ModelMine: A Tool to Facilitate Mining Models from Open Source Repositories

Sayed Mohsin Reza
University of Texas at El Paso
El Paso, Texas, USA
sreza3@miners.utep.edu

Omar Badreddin
University of Texas at El Paso
El Paso, Texas, USA
obbadreddin@utep.edu

Khandoker Rahad
University of Texas at El Paso
El Paso, Texas, USA
karahad@miners.utep.edu

ABSTRACT

Mining Software Repositories (MSR) has opened up new pathways and rich sources of data for research and practical purposes. This research discipline facilitates mining data from open source repositories and analyzing software defects, development activities, processes, patterns, and more. Contemporary mining tools are geared towards data extraction, analysis primarily from textual artifacts and have limitations in representation, ranking and availability. This paper presents ModelMine, a novel mining tool focuses on mining model-based artifacts and designs from open source repositories. ModelMine is designed particularly to mine software repositories, artifacts and commit history to uncover information about software designs and practices in open-source communities. ModelMine supports features that include identification and ranking of open source repositories based on the extent of presence of model-based artifacts and querying repositories to extract models and design artifacts based on customizable criteria. It supports phase-by-phase caching of intermediate results to speed up the processing to enable efficient mining of data. We compare ModelMine against a state-of-the-art tool named PyDriller in terms of performance and usability. The results show that ModelMine has the potential to become instrumental for cross-disciplinary research that combines modeling and design with repository mining and artifacts extraction.

URL: https://www.smreza.com/projects/modelmine/

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; **System description languages**; **System modeling languages**; **Unified Modeling Language (UML)**;

KEYWORDS

Mining Software Repositories, Model Mining, Software Engineering

ACM Reference Format:
Sayed Mohsin Reza, Omar Badreddin, and Khandoker Rahad. 2020. ModelMine: A Tool to Facilitate Mining Models from Open Source Repositories. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3417990.3422006

MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8135-2/20/10.
https://doi.org/10.1145/3417990.3422006

1 INTRODUCTION

Mining software repositories has witnessed tremendous growth in the past few years. Software repositories, its versions, and commit histories contain significant information about software development activities and contribute to establishing research agendas in software development, cost estimation, testing, and quality assurance [6, 18, 20]. With such research advancement, the impact of modeling, code smell, code reviews, prediction of change-prone classes become more influential in software engineering research [2, 14, 22].

Unfortunately, there is limited support for software engineering researchers who target mining models and design artifacts for system development from open source repositories [3, 15, 17]. Mining tools like PyDriller [21] and MetricMiner [20] can extract data primarily from either code or commit history, with limited support for mining non-textual artifacts. Current textual-based mining tools expose some key deficiencies when mining repositories and model artifacts. First, textual artifacts are relatively smaller in size compared to modeling artifacts. Second, models and design artifacts are much less prevalent than code in the majority of repositories. These deficiencies have limited the depth and breadth of the extractable data, and have consequently limited the scope of the possible research. As such, the ultimate goal of the paper is to propel research in software design and related practices by facilitating a tool that enhances data extraction for model-based artifacts.

This paper presents a novel model mining tool called ModelMine which facilitates mining models by analyzing model artifacts and repository metadata information. The tool brings some key benefits to researchers in software design and modeling. First, it enables the ranking of repositories based on the prevalence of model-based artifacts. Second, it enables faster data extraction for non-textual artifacts using a phased data pre-fetching approach. Third, it supports different filtering mechanisms to extract models from open source repositories without requiring extensive data mining knowledge or expertise.

To evaluate the usefulness of our tool, we compare our tool with the state-of-the-art PyDriller [21], a python framework in terms of execution time, memory consumption, cyclomatic complexity, and usability. Results show that ModelMine requires less time, memory, and can achieve similar results as PyDriller without having the coding knowledge. In addition to that, ModelMine can mine repositories and file artifacts which are unsupported in PyDriller.

The paper is organized as follows. We present related works in Section 2. Section 3 presents ModelMine architecture, followed by a demonstration of ModelMine user interfaces. We evaluate the tool compared against a well-known tool named PyDriller in Section 5. In the last section, we conclude the paper with future work ideas.

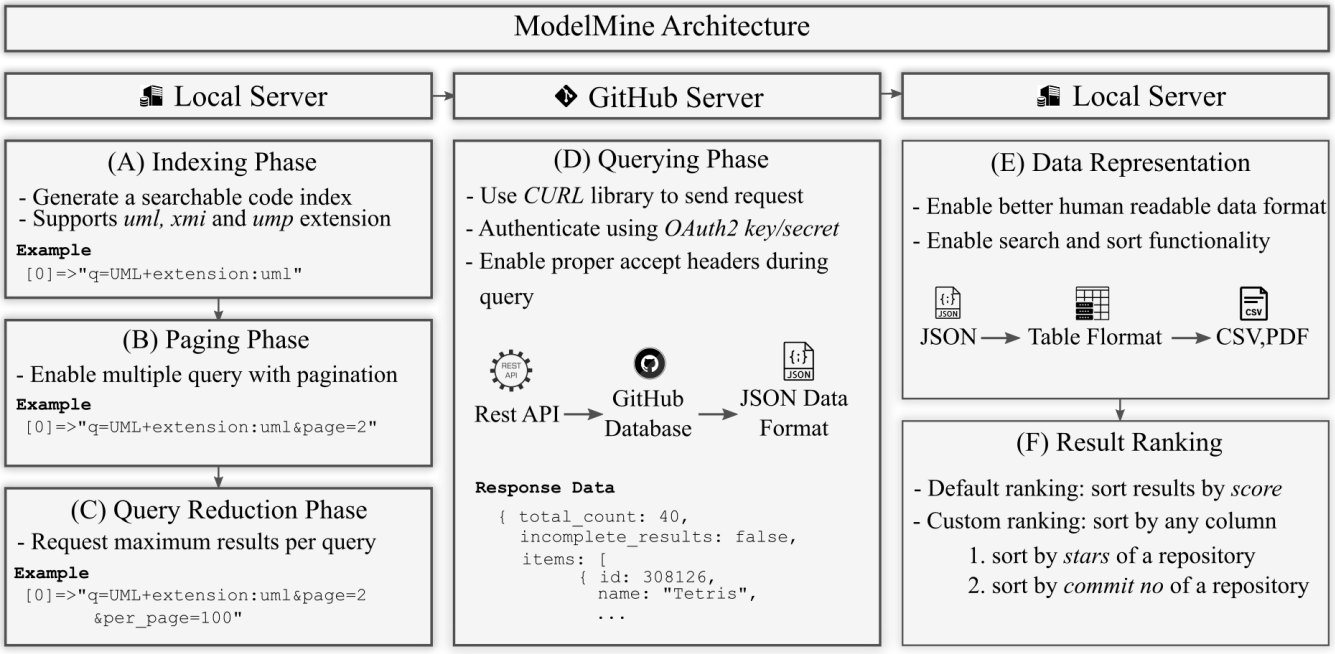

Figure 1: Architecture of ModelMine tool

2 RELATED WORK

The general idea of retrieving software-related data on demand is not new and can be related to Data as a Service (DaaS). These data are being collected from source code, metadata information, structured data, graphics files, etc. [11]. Some tools provide the static and dynamic process to collect metadata information as well as file structure. However, the process is not trivial to mine specific types of models from open source repositories. GHTorrent provides repositories of GitHub in a static way that was archived each year since 2013 [10]. As a public repository of software repositories, it provides cross-domain analysis on metadata information of repositories [10, 11]. A study by Goes [9] in 2014 showed that Big Data concepts in software repositories such as GHTorrent are characterized by 5-Vs (Volume, Variety, Velocity, Value, and Veracity). However, it limits research on current data files and commit history [4][13]. This issue was also mentioned by Gousios et al [10]. The author mentioned that the advantage of GHtorrent in terms of repository information that are mostly static helps to analyze software structure and basic metadata information not dynamic attributes such as recent commits, comments, or issues [10].

Several mining tools have emerged to enable such research and discovery. For example, PyDriller, a python framework for mining software repositories can extract recent information from open source repositories such as commits, developer information, modifications, differences, and source codes [21]. MetricMiner is another cloud-based application suitable for mining software repositories for metrics calculation, data extraction, and statistical inference [20]. These tools focus on extracting data primarily from either code or commit history, with limited support for mining non-textual artifacts.

Model mining from public repositories can be related to software quality, maintainability research to analyze when model files are created, how those files are maintained, and sustained throughout the project life cycle. A study conducted by Gregorio et al. [19] reported that creating a model-based dataset helps to study the impact of specific extension models on the code structure and how software quality, productivity are changing throughout the project life cycle. A study by Noten et. al. [13] showed that such dynamic model files allow one to analyze recent data on repositories, artifacts or commit information to maintain software quality.

In this study, we identify existing problems related to MSR and develop a comprehensive tool, ModelMine that provides mining model repositories, artifacts and commit history in one platform.

3 MODELMINE ARCHITECTURE

In this section, we discuss the architecture that we use to build ModeMine. The tool adopts a six-phased approach (indexing, paging, query reduction, querying, data representation, and results ranking) to mine model-based repositories, artifacts and commit history from open source repositories. The architecture of the tool is visualized in Figure 1 and the details are provided in the following subsections.

3.1 Indexing Phase

In this phase, ModelMine processes model extensions to generate a searchable code index. The tool supports several types of code extensions (uml, xmi, ump, and sysml, etc.) which are required to index in a search query. To get proper search results for the model file extension, we create indices one by one if there are multiple file extensions. A sample query indices are shown in Figure 1(A).

3.2 Paging Phase

In this phase, ModelMine tool works on paging the results. Since generating more results require more time and server load, we introduce the paging concept to limit the number of response results. If a researcher wants to generate more results, ModelMine tool allows them to perform that with the cost of time. Without pagination, the researcher gets maximum set limit results. To request further results, ModelMine adds a new query parameter named *page*. A sample example is shown in Figure 1(B) with paging concept.

3.3 Query Reduction Phase

In this phase, we implement a technique to overcome the issue limit per request. To process each request, ModelMine tool applies a query reduction technique which enables a request to have a maximum number of results set by the administrator. Such a limit is very important during the development of any mining tools [16]. However, to get a maximum number of results per query, ModelMine introduces a new query parameter named *per_page* and it reduces approximately 70% requests to get more results per request. A sample example is shown in Figure 1(C).

3.4 Querying Phase

In this phase, ModelMine prepares the queries created in last three phases. Two important steps are required to secure the server. First, authentication and second, blocking too many requests within a short period of time. In authentication, ModelMine tool uses $OAuth2 - key/secret$ technique [8] to protect the tool from unauthenticated users. For the second part, we introduce middleware to block multiple requests from the same IP within a short period of time [12]. A sample example is shown in Figure 1(D).

3.5 Data Representation

Data representation is the phase where ModelMine prepares the resulted data and its format of presentation. In the background, ModelMine represents the response in JavaScript Object Notation (JSON) format but for users, the responses are presented as a human-readable format (table, csv, excel, pdf). Additionally, ModelMine provides search and sort operations to maximize user satisfaction on data presentation.

3.6 Results Ranking

The ranking of the results is used to position the responses on a scale. To ensure a better result ranking system in ModelMine, we use score concept to serialize the results. This default score system represents the relevance of a search item relative to the other items in the result set. However, this ranking system can be changed to repository popularity, watchers, etc.

4 MODELMINE USER INTERFACE

In this section, we discuss user interface features and the mining process in ModelMine. The tool provides a simple extensible user interface to mine software repositories, artifacts, and commit history. The tool has three unique user interface features: (1) repository search, (2) artifacts search, and (3) commit history search to ensure the possibility of mining different types of dataset for MSR research.

4.1 Model-based Repository Search

A typical first exploratory step involves searching for repositories with prevalent modeling artifacts. Researchers can query code repositories with any model file extensions (i.e. uml, xmi, ump etc.) as shown in Figure 2. In addition to that, ModelMine allows additional criteria to query for repositories. Additional criteria include repository size, popularity, primary programming language, and repository timestamped information.

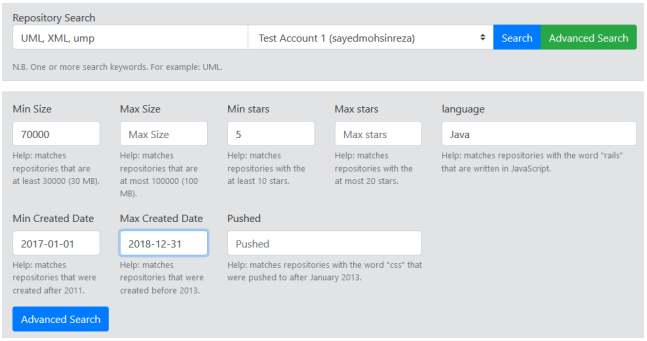

Figure 2: Model-based repository search

4.2 Model-based Artifact Search

ModelMine provides another user interface to search model-based files in open source repositories. Figure 3 visualizes the user interface for this operation where researchers can search model files with extensions (i.e. uml, xmi, etc.). In addition to the extension, the user can also provide a basic string that may exist inside the model file metadata.

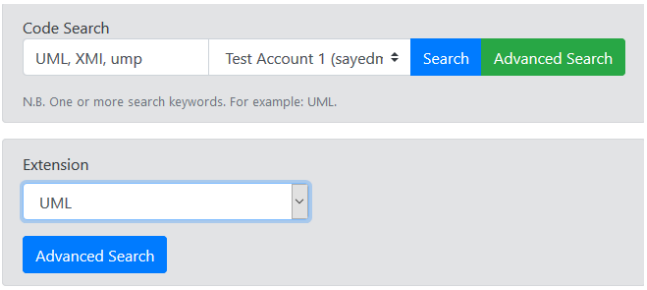

Figure 3: Model-based artifact search

4.3 Model-based Commit Search

Once a subset of repositories are identified using repository search feature described in Subsection 4.1, researchers are able to search for commit history. In a repository, there are multiple types of files and ModelMine allows researchers to investigate specific extension-based commit search. This feature allows researchers to analyze the version of the files of repositories as well as the behavior of software code updates of specific file extensions for different projects [13]. The user interface for this operation is visualized in Figure 4.

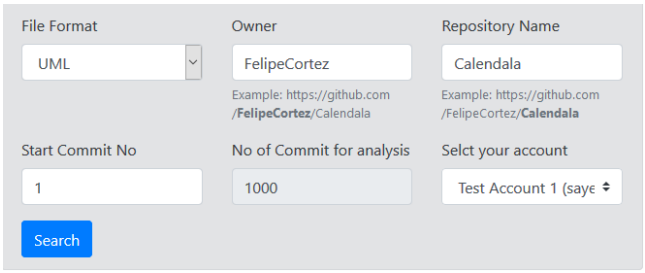

Figure 4: Model-based commit search

5 EVALUATION

The evaluation focuses on two dimensions; performance and usability. The performance dimension is motivated by the fact that model-based artifacts are much rarer in repositories compared to code, and tend to be significantly larger in size. This often translates to computational complexity in identifying and extracting model-based artifacts. The usability dimension is motivated by the targeted audience; software design practitioners and researchers who are not necessarily competent in data mining and data extraction.

For reference, we compare ModelMine with PyDriller [21], a well-established Python framework for mining software repositories. We identify five unique tasks that are common for the majority of MSR research with available support in both mining tools. The tasks are as follows:

(1) **Task 1 (Size related):** Retrieve the list of repositories that include at least one model artifact developed in UML and the repository size is larger than 30 MB.
(2) **Task 2 (Time related):** Retrieve the list of repositories that include at least one model artifact developed in UML and the repository was created between January 2019 and December 2019.
(3) **Task 3 (File property related):** Retrieve the list of artifacts with *.uml* file extension.
(4) **Task 4 (Commit related):** Retrieve the list of commits with a model artifact and the repository has at least one model artifact.
(5) **Task 5 (File property + commit related):** Retrieve the list of commits with any model artifacts (any model-based file extension) and the repository has at least one model artifact.

These tasks are implemented using both frameworks: (1) ModelMine and (2) PyDriller. To compare the frameworks , we use two different types of metrics: (1) performance & (2) usability metrics. Such evaluation metrics are used in the evaluation of different software mining repositories tool [7, 21]. The reason behind evaluation is to check how easy the tool is to learn and whether the results are provided in a meaningful way or not. The metrics we used to evaluate our tool are provided in Table 1.

In performance and usability study, we select ten participants who are actively working in software engineering research. Eight of them are doctoral students and two of them are master's students in computer science. We request participants to perform an usability study on ModelMine and PyDriller and fill up a questionnaire that collects all the usability metrics. The performance

Table 1: Evaluation metrics for task's

No	Metric Type	Metric Name	Subcategory	Shorthand	Unit
1	Performance Metrics	Execution Time		ET	Second
2		Max Memory		MM	Kilobytes
3		Cyclomatic Complexity		CCOM	N/A
4	Usability Metrics	Subjective Satisfaction	User Interface	UI	Rating (1-5)
5			Learning Curve	LC	
6			Data Visualization	DV	
7			Error Reporting	ER	

metrics are collected when the participants used the tools to do the evaluation. The details of the evaluation are available online at https://www.smreza.com/projects/modelmine/eval/. The results of the performance analysis of each task are shown in Table 2.

The result clearly shows that PyDriller takes more time and memory than ModelMine. The reason behind this result is that PyDriller fetches the whole git file of a selected repository and then mine the commit information. But ModelMine fetches the information directly without downloading any file. As there is no intermediate process, ModelMine takes less time and memory. Note that, the current version of PyDriller [21] is unable to search repositories (Task 1 & 2) or artifacts (Task 3).

Table 2: Performance evaluation results

Tasks	Metrics	ModelMine	PyDriller
Task 1 (Size)	ET	0.793 s	Not supported
	MM	701 KB	
	CCOM	5	
Task 2 (Time)	ET	0.675 s	Not supported
	MM	698 KB	
	CCOM	5	
Task 3 (Property)	ET	0.633 s	Not supported
	MM	611 KB	
	CCOM	4	
Task 4 (Commit)	ET	1.422 s	3.8s
	MM	630 KB	24220 KB
	CCOM	4	5
Task 5 (Composite)	ET	1.508 s	9.3 s
	MM	660 KB	24616 KB
	CCOM	4	4

In usability study, we adopt a framework proposed by Altalhi et al. [1] where usability analysis is designed based on a list of questionnaires presented to participants. Table 3 lists the questions and their related category. Similar usability studies were used to analyze overall tool ratings as well as task-based user satisfaction ratings [1, 5].

Table 3: Usability study questionnaires

Category	Questions
User Interface	(1) How easy are the tools to navigate? (2) How clearly the tools provide results in a meaningful way?
Learning Curve	(3) How easy are the tools to learn?
Data Visualization	(4) How well does the tool about presenting the data and modeling results?
Error Reporting	(5) How relevant is the error reporting?

Participants are required to answer the given questions for each task with a scale from 1-5 to express their usability responses. The

lower value represents less usability for each criterion. Figure 5 shows the usability results of ModelMine compared with PyDriller.

The results clearly show that ModelMine has better usability in all usability criteria than PyDriller. In user interface & learning curve category, ModelMine has 50% more ratings than PyDriller. The evaluation study also asks participants to provide comments for their ratings. One participant mentions that ModelMine provides a platform to mine data without worrying about the data collection part. Most of the MSR tools need extensive tasks to mine and have limitation in data preparation, ranking and availability. ModelMine has incorporated most of them to enrich mining model experience from open source repositories. Another participant comments that ModelMine provides faster learning experience than PyDriller due to its easy UI design and better readability.

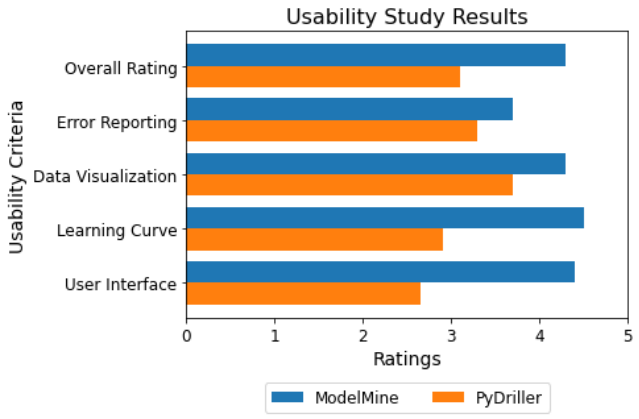

Figure 5: Usability study results

6 CONCLUSION

Prevalent mining tools are geared towards textual artifacts and tend to exhibit poor performance in mining models and software designs. Model-based artifacts are much less prevalent and tend to be much larger in size when compared to codes and other textual-based artifacts. This paper presents ModelMine, a tool crafted to facilitate mining models and designs from open source repositories. It enables the ranking of repositories based on the presence of designs and models, and uses phased data pre-fetching to enhance performance and broaden the scope for the mining processes. To evaluate our tool, we compare with state-of-the-art tool PyDriller and perform an usability and performance analysis with ten graduate students. Results show that ModelMine performs better than PyDriller in both analysis. The reported results demonstrate a significant potentiality of ModelMine in MSR research. ModelMine fills an important gap in the repository mining landscape, and aims at realizing research that combines data mining and data extraction from open source repositories.

ModelMine tool can be extended with image-based model mining feature. We plan to mine image-based model files (i.e png, jpg, jpeg, etc.) which will enrich MSR research. We also plan to ensure the best search results by analyzing text inside those image-based model files to understand the model semantically.

REFERENCES

[1] Abdulrahman H Altalhi, José María Luna, MA Vallejo, and Sebastián Ventura. 2017. Evaluation and comparison of open source software suites for data mining and knowledge discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 3 (2017), e1204.

[2] Omar Baddreddin and Khandoker Rahad. 2018. The impact of design and UML modeling on codebase quality and sustainability. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. 236–244.

[3] Omar Badreddin, Rahad Khandoker, Andrew Forward, Omar Masmali, and Timothy C Lethbridge. 2018. A decade of software design and modeling: A survey to uncover trends of the practice. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 245–255.

[4] Shahabedin Bayati and Arvind K Tripathi. 2016. Designing a Knowledge Base for OSS Project Recommender System: A Big Data Analytics Approach. (2016).

[5] Nigel Bevan. 2008. Classifying and selecting UX and usability measures. In *International Workshop on Meaningful Measures: Valid Useful User Experience Measurement*, Vol. 11. 13–18.

[6] Krishna Kumar Chaturvedi, VB Sing, and Prashast Singh. 2013. Tools in mining software repositories. In *2013 13th International Conference on Computational Science and Its Applications*. IEEE, 89–98.

[7] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 422–431.

[8] Daniel Fett, Ralf Küsters, and Guido Schmitz. 2016. A comprehensive formal security analysis of OAuth 2.0. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1204–1215.

[9] Paulo B Goes. 2014. Editor's comments: big data and IS research. (2014).

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

[11] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *Proceedings of the 11th working conference on mining software repositories*. 384–387.

[12] Piyush Maheshwari, Hua Tang, and Roger Liang. 2004. Enhancing web services with message-oriented middleware. In *Proceedings. IEEE International Conference on Web Services, 2004.* IEEE, 524–531.

[13] Jeroen Noten, Josh GM Mengerink, and Alexander Serebrenik. 2017. A data set of OCL expressions on GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 531–534.

[14] Fabio Palomba, Annibale Panichella, Andy Zaidman, Rocco Oliveto, and Andrea De Lucia. 2017. The scent of a smell: An extensive comparison between textual and structural smells. *IEEE Transactions on Software Engineering* 44, 10 (2017), 977–1000.

[15] Mithun Chandra Paul, Suman Sarkar, Md Mahfujur Rahman, Sayed Mohsin Reza, and M Shamim Kaiser. 2016. Low cost and portable patient monitoring system for e-Health services in Bangladesh. In *2016 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 1–4.

[16] Mohammad Masudur Rahman and Chanchal K Roy. 2014. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. 364–367.

[17] Sayed Mohsin Reza, Md Mahfujur Rahman, and Shamim Al Mamun. 2014. A new approach for road networks-A vehicle XML device collaboration with big data. In *2014 International Conference on Electrical Engineering and Information & Communication Technology*. IEEE, 1–5.

[18] Sayed Mohsin Reza, Md Mahfujur Rahman, Md Hasnat Parvez, M Shamim Kaiser, and Shamim Al Mamun. 2015. Innovative approach in web application effort & cost estimation using functional measurement type. In *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*. IEEE, 1–7.

[19] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel RV Chaudron, and Miguel Angel Fernandez. 2017. An extensive dataset of UML models in GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 519–522.

[20] Francisco Zigmund Sokol, Mauricio Finavaro Aniche, and Marco Aurélio Gerosa. 2013. MetricMiner: Supporting researchers in mining software repositories. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 142–146.

[21] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 908–911.

[22] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2018. Review participation in modern code review: An empirical study of the Android, Qt, and OpenStack projects (journal-first abstract). In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 475–475.