MCRMiner: a Tool to Support Empirical Studies on Modern Code Review

Igor Ferreira and Ingrid Nunes

[1]Instituto de Infomática, Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil

{ipferreira,ingridnunes}@inf.ufrgs.br

Abstract. *Modern Code Review (MCR) has been increasingly adopted in the industry and open source development. It is a lightweight code review process, being asynchronous and tool-supported. Due to the use of tools and their underlying databases, much data associated with MCR has become available, leading to many research studies to understand and improve this process. The first step in these studies typically requires learning and accessing the API of an MCR repository, and exporting it to a particular format to be analyzed. In this paper, we introduce a tool, named MCRMiner, which automates this step to ease the conduction of empirical studies in the context of MCR. The framework provides a graphical interface and the infrastructure needed to access an MCR repository, import its data, and export them to formats typically used to process this kind of data. MCRMiner allows accessing repositories from Gerrit, an MCR tool, but is implemented as a framework so that it can be instantiated to work with other MCR tools.*

1. Introduction

To improve software quality, many static verification techniques can be performed. Code review is one of them, which gained popularity with the inspection process proposed by Fagan [Fagan 1986] to minimize the existence of software problems such as defects, low maintainability, and high coupling. Code inspections are performed following a formal process composed of a specific set of phases that must be followed and checklists to analyze the code, with evidence of being effective [Johnson 1998]. As the software development process evolved, with a growing number of developers working on the same project, possibly in different locations, the formal process became less well-suited for the new reality and, likewise, underwent transformations, leading to the so-called Modern Code Review.

Modern Code Review (MCR) [Bacchelli and Bird 2013] is a lightweight and asynchronous process, in which authors submit their work for being reviewed employing supporting tools. Reviewers are invited to contribute, who can vote accepting or rejecting the code change as well as make comments, e.g. reporting bugs or suggesting code improvements. The main original motivation for adopting code review was the early detection of bugs. However, MCR has been reported to provide other benefits, such as improving code maintainability and legibility, knowledge sharing, and learning.

MCR supporting tools store code review data in repositories containing, e.g., code changes, authors, reviewers, and comments. Based on these data, many empirical studies, e.g. [Yang et al. 2016, Santos and Nunes 2017, Thongtanunam et al. 2015],

have been carried out to understand the outcomes of MCR, what influences them, and lessons learned that can be used to improve MCR. The first step in these studies requires extracting data from an MCR repository and exporting them to a particular format to be analyzed. Although there are APIs to collect data from MCR tools, there is a need for learning such APIs, implementing code to access it, which is a recurrent activity that can be automated reducing the effort of performing studies based on MCR data.

In response, in this work, we present *MCRMiner*, a tool to support the analysis of MCR data through the automation of its extraction. The tool provides a graphical user interface that allows accessing and importing data from repositories of Gerrit[1], a widely known MCR supporting tool. Imported repositories can be exported to particular formats, such as comma-separated values (CSV) files, which can be easily loaded to be mined in many programming languages and tools. It also provides basic statistics of imported project repositories, which are useful to decide whether a project is suitable for a particular study. MCRMiner is implemented as a framework, so that it can be extended to import data from other MCR supporting tools, e.g. Review Board[2], CodeFlow[3], Crucible[4] and GitLab[5]. We show, as an evaluation, how the tool performs while importing data from a set of open-source projects, demonstrating that MCRMiner facilitates research on MCR by eliminating the manual steps required to gather MCR data.

In the next section, we introduce existing tools to support MCR and discuss how they are related to MCRMiner. Then, in Section 3, we overview our tool and describe its architecture, including used technologies and its domain model. In Section 4, the MCRMiner features are introduced in more detail. Section 5 shows our tool in action, with details of its use to obtain data of different open software projects. Finally, we present conclusions in Section 6.

2. Related Work

Gerrit and ReviewBoard, mentioned in the introduction, are code review systems to support the MCR process, which store MCR data that can be extracted. To analyze these data, there are two existing tools, which are discussed in this section.

BugTracking [Rodríguez-Pérez et al. 2016] is a tool whose purpose is to help researchers in the process of analysing registered issues in issue-tracking systems and identifying those that correspond to bug reports. Datasets containing commits that are bugs are used for a wide range of purposes, such as building bug predictors [Araújo et al. 2017]. BugTracking gathers not only information from issue-tracking systems to help make such a classification, but also discussions from code review systems. Although it handles code review data, as MCRMiner, it does not extract data and make them available in a suitable format for posterior analysis.

With a goal similar to ours, that is, the goal of supporting researchers that need to mine MCR data, *Review Data Analyzer* (ReDA) [Thongtanunam et al. 2014] has been proposed. It is a web-based visualization tool to help visualize and understand code re-

[1] `https://www.gerritcodereview.com/`
[2] `https://www.reviewboard.org/`
[3] `https://www.getcodeflow.com/`
[4] `https://www.atlassian.com/software/crucible`
[5] `https://about.gitlab.com/`

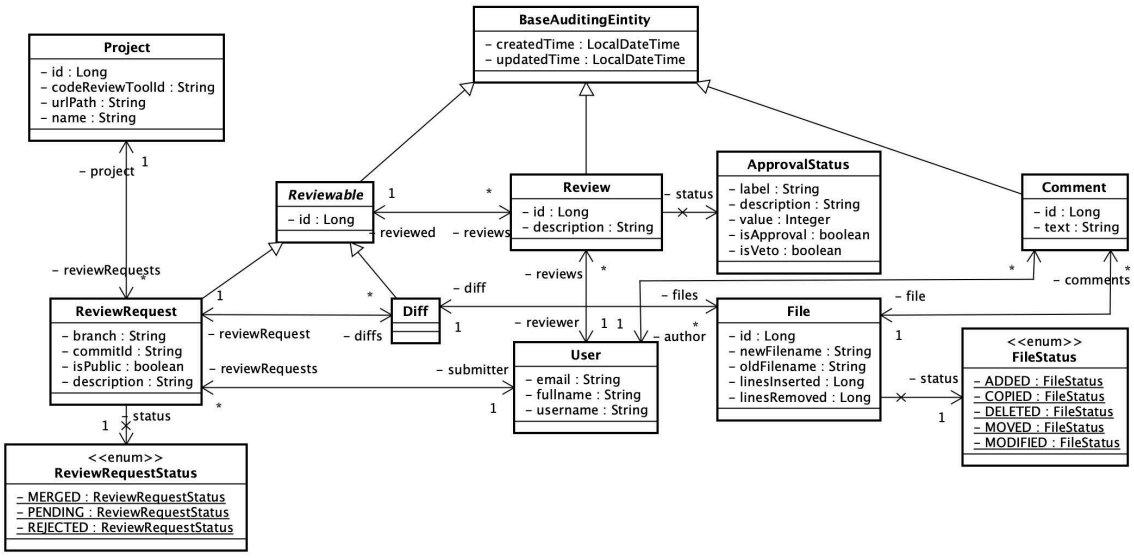

Figure 1. Domain Model.

view data, providing three visualizations: review statistics, activity statistics, and contributor activities. In its current version, ReDA uses data extracted from the Android Open Source Project (AOSP), extracted from Gerrit, i.e., it neither provides a means of importing data from other projects, nor exports them to files that can be loaded and mined by, e.g., R (a software environment for statistical computing and graphics). ReDA can be used in a complementary way to MCRMiner because the latter extracts data while the former ease their analysis using visualizations.

3. MCRMiner Overview and Design

MCRMiner is a tool to support researchers by obtaining MCR data by means of a graphical user interface. The tool has three main features: (i) repository importing; (ii) repository exporting; and (iii) basic statistics. It is implemented as a framework, with hotspots implemented with design patterns, so that it can be instantiated to code review systems other than Gerrit (our current implementation) by the extension of classes that are part of the core of the framework. We next detail the MCRMiner domain model and how the framework (and its instantiation) is structured.

3.1. Domain Model

In order to provide MCRMiner with the flexibility to be instantiated to different code review systems, we analyzed the REST APIs (and their documentation) of Gerrit and Review Board. This allows the data obtained from different code review systems to be mapped to our common domain model. Extracted data can then be an instance of our domain model and persisted in the MCRMiner database.

In Figure 1, we present the MCRMiner domain model. A project can have multiple review requests, which are submitted by a user and has a status. A request is also associated with a "diff", which corresponds to a code change. A diff is associated with a set of files, with information about the code change.

Both review requests and diffs can be reviewed. This is to accommodate reviews from both our analyzed code review systems. Reviews can have a description and an

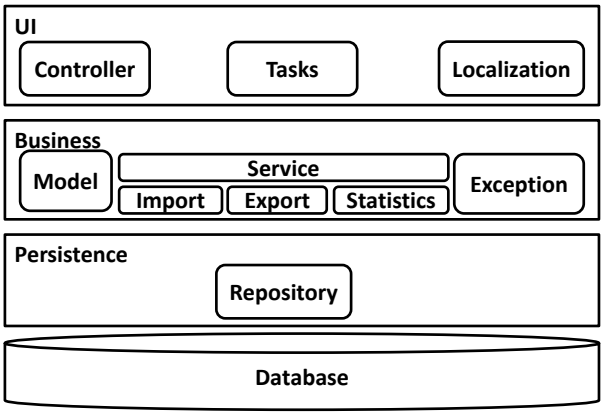

Figure 2. MCRMiner Architecture.

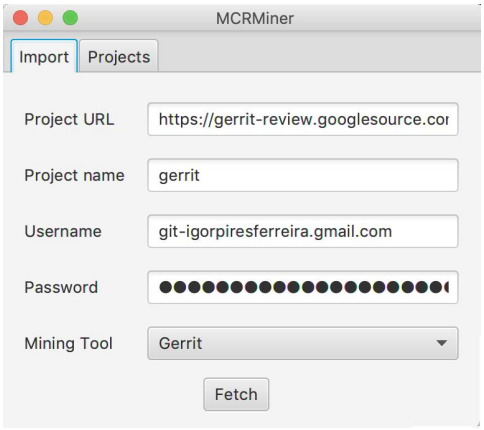

Figure 3. Repository Import.

approval status, and are associated with *reviewable* entities and a user who made the review. As part of the review, users can add comments to files.

3.2. Architecture and Adopted Technologies

The MCRMiner architecture is structured using a typical three-layered structure, as can be seen in Figure 2. The adopted programming language is Java 8 and Spring[6] was used as the main framework. We also used Gradle[7] to manage build automation tasks and application dependencies. In addition, several unit and integration tests were developed with the support of the JUnit and Mockito tools. In order to control the quality and stability of the project builds, we used the Travis CI[8] server.

The graphical user interface through which end users can use the MCRMiner features is implemented in the *UI* layer. End-user operations are received in this layer which calls services from the business layer. The UI layer was built upon the JavaFX[9] and JavaFX Scene Builder technologies. The latter allows applications to be built using visual aids that increase developer productivity.

The *Business* layer is composed of six modules. The model module implements our domain model, while the exception module has the implementations of exceptions that can be thrown by the different services offered in the business layer. The service module is a *Facade* to the core MCRMiner services, namely import, export, and statistics. In the import module, there is the logic for importing data from code review systems. The import module makes extensive use of inheritance and polymorphism so that interfaces can be implemented and abstract classes extended to different code review systems, being a hotspot of our framework. Details of how to instantiate this and the other framework hotspots can be seen elsewhere [Ferreira 2018]. We already provide an implementation for Gerrit, using its REST API, accessed using the gerrit-rest-java-client[10] open source library.

The export module implements the export feature of imported data under different

[6] https://spring.io/
[7] https://gradle.org/
[8] https://travis-ci.org/
[9] http://javafx.com/
[10] https://github.com/uwolfer/gerrit-rest-java-client

perspectives of the code review process, while the statistics module is responsible for calculating the statistics of a given imported project. The export and statistics modules also include hotspots, so that additional data can be exported and metrics can be added, respectively. Therefore, MCRMiner is extensible and customizable.

Finally, the *persistence* layer is responsible for all logic related to data access and the object-relational mapping. For its implementation, the data module of the Spring framework was used. The current MCRMiner implementation uses the H2[11] database. However, other databases can be used as long as they are a relational database.

4. MCRMiner Features

The previous section introduced MCRMiner and overviewed its architecture. We now detail the features provided by our tool, which are repository import, repository export, and project statistics.

4.1. Repository Import

The import feature, depicted in Figure 3, allows importing data from project repositories stored in code review systems. With it, it is possible to gather information about a particular project, such as all the patches submitted for review along with their modified files and the feedback submitted by the reviewers. As aforementioned, the current MCRMiner implementation is able to retrieve data only from Gerrit.

To import a project, users must provide its URL along with a username and password to access it. They must also assign a name to the project to be used within the scope of MCRMiner. After providing the required data, users can request to fetch the project.

4.2. Repository Export

Imported projects that are stored in MCRMiner can be visualized in the export feature, as can be seen on the left-hand side of Figure 4. When selecting a particular project, users can export it and visualize its statistics (detailed in the next section).

The export feature allows the imported data from code review systems be exported using one of five different possible perspectives: *file*, *author*, *comment*, *reviewer* and *reviewable*. Exported data is given in a text file in which each line corresponds to an entity of the selected perspective. Then, each line contains a set of values, separated using a selected separator. Therefore, if a comma is used as a separator, the export file is in the CSV format; and if a tab is used as a separator, the export file is in the tab-separated values (TSV) format. As said, this type of file can be loaded in programming languages and tools easily (i.e. with a single command).

The values of each line correspond to attributes of the selected entity (perspective) as well as its relationships. When there is a 1:1 relationship, it is also possible to include attributes of the related entity. When there is a 1:N relationship, it is possible to include attributes in an aggregated form (such as a counter or a sum). Before exporting data, users can select the values to be included in the exported file.

[11] `http://www.h2database.com`

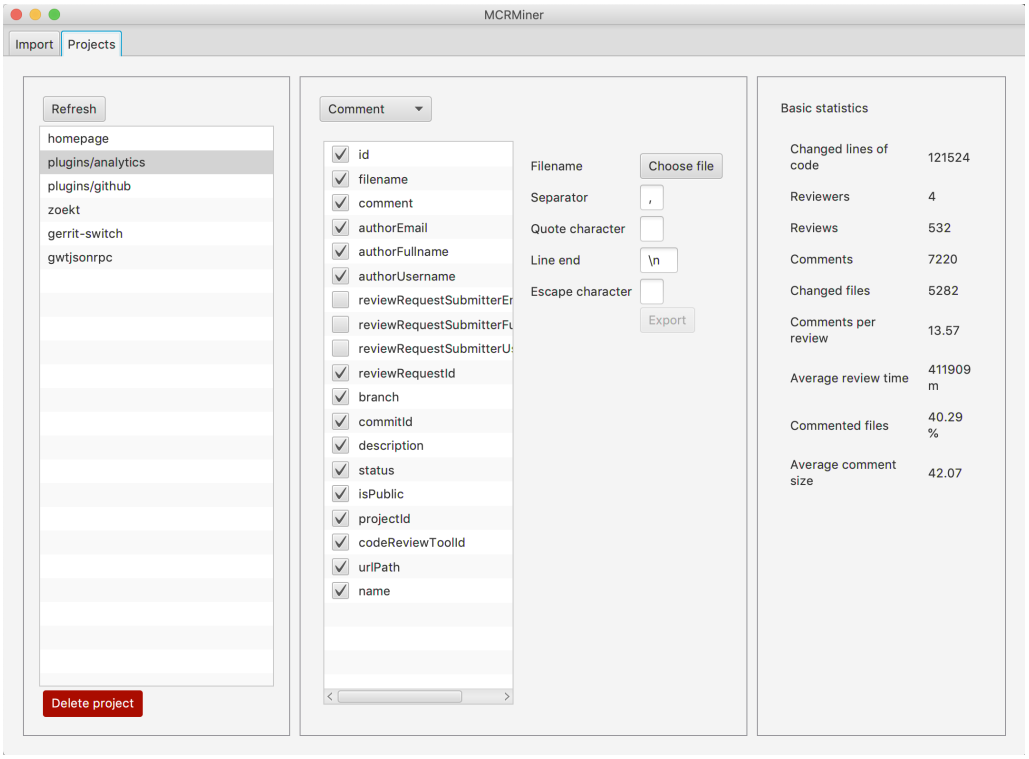

Figure 4. Repository Export and Project Statistics.

4.2.1. Project Statistics

Lastly, the statistics feature, shown on the right-hand side of Figure 4, allows the extraction of basic statistics of a given project to provide additional data about them. This information can be used by a researcher to, e.g., decide to use a project in an empirical study. The provided statistics are: average comments per review, mean review time in minutes, modified lines of code, number of reviewers, number of comments, number of reviews, percentage of commented files, average comment size in characters and number of modified files.

5. Evaluation

To validate MCRMiner, in particular its instance to import data from Gerrit, we imported a set of repositories. In total, we selected seven repositories, considering different amounts of data, as well as different values of the parameters measured, such as the number of revisions and files. The data was extracted from repositories using a notebook with an Intel© Core™ i5-7200U processor, with 4 cores and 2.50 GHz, 4 GB RAM memory, and 10 Mbps of internet connection. All repositories that had its data extracted are hosted in the repository located at the Gerrit's official address[12]. The imported projects, as well as the time taken to fetch data and calculate statistics of the projects, can be seen in Table 1. Note that some of the projects have no comments (which are associated with files). In these projects, the feedback was always given as reviews (with a description and a status).

It can be observed, based on the results presented in Table 1, that the execution

[12] `https://gerrit-review.googlesource.com`

Table 1. Imported MCR Repositories.

Project	Execution Time	Reviewers	Reviews	Comments	Changed LOC	Changed Files
bazlets	1h 10m 9s	5	1490	0	22052	2533
gerrit-switch	15m 17s	6	152	456	1311	247
gitfs	3h 28m 9s	2	1116	2604	235724	8060
gwtjsonrpc	50m 48s	3	1728	0	3888	1296
plugins/analytics	3h 55m 17s	4	532	7220	121524	5282
plugins/github	1h 32m 45s	3	2592	0	15552	3456
zoekt	8h 13m 55s	2	2982	5467	1083460	25844

time is directly proportional to the number of revisions, comments and changed files, and this is due to the fact that each of these corresponds to entities that must be downloaded from the Gerrit repository, mapped to our domain model and stored in the local database. Due to this, the number of changed lines of code or the number of reviewers does not impact on the time results. The number of reviewers could impact the execution time if it was directly proportional to the number of comments, but it was not always the case. During the MCRMiner execution, we observed that the most significant bottleneck in the performance of the import feature is the latency time of the network. However, the time of reading and writing operations in the database and the speed of the processor also influenced the results.

6. Conclusion

Modern Code Review (MCR) had its popularity increased over the past years due to its benefits of improving product and code quality as well as promoting knowledge sharing and team cooperation. To improve this practice, much research has been carried out including empirical studies that analyze data stored in code review systems and derive lessons learned to improve MCR. Although such data is available by means of APIs, a recurrent step in these studies is to write code to extract data and store it in a format to be processed—an effort that can be reduced.

In this paper, we presented MCRMiner, a tool that extracts data from code reviews systems, currently supporting only Gerrit, and exports them to files that can be loaded in a single command in many programming languages and statistical tools. MCRMiner is implemented as a framework on top of up-to-date technologies, following a typical layered architecture. Its domain model was built taking into account two known code review systems so that it can be used to store information from systems other than Gerrit. MCRMiner includes a set of hotspots that can be extended to customize and adapt the framework. Its import feature requires only to indicate a project repository to be imported. Imported projects can have basic statistics visualized, such as the number of reviews and reviewers, and its data exported in CSV or other similar file formats, with data displayed according to a selected from five different perspectives. MCRMiner is an open-source project, which has additional information and source code available on the web.[13]

[13] `http://www.inf.ufrgs.br/prosoft/tools/mcrminer`

MCRMiner serves as a tool to support the starting point of many empirical studies on MCR. However, it can be further extended to include other features, leading to future work. First, it can be extended to import data from other code review systems. Second, it can include additional data processing before exporting data, such as filtering a subset of data that is of interest. Finally, it can be integrated to other tools, such as ReDA, to provide visualizations.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors would like to thank CNPq for grants ref. 313357/2018-8 and 428157/2018-1.

References

Araújo, C. W., Nunes, I., and Nunes, D. (2017). On the effectiveness of bug predictors with procedural systems: A quantitative study. In *Proceedings of the 20th International Conference on Fundamental Approaches to Software Engineering - Volume 10202*, pages 78–95, New York, NY, USA. Springer-Verlag New York, Inc.

Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712–721.

Fagan, M. E. (1986). Advances in software inspections. *IEEE Transactions on Software Engineering*, SE-12(7):744–751.

Ferreira, I. P. (2018). MCRMiner: um framework de mineração de repositórios de code review. Technical report, Porto Alegre, BR.

Johnson, P. M. (1998). Reengineering inspection. *Commun. ACM*, 41(2):49–52.

Rodríguez-Pérez, G., Gonzalez-Barahona, J. M., Robles, G., Dalipaj, D., and Sekitoleko, N. (2016). Bugtracking: A tool to assist in the identification of bug reports. In Crowston, K., Hammouda, I., Lundell, B., Robles, G., Gamalielsson, J., and Lindman, J., editors, *Open Source Systems: Integrating Communities*, pages 192–198, Cham. Springer International Publishing.

Santos, E. W. d. and Nunes, I. (2017). Investigating the effectiveness of peer code review in distributed software development. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES'17, pages 84–93, New York, NY, USA. ACM.

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., and Matsumoto, K. (2015). Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 141–150.

Thongtanunam, P., Yang, X., Yoshida, N., Kula, R. G., Cruz, A. E. C., Fujiwara, K., and Iida, H. (2014). Reda: A web-based visualization tool for analyzing modern code review dataset. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 605–608.

Yang, X., Kula, R. G., Yoshida, N., and Iida, H. (2016). Mining the modern code review repositories: A dataset of people, process and product. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 460–463.