Employing Contribution and Quality Metrics for Quantifying the Software Development Process

Themistoklis Diamantopoulos, Michail D. Papamichail, Thomas Karanikiotis, Kyriakos C. Chatzidimitriou, and Andreas L. Symeonidis
(thdiaman,mpapamic,thomas.karanikiotis,kyrcha)@issel.ee.auth.gr,asymeon@eng.auth.gr
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki
Thessaloniki, Greece

ABSTRACT

The full integration of online repositories in the contemporary software development process promotes remote work and remote collaboration. Apart from the apparent benefits, online repositories offer a deluge of data that can be utilized to monitor and improve the software development process. Towards this direction, we have designed and implemented a platform that analyzes data from GitHub in order to compute a series of metrics that quantify the contributions of project collaborators, both from a development as well as an operations (communication) perspective. We analyze contributions in an evolutionary manner throughout the projects' lifecycle and track the number of coding violations generated, this way aspiring to identify cases of software development that need closer monitoring and (possibly) further actions to be taken. In this context, we have analyzed the 3000 most popular Java GitHub projects and provide the data to the community.

KEYWORDS

mining software repositories, contribution analysis, DevOps, GitHub issues, code violations

1 INTRODUCTION

The emergence of the open-source initiative has changed the way software is developed. Software development is now a collaborative process, taking place in online code hosting facilities, such as GitHub, Bitbucket or GitLab. And although software engineering has always relied on effective teamwork, today more than ever we are able to observe this process transparently and at such a massive scale. Indicatively, GitHub at the time of writing hosts more than 100 million repositories from more than 30 million developers[1].

In this collaborative context, software projects are the summation of different contributions, which are often not limited to those of specific team members, but extend also to those of outside collaborators. Collaboration often expands to multiple axes, including not only writing code, but also augmenting documentation, proposing the development of new features, discussing any issues raised by end-users, resolving bugs, etc. This information, is available in different formats (i.e. source code, commits, issues, pull requests, etc.), and tracks the whole timeline of the project.

Keeping track of these data is quite beneficial for various reasons. Apart from collaboration, versioning, backup/restore, etc., these data constitute what we may call the 'story' of the project, and can be harnessed to answer questions such as 'what issues have arisen?', 'who has worked on what?', 'how does heavy workload affects quality?', or even 'who is the most suitable developer for fixing this bug?'. This story is often considered as an integral part of DevOps, an agile-oriented methodology that combines software development (Dev) with operations (Ops) with the aim of building high quality software and reducing the time between production releases [2, 13]. One of the most important aspects of DevOps is the measurement of the software development process[2]. By doing so (and hence by answering questions like the ones mentioned above), one can increase the efficiency of product development [19].

Against this background, we have built a platform that crawls the infrastructure of GitHub, analyzes all contributions-related data (sourec code, commits, issues, contributors, etc.) in an evolutionary manner and calculates both contributions and quality related metrics that can be used to answer multiple research questions relevant to the software development process. Our platform currently includes the 3000 most popular Java projects of GitHub[3]. The constructed dataset contains all the analysis results regarding the crawled projects and is available as a MongoDB dump, thus allowing one click set-up and advanced querying capabilities.

2 ARCHITECTURE AND TOOLS

Figure 1 depicts the architecture of our platform, which comprises four modules: the *Data Downloader*, the *MongoDB Management System*, the *Contributions Analyzer*, and the *Quality Analyzer*. These modules are presented in detail in the following paragraphs.

Data Downloader. It is a Python application that uses the GitHub API[4] in order to retrieve all information offered for a given repository. This information includes *commits*, *issues*, *commit comments*, *issue comments*, *issue events*, *contributors information*, *repository information*, as well as the *source code* of the repository. Apart from downloading the data as raw .json files, the data downloader offers integration capabilities with MongoDB, which can be used for storing, retrieving and querying the data.

MongoDB Management System. We use MongoDB as our database management system, which contains all the raw data retrieved from GitHub along with the results of the contributions and the quality analysis. The schema of our database is shown in Figure 2. Data are organized in ten collections, each referring to a different type of information (commits, issues, statistics, etc.). In an effort to provide efficient data filtering/retrieval capabilities, the documents of all collections are connected through certain attributes

[1] https://octoverse.github.com/

[2] The others are collaboration, automation, and monitoring [14].

[3] Although our methodology is mostly agnostic, we use Java repositories as a proof of concept to account for commit metrics and code violations.

[4] https://developer.github.com/v3/

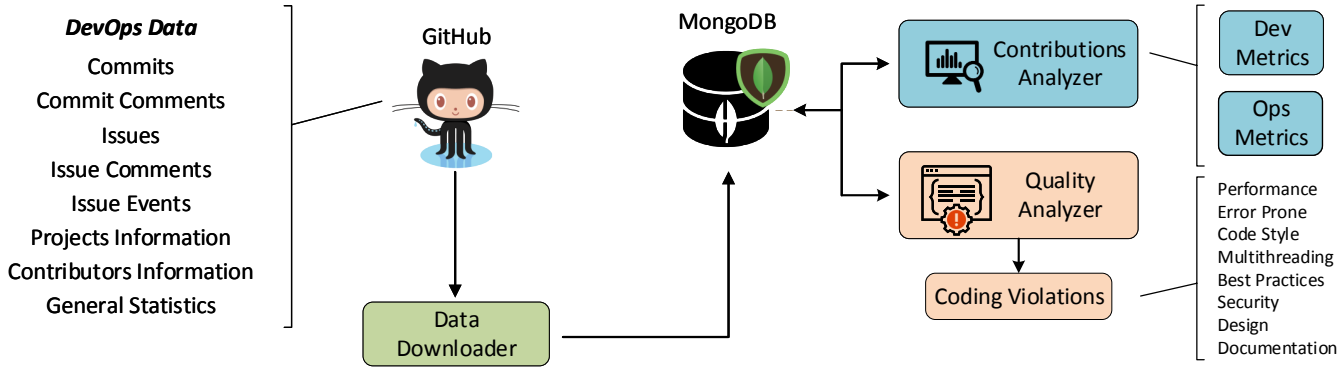

Figure 1: Architecture Overview

(the equivalent of foreign keys in relational database terms) such as the *repository name*, *the commit sha* etc. We have also created indexes for these attributes, to enable fast searching (e.g. searching by repository name in the commits collection that contains approximately 4M documents takes no more than 50ms on average). Our design choice for using MongoDB originates from the fact that it provides schema less architecture with powerful scaling capabilities along with robust querying service.

Contributions Analyzer. It uses the information retrieved from GitHub in order to compute a series of metrics that quantify the activity of each contributor both in terms of development and operations. Table 1 presents the calculated metrics along with their category (*Dev for "development"* or *Ops for "operations"*) and their computation interval (*Weekly* or *Full*). *Weekly* denotes that the respective metric is computed for every week of the project lifecycle (starting from its day one until the present day), while *Full* refers to metrics that are computed aggregatively taking into account all project data. The results are stored in the collection "*metrics*". Given that each contributor may contribute in more than one repositories, each document refers to the combination *contributor-repository*.

The rationale behind the selection of these metrics originates from their use in current research [9, 12, 15, 16, 20] from a development and operations perspective. For instance, several development-oriented metrics, such as commits_authored, file additions/deletions/modifications, etc., are often used to measure productivity [12] or even to predict defects [15]. Change burst metrics, such as change_bursts, average_burst_length, etc., have also been shown to be useful in this aspect [16]. Similar challenges can also be addressed using early and late contributions (early/late additions/deletions), which are the ones that belong in the first 20% and the last 20% of the project lifecycle, respectively (late activities are often proven to be quite different from early ones [16]). Operations-oriented metrics, such as the issues_opened, issues_closed, issues_participated (i.e. the issues that are created/deleted/modified by the developer as well as the ones in which he/she is mentioned by others), etc., are typically used to quantify the involvement, and generally interpret the profile characteristics, of the developer [9, 20].

Quality Analyzer. Effective monitoring of the software development process requires analyzing the quality of the final product. In an effort to account for the evolution of the product throughout its lifecycle, we perform static analysis on a weekly basis (we perform full analysis on the last commit of each week) in order to identify violations of widely accepted coding practices. We employ PMD[5] in order to identify violations that cover various source code properties (such as Performance, Code Style, Security, etc.). The tool assigns each identified violation with a priority from 1 (of utmost importance) to 5 (simple suggestion). Using these priorities, the reported violations are classified into three categories based on their impact: *Minor* (priorities 4 and 5), *Major* (priorities 2 and 3), and *Critical* (priority 1), before stored in the *Violations* collection. Finally, towards ensuring full traceability, for each identified violation, we store the full path of the respective source code file along with the line number.

Table 1: Contribution Metrics

Metric	Dev	Ops	Weekly	Full
change_bursts	×	–	–	×
average_burst_length	×	–	–	×
largest_burst_length	×	–	–	×
early_{additions, deletions}	×	–	–	×
late_{additions, deletions}	×	–	–	×
commits_authored	×	–	×	×
issues_{opened, closed}	–	×	×	×
issues_closed_per_day	–	×	×	×
issues_participated	–	×	×	×
average_comments_per_issue	–	×	×	×
average_issues_comments_length	–	×	×	×
average_time_to_close_issues	–	×	×	×
total_file_{additions, deletions}	×	–	×	×
total_file_{modifications, changes}	×	–	×	×
total_lines_of_code_changed	×	–	×	×
tot_{additions, deletions}	×	–	×	×
dev_activity_period_in_days	×	–	×	×
dev_inactive_period	×	–	–	×
ops_activity_period_in_days	–	×	×	×

[5] https://pmd.github.io/

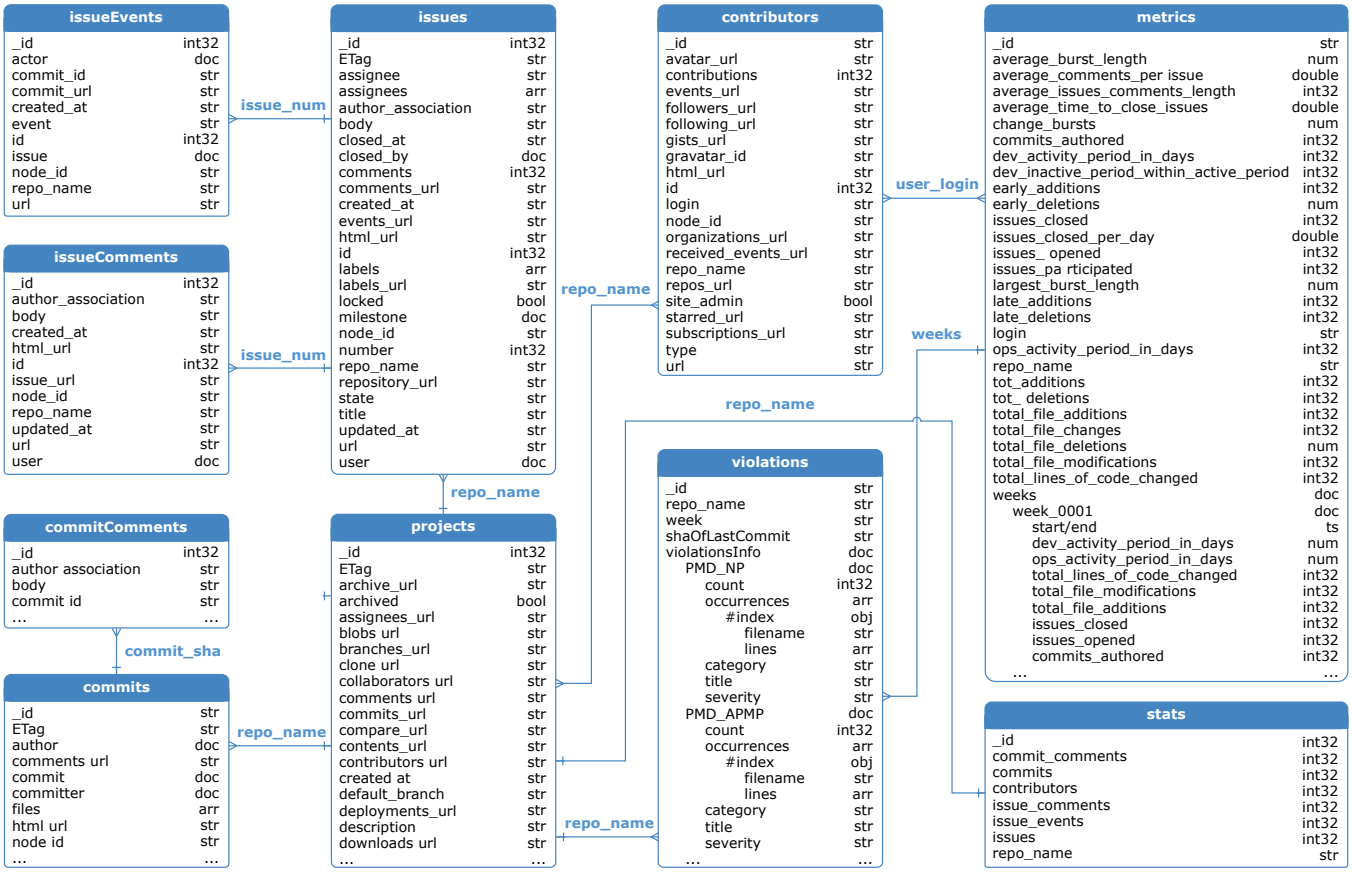

Figure 2: Database Schema Overview (not all connections are shown)

3 DATASET CONSTRUCTION

In a development community that is continuously moving towards component reuse, online software repositories constitute an integral part of the software development process. While this poses a series of challenges in terms of managing software development, it also provides a series of opportunities. These opportunities originate from the deluge of available data that enable quantifying the software development process and building effective evaluation models. Towards this direction, we extracted data residing in GitHub in order to construct a dataset that can be used for these purposes. Any challenges faced during this process are discussed in the following paragraphs.

Which projects to analyze. Our primary target was to create a dataset that covers a wide range of different development scenarios. To that end, we chose to analyze 3,000 projects, a number large enough to include projects that exhibit big differences in size, complexity, number of contributors, and duration. In an effort to be systematic in our analysis we used *popularity* (reflected by the number of stars) as our selection criterion and *Java* as the primary programming language.

Which metrics to compute. Monitoring the software development process is a non-trivial task that involves a series of parameters that need to be taken into consideration. Such parameters are the individual characteristics of each software project (inner structure, scope/area of application, time/resources constraints, etc.), along with the individual characteristics of each contributor (role in the project, collaboration in more than one projects, tendency to work in bursts, etc.). As a result, we resort in selecting 25 different metrics (see Table 1) that thoroughly describe the profile of each contributor both in terms of development-oriented and operations-oriented activities. On top of that, given the fact that a contributor can also change role within a project and thus a single value for a certain metric cannot fully reflect his profile over time, we perform a periodic analysis and compute each metric on a weekly basis.

Enable tailor-made dataset construction. One of the main targets of our approach is to provide a dataset that can be flexible enough to be used by researchers for elaborating on a wide range of research questions. To that end, we decided to facilitate the construction of tailor-made datasets by storing the data into a database that allows effective querying. This infrastructure along with the wide variety of stored information (metrics, statistics, etc.), regarding both repositories and contributors, enable effective data filtering based on the individual needs and objectives of each researcher. Two sample queries that can be used for dataset construction purposes are shown in Figure 3. The first filters repositories with certain

characteristics (number of contributors and commits), while the second targets at finding the occurrences of a certain violation over the lifecycle of a certain project.

```
// Repos with more than 20 contributors and 1,000 commits
db.getCollection('stats').find({
    "$and":[
        {"contributors": {"$gt": 20}},
        {"commits": {"$gte": 1000}}
    ]
})

// Get the total number of weeks a certain violation occurred in a project
db.getCollection('violations').aggregate([
    { "$match": { "repo_name": "the repository name" } },
    { "$match": { "violationsInfo.ViolationID": { "$exists": True } } },
    { "$group": { "_id": "null", "total": { "$sum": 1 } } }
])
```

Figure 3: Example queries

Finally, our dataset is available online in the form of a MongoDB data dump[6]. Certain statistics are shown in Table 2.

Table 2: Dataset Statistics

Metric	Value
Number of GitHub projects	3,000
Number of Commits	3,948,945
Number of Commit Comments	41,099
Number of Issues	819,031
Number of Issue Comments	1,898,101
Number of Issue Events	2,300,490
Number of Contributors	62,597
Lines of Code Analyzed	5,216,361,494
Total database size	125GB (19.8GB compressed)

4 IMPACT AND RESEARCH DIRECTIONS

Our dataset can be used to confront several challenges in current research. At first, given that it comprises various software process metrics along with the occurrence of multiple violations, it could be employed for extracting the behavior of the metrics and studying their co-evolution. For instance, one could study the correlation between issues and GitHub stars/forks [5], explore the effects of issue labeling on the time required to resolve them [11], or even determine whether resolving issues as early as possible is beneficial for the project [6]. Similar directions can be pursued for commits [4, 21]; e.g. commit-level metrics, such as the time between consecutive commits or the changes between releases, which can be used to determine the habits of project developers and align them to produce a better plan of actions [21] and/or quantify the impact of the design choices on the quality of the final product [8].

[6]http://doi.org/10.5281/zenodo.2556151

Developer behavior can also be analyzed in several other axes. Contribution metrics, such as the number of commits relevant to resolving bugs, the number of issue/commit comments and their distribution within the week (e.g. weekday or weekend), etc., have been proven useful for assessing the involvement of each developer [12] and for distinguishing among full-time contributors and volunteers in open source projects [17]. In the same context, one can also identify the roles of developers in a project (e.g. front-end engineer, back-end engineer, database engineer), by determining the source code components associated with each developer [10, 18]. Another novel idea in this field would be to extract higher level roles, indicating whether an engineer is more development-oriented (e.g. by focusing on commits, development activity, etc.) or more operations/process-oriented (e.g. by often participating on issues, exhibiting high response times, etc.), or even whether he/she achieves the DevOps sweet spot (i.e. an effective mixture of the above contribution patterns).

The integration of semantics can also produce interesting findings about the expertise of individual developers. Our dataset includes source code (e.g. commits), textual (e.g. commit/issue comments), and process (e.g. bursts, activity periods) data that can be mined to extract the areas of expertise of each developer [9]. This information can be then used to construct developer profiles [9] or even visual resumes [20] and find the best match for the team. In an even more practical context, one can also search within a project for "the right developer for the job". Research in the field of automated bug/feature assignment is broad and current methods use several metrics found in our dataset, including not only issue-related data (issue text, keywords) [1, 22], but also information about the acquaintance of developers with specific components [3] or even about their latest activity in the project [7].

5 CONCLUSIONS

Mining contribution data from GitHub can offer useful insights on the software development process and produce practical results that may be used to improve it. In this work, we have pursued this research direction by creating a platform that can be employed to produce useful metrics. Furthermore, we have proposed a set of metrics and built a large dataset that can be used to answer multiple research questions. As future work, we plan to augment our dataset, by adding more repositories and, most importantly, more metrics that can cover additional development scenarios, such as supporting various programming languages (e.g. Python, JavaScript). Finally, it would be interesting to study the co-evolution of these metrics with regard to metrics relevant to the quality of the projects.

REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who Should Fix This Bug?. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)* (Shanghai, China). ACM, New York, NY, USA, 361–370. https://doi.org/10.1145/1134285.1134336

[2] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective* (1st ed.). Addison-Wesley Professional, Reading, Massachusetts, USA.

[3] Pamela Bhattacharya, Iulian Neamtiu, and Christian R. Shelton. 2012. Automated, Highly-accurate, Bug Assignment Using Machine Learning and Tossing Graphs. *J. Syst. Softw.* 85, 10 (Oct. 2012), 2275–2292. https://doi.org/10.1016/j.jss.2012.04.053

[4] Marco Biazzini and Benoit Baudry. 2014. "May the Fork Be with You": Novel Metrics to Analyze Collaboration on GitHub. In *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics (WETSoM 2014)* (Hyderabad, India). ACM, New York, NY, USA, 37–43. https://doi.org/10.1145/2593868.2593875

[5] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *Proceedings of the 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, Washington, DC, USA, 188–197. https://doi.org/10.1109/ISSRE.2013.6698918

[6] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi. 2015. Exploring the use of labels to categorize issues in Open-Source Software projects. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE Computer Society, Washington, DC, USA, 550–554. https://doi.org/10.1109/SANER.2015.7081875

[7] Konstantinos Christidis, Fotis Paraskevopoulos, Dimitris Panagiotou, and Gregoris Mentzas. 2012. Combining Activity Metrics and Contribution Topics for Software Recommendations. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering (RSSE '12)* (Zurich, Switzerland). IEEE Press, Piscataway, NJ, USA, 43–46.

[8] WF Gonçalves, CB de Almeida, LL de Araújo, MS Ferraz, RB Xandú, and I de Farias Junior. 2017. The Impact of Human Factors on the Software Testing Process: The Importance of These Factors in a Software Testing Environment. *Journal of Information Systems Engineering & Management* 2, 4 (2017), 24.

[9] Gillian J. Greene and Bernd Fischer. 2016. CVExplorer: Identifying Candidate Developers by Mining and Exploring Their Open Source Contributions. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)* (Singapore, Singapore). ACM, New York, NY, USA, 804–809. https://doi.org/10.1145/2970276.2970285

[10] S. Li, H. Tsukiji, and K. Takano. 2016. Analysis of Software Developer Activity on a Distributed Version Control System. In *Proceedings of the 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE Computer Society, Washington, DC, USA, 701–707. https://doi.org/10.1109/WAINA.2016.107

[11] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu. 2018. Exploring the Characteristics of Issue-Related Behaviors in GitHub Using Visualization Techniques. *IEEE Access* 6 (2018), 24003–24015. https://doi.org/10.1109/ACCESS.2018.2810295

[12] Jalerson Lima, Christoph Treude, Fernando Figueira Filho, and Uira Kulesza. 2015. Assessing Developer Contribution with Repository Mining-based Metrics. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME '15)*. IEEE Computer Society, Washington, DC, USA, 536–540. https://doi.org/10.1109/ICSM.2015.7332509

[13] Mike Loukides. 2012. *What is DevOps?* O'Reilly Media, Inc., Champaign, IL, USA.

[14] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. 2015. Dimensions of DevOps. In *Proceedings of the Agile Processes in Software Engineering and Extreme Programming*. Springer International Publishing, Cham, 212–217.

[15] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. 2008. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the 30th International Conference on Software Engineering* (Leipzig, Germany) *(ICSE '08)*. ACM, New York, NY, USA, 181–190. https://doi.org/10.1145/1368088.1368114

[16] Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, and Brendan Murphy. 2010. Change bursts as defect predictors. In *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, IEEE Computer Society, Washington, DC, USA, 309–318.

[17] Saya Onoue, Hideaki Hata, and Ken-ichi Matsumoto. 2013. A Study of the Characteristics of Developers' Activities in GitHub. In *Proceedings of the 2013 20th Asia-Pacific Software Engineering Conference (APSEC '13) - Volume 02*. IEEE Computer Society, Washington, DC, USA, 7–12. https://doi.org/10.1109/APSEC.2013.104

[18] Michail D. Papamichail, Themistoklis Diamantopoulos, Vasileios Matsoukas, Christos Athanasiadis, and Andreas L. Symeonidis. 2019. Towards Extracting the Role and Behavior of Contributors in Open-source Projects. In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT)*. SciTePress, Prague, Czech Republic, 536–543. https://doi.org/10.5220/0007966505360543

[19] James Roche. 2013. Adopting DevOps Practices in Quality Assurance. *Commun. ACM* 56, 11 (Nov. 2013), 38–43. https://doi.org/10.1145/2524713.2524721

[20] A. Sarma, X. Chen, S. Kuttal, L. Dabbish, and Z. Wang. 2016. Hiring in the Global Stage: Profiles of Online Contributions. In *Proceedings of the 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*. IEEE Computer Society, Washington, DC, USA, 1–10. https://doi.org/10.1109/ICGSE.2016.35

[21] Yang Weicheng, Shen Beijun, and Xu Ben. 2013. Mining GitHub: Why Commit Stops – Exploring the Relationship Between Developer's Commit Pattern and File Version Evolution. In *Proceedings of the 2013 20th Asia-Pacific Software Engineering Conference (APSEC '13) - Volume 02*. IEEE Computer Society, Washington, DC, USA, 165–169. https://doi.org/10.1109/APSEC.2013.133

[22] X. Xia, D. Lo, X. Wang, and B. Zhou. 2013. Accurate developer recommendation for bug resolution. In *Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society, Washington, DC, USA, 72–81. https://doi.org/10.1109/WCRE.2013.6671282