable reports and aggregation pipelines tailored to specific needs, while offering ready-to-use templates for immediate analysis. This design ensures adaptability for diverse project requirements while maintaining ease of use.

As a demonstration, PatchScope includes an enhanced GitHub Insights-style report. This report offers a detailed overview of different types of user contributions, tracking their evolution over time. It features a distribution of changed lines among different line types, and a developer-centric view that highlights individual contributions. These visualizations offer actionable insights into project dynamics and individual developer contributions.

By integrating foundational reporting tools with extensibility, this component supports advanced use cases such as expertise identification, contribution comparisons, and engagement trend analysis. Its modularity empowers users to customize or extend reporting, enabling comprehensive project management and contributor evaluation across varied development contexts.

Dashboard and supported plots: The tool aggregates various visualizations into a unified dashboard resembling GitHub Insights. By presenting a cohesive project overview, it offers actionable insights into team dynamics, contribution patterns, and project health. This allows project managers to optimize workflows, balance priorities, and maintain high-quality outputs with minimal configuration.

Week-Hour Heatmap (2a): This visualization captures developer activity across the week and by hour of the day, highlighting productivity patterns. It can help a project manager to identify peak activity periods. It can also be an indicator of a change in developers' behaviour.

Sankey Project Flow (2b): This plot illustrates the flow of lines through project directories and their associated labels. It provides a compact view of how contributions are distributed through project structure. It helps detecting imbalances in development efforts across different parts of the project.

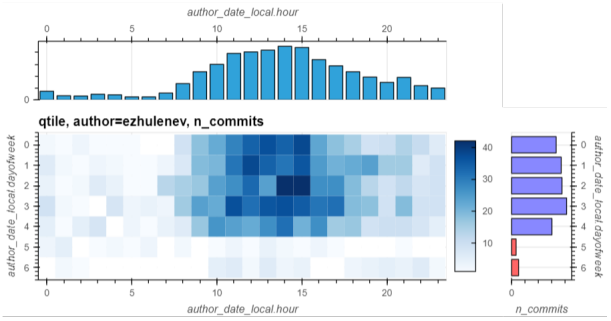
Monthly Contribution Heatmap (2c): It offers a longitudinal perspective. This heatmap tracks contributions by assigned label (e.g. "code," "testing,", "documentation") over time. It can uncover trends, such as a decrease in "testing" or "documenting" activity post-release. Thus it gives actionable insights into the development process.

User and Project Summary (2d): This summary consolidates metrics on lines added, removed, and their distribution across predefined categories. It provides insights into individual contributions, identifying gaps such as minimal "testing" involvement, and supports targeted strategies like mentoring or task reassignment to address disparities.

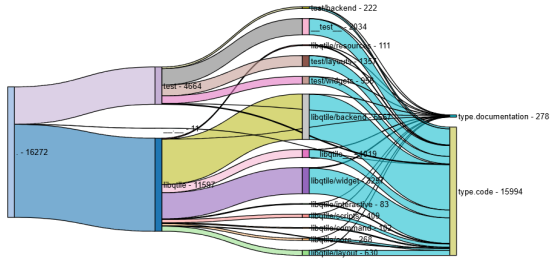
III. Possible Applications

Manual annotation frameworks – PatchScope is a powerful tool for manual annotation workflows, particularly in research contexts where expert-driven line annotations are crucial for specific objectives [7], [3], [5]. By automating routine annotations (pre-labeling of lines), PatchScope reduces the manual workloads and enables experts to focus on intricate, high-value tasks that require specialized attention.

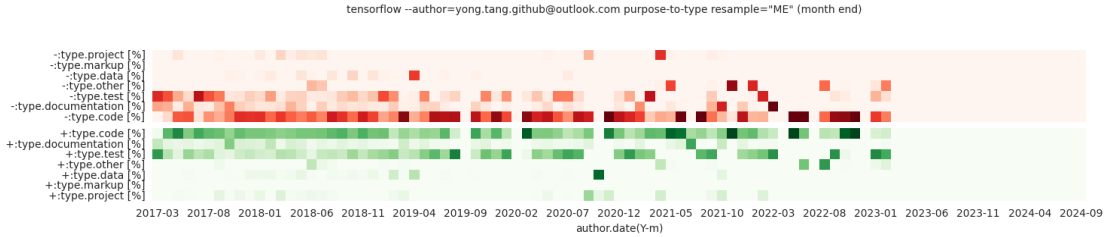
Bug localization – PatchScope is currently being tested for its effectiveness in precise bug localization at both file and line levels. By generating detailed annotations, it enhances bug localization systems not only by identifying modified



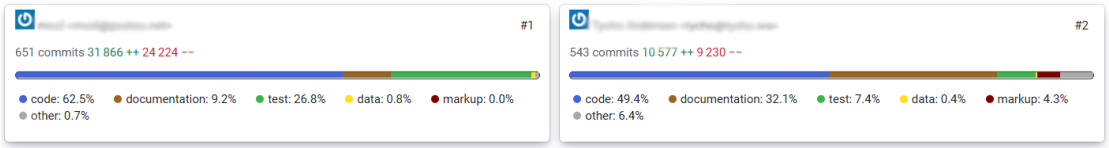
(a) Working days and working hours.



(b) Distribution of contribution to the project structure (left to right) with annotations (right).



(c) Annotated heatmap of daily contributions.



(d) Annotated summary of user contributions.

lines but also by assessing their significance in the repair process. Preliminary evaluations suggest that these annotations aid prioritizing files or lines for fixes. This functionality is under active investigation [8].

Project Monitoring – Furthermore, PatchScope can be used by project managers to monitor team workflows and project progress effectively. Through visualizations like activity heatmaps and contribution summaries, managers can identify productivity patterns, track trends, and detect imbalances. This supports efficient task scheduling, resource allocation, and performance evaluation, helping optimize team coordination and project outcomes.

Supporting Bug Dataset Research – PatchScope enhances research on patch-based datasets such as those presented in articles [6], [3], [5]. By streamlining patch analysis and providing key characteristics, it offers researchers a quick starting point for exploration. This facilitates rapid prototyping, hypothesis validation, and sanity checks. Thus it enables researchers to efficiently refine their focus before fully committing to a problem.

Behavioral Analysis and Security – PatchScope enables the investigation of user behaviors over extended periods. For instance, the XZ attack revealed how malicious developers can hide harmful contributions among legitimate ones. PatchScope supports efficient post-mortem analyses, uncovering patterns and strategies employed by malicious actors, enhancing understanding of such tactics.

IV. Experiments

Effectiveness in Line Annotation To evaluate the effectiveness of PatchScope, we compared its performance against two manually annotated datasets: the Java-based annotated dataset from Herbold et al. [3], and the Python-based BugsInPy dataset [4] — with annotations from HaPy-Bugs [5]. Each bug in these datasets, along with its associated changed lines, has been annotated by four developers for the former [3], and three developers for the latter [5]. Consensus is defined accordingly for each dataset (in line with dataset assumptions). Details can be found in Table I.

We measured the effectiveness of PatchScope by matching lines with the dataset, and then finding if the label generated for a line matches the consensus label from the dataset. The overall agreement is reported as the percentage of correctly annotated lines, compared to all matched lines with consensus. Agreement for individual bugs is presented as the mean of the agreement as percentage, with the standard deviation, providing a measure of confidence. The majority of the differences concern labels that PatchScope currently does not support, e.g. various refactoring-related labels.

Thus, we conclude that PatchScope achieves over 91% line-level annotation accuracy across both datasets. At the patch level, it correctly annotates an average of $(93 \pm 17)\,\%$ per patch in [3], and $(96 \pm 9)\,\%$ per patch in [5]. These results demonstrate PatchScope's high accuracy and satisfactory agreement across the analyzed datasets.

TABLE I: Evaluation Results for PatchScope Across Datasets

Dataset	Projects	Patches	Lines	Consensus	% Lines Agree	Patch Agree (%)	Time (s)
Herbold et al. [3]*	27(29)	3,179(3,519)	253,938(290,812)	3+ of 4	91.272 %	(93 ± 17) %	626.94 s
BugsInPy [4] (annotations [5])	17	496	60,194	2+ of 3	91.176 %	(96 ± 9) %	321.45 s

For two projects from Herbold et al. [3] ('wss4j' and 'santuario-java'), 340 commits (36,874 lines) are missing from git repository. Agreement is reported for the existing patches.

A. Execution time

PatchScope leverages Pygments for lexical analysis of changed lines, facilitating line type classification and annotation. Parsing accuracy depends on contextual information, such as preceding lines for multiline constructs (e.g., comments). To enhance precision, PatchScope analyzes the entire line range from the unified diff and, and when available (git repository): the complete file contents before and after changes. Parsing entire files incurs a performance overhead, as reflected in the execution times in Table I. Despite this, the tool performs reasonably, processing the dataset in 629 s for [3] and 321 s for [5].

V. Related work

In previous studies, repository annotation tools fell into two categories: (a) custom build feature generation tools, designed for a specific purpose or research direction [9], [10], [11], [12], [13] and (b) generic dataset creation tools, which require complex setups, such as working database instances [14], [15]. In this article we present PatchScope, an extensible tool that serves both purposes while retaining a simple setup.

Rabbit [9] has the goal of detecting bot accounts on GitHub. The underlying algorithm utilizes 6 features created from GitHub API events and XGBoost classification model. However, feature selection and engineering for similar learn to rank models can be automated utilizing our PatchScope.

DRMiner [10] detects refactorization in Dockerfiles via comparison of enhanced abstract syntax trees (E-AST) between two revisions of the same file. However, PatchScope can be used in similar cases to provide preprocessing even before AST construction.

CADV [11] and AnnotationSniffer [12] are Java annotation processing tools. CADV visualizes annotation usage per package, class and whole system to help with program comprehension by developers. AnnotationSniffer extracts annotation form source to prepare metric like frequency features, which can be used by other algorithms. Similar goals can be achieved utilizing PatchScope with custom rules for Java files.

CoaCor [13] is a framework that employs reinforcement learning to train a code annotation model for generating natural language descriptions of code snippets, thereby enhancing code retrieval effectiveness. By integrating a code retrieval model to provide feedback, CoaCor ensures that the generated annotations accurately represent code semantics, facilitating improved retrieval performance. Experimental results demonstrate that CoaCor's annotations significantly enhance retrieval accuracy, outperforming traditional methods. PatchScope simplifies annotation with configurable rules, potentially reducing the need for algorithm training step. Moreover, it could be integrated seamlessly into frameworks like CoaCor as a data source.

ETCR [14] infrastructure aids mining source code and corresponding code review via GitHub or Gerrit API. This tool was successfully used for review collection for CodeReviewer pre-trained model [16]. It helps automating tasks like code quality estimation, review generation and code refinement. ETCR stores data in Postgres database and allows access via the connected Elasticsearch engine.

SmartSHARK [15] consists of multiple tools, integrated with central MongoDB database to provide reproducible datasets for mining software repositories. Data can be retrieved from repositories via vcsSHARK and enriched with additional information, such as issue tracking, mailing list scraping, and software metrics calculation. This data can be accessed through a Django-hosted website, directly from the database, or via a dedicated ORM library (pycoSHARK for Python and jSHARK for Java).

In comparison to ETCR [14] and SmartSHARK [15], PatchScope stores all data as files, for simpler setup and easier scripting.

VI. Conclusion

PatchScope provides a modular and adaptable framework for analyzing software contributions, enabling project teams, researchers, and security analysts to derive detailed insights beyond traditional metrics. Its independently operable modules ensure flexibility, allowing users to customize components for diverse analytical needs. The tool delivers a granular view of contribution dynamics, supporting deeper understanding and informed decision-making for task assignments, security audits, and project evolution.

PatchScope has demonstrated potential in automating annotation workflows, improving bug localization, and aiding project monitoring and research on patch-based datasets. Its adaptability, as shown through throughput and accuracy trade-offs, allows users to optimize performance and precision based on specific requirements.

By bridging manual expertise with automated efficiency, PatchScope empowers researchers and practitioners to focus on high-value tasks. We believe PatchScope holds significant utility for both industry and academia, providing enhanced insights into software development practices. Future efforts will aim to expand its language support, improve scalability, add new analyses and visualizations, and refine its integration into diverse workflows, further amplifying its utility and impact.

References

[1] Pygments — a generic syntax highlighter written in Python, https://pygments.org/, 2024, version 2.18.0.

[2] GitHub Linguist, https://github.com/github-linguist/linguist, 2024, 'languages.yml' file version from Aug 31, 2024.

[3] S. Herbold, A. Trautsch, B. Ledel, A. Aghamohammadi, T. A. Ghaleb, K. K. Chahal, T. Bossenmaier, B. Nagaria, P. Makedonski, M. N. Ahmadabadi, K. Szabados, H. Spieker, M. Madeja, N. Hoy, V. Lenarduzzi, S. Wang, G. Rodríguez-Pérez, R. Colomo-Palacios, R. Verdecchia, P. Singh, Y. Qin, D. Chakroborti, W. Davis, V. Walunj, H. Wu, D. Marcilio, O. Alam, A. Aldaeej, I. Amit, B. Turhan, S. Eismann, A.-K. Wickert, I. Malavolta, M. Sulír, F. Fard, A. Z. Henley, S. Kourtzanidis, E. Tuzun, C. Treude, S. M. Shamasbi, I. Pashchenko, M. Wyrich, J. Davis, A. Serebrenik, E. Albrecht, E. U. Aktas, D. Strüber, and J. Erbel, "A fine-grained data set and analysis of tangling in bug fixing commits," *Empirical Software Engineering*, vol. 27, no. 6, p. 125, Nov. 2022.

[4] R. Widyasari, S. Q. Sim, C. Lok, H. Qi, J. Phan, Q. Tay, C. Tan, F. Wee, J. E. Tan, Y. Yieh, B. Goh, F. Thung, H. J. Kang, T. Hoang, D. Lo, and E. L. Ouh, "Bugsinpy: a database of existing bugs in python programs to enable controlled testing and debugging studies," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 1556–1560. [Online]. Available: https://doi.org/10.1145/3368089.3417943

[5] P. Przymus, M. Fejzer, J. Narebski, R. Wozniak, L. Halada, A. Kazecki, M. Molchanov, and K. Stencel, "HaPy-Bug – Human Annotated Python Bug Resolution Dataset," (under review).

[6] V. Sobreira, T. Durieux, F. Madeiral, M. Monperrus, and M. de Almeida Maia, "Dissection of a bug dataset: Anatomy of 395 patches from Defects4J," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2018, pp. 130–140. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8330203

[7] A. Trautsch, F. Trautsch, S. Herbold, B. Ledel, and J. Grabowski, "The SmartSHARK ecosystem for software repository mining," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 25–28.

[8] P. Przymus, "Project: Applications of grayscale data models in software vulnerabilities." https://radon.nauka.gov.pl/dane/profil/63e5f96bd968d56c86ce5917.

[9] N. Chidambaram, T. Mens, and A. Decan, "RABBIT: A tool for identifying bot accounts based on their recent github event history," in *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, D. Spinellis, A. Bacchelli, and E. Constantinou, Eds. ACM, 2024, pp. 687–691. [Online]. Available: https://doi.org/10.1145/3643991.3644877

[10] E. Ksontini, A. Abid, R. Khalsi, and M. Kessentini, "Drminer: A tool for identifying and analyzing refactorings in dockerfile," in *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, D. Spinellis, A. Bacchelli, and E. Constantinou, Eds. ACM, 2024, pp. 584–594. [Online]. Available: https://doi.org/10.1145/3643991.3644921

[11] P. Lima, J. Melegati, E. Gomes, N. S. Pereira, E. Guerra, and P. Meirelles, "CADV: A software visualization approach for code annotations distribution," *Information and Software Technology*, vol. 154, p. 107089, Feb. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922001987

[12] P. Lima, E. Guerra, P. Meirelles, L. Kanashiro, H. Silva, and F. F. Silveira, "A Metrics Suite for code annotation assessment," *Journal of Systems and Software*, vol. 137, pp. 163–183, Mar. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412121730273X

[13] Z. Yao, J. R. Peddamail, and H. Sun, "CoaCor: Code Annotation for Code Retrieval with Reinforcement Learning," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 2203–2214. [Online]. Available: https://doi.org/10.1145/3308558.3313632

[14] R. Heumüller, S. Nielebock, and F. Ortmeier, "Exploit those code reviews! bigger data for deeper learning," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik, and M. D. Penta, Eds. ACM, 2021, pp. 1505–1509. [Online]. Available: https://doi.org/10.1145/3468264.3473110

[15] F. Trautsch, S. Herbold, P. Makedonski, and J. Grabowski, "Addressing problems with replicability and validity of repository mining studies through a smart data platform," *Empir. Softw. Eng.*, vol. 23, no. 2, pp. 1036–1083, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9537-x

[16] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, and N. Sundaresan, "Automating code review activities by large-scale pre-training," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1035–1047. [Online]. Available: https://doi.org/10.1145/3540250.3549081