MetricMiner: Supporting Researchers in Mining Software Repositories

Francisco Zigmund Sokol, Mauricio Finavaro Aniche, Marco Aurélio Gerosa
Department of Computer Science
University of Sao Paulo (USP) - Brazil
E-mail: francisco.sokol@usp.br, {aniche, gerosa}@ime.usp.br

***Abstract*—Researchers use mining software repository (MSR) techniques for studying software engineering empirically, by means of analysis of artifacts, such as source code, version control systems metadata, etc. However, to conduct a study using these techniques, researchers usually spend time collecting data and developing a complex infrastructure, which demands disk space and processing time. In this paper, we present MetricMiner, a web application aimed to support researchers in some steps of mining software repositories, such as metrics calculation, data extraction, and statistical inference. The tool also contains data ready to be analyzed, saving time and computational resources.**

***Index Terms*—mining software repositories; supporting tool; code metrics**

I. Introduction

Techniques of mining software repositories enables researchers to study software engineering practices empirically. Practitioners by means of these techniques uncover useful information for the software development team, such as frequently changed or error-prone classes, or the identification of core developers in order to transfer knowledge. With this information exposed, teams can take actions to improve their code and processes.

To develop a study in the area, researchers need to gather large amounts of data sometimes from many different projects and store them in their own workstations or servers. Then, manually run code metrics, and perform statistical calculations. This process requires the installation of several tools and libraries, making the process complex and slow. Besides the complexity, this kind of research consumes many computational resources. To start with, the repositories download uses a reasonable amount of bandwidth. After being processed and persisted in a database, the data occupies a huge disk volume. To calculate metrics on a lot of artifacts a large amount of processing time is required. Finally, after all these steps, it is possible to extract information, and evaluate them by means of statistical analysis. It means that researchers spend a lot of time working on the tools, rather than in analyzing the data and interpreting the results.

Based on all these difficulties, we decided to develop MetricMiner, a web application that performs all these steps without requiring great effort from the researcher. With it, researchers can write new metrics, and extract information from a reasonable quantity of different projects. In this paper, we present the tool, its functionalities, and architecture decisions. We also present a replication study that was developed using the tool.

II. MetricMiner: A Web Application to Support Research in MSR

Understanding the process of software evolution is a hard task. Large systems tend to have a long development history, with many different developers working on different parts of the system. It is common that no developers know all source code of the project. Because of that, the idea of a manual analysis of all software is impracticable.

Mining Software Repositories (MSR) analyses the software evolution in an automated way, through the application of data mining techniques into the development history data. Studies in this field reveal useful information to the development of a particular project or even find patterns in software evolution that can be generalized to other software systems.

The term *"software repository"* comprises all artifacts created during the development of a software system. From source code files that are stored in a source control manager, such as Git or SVN, to messages that developers exchange in mailing lists. Such repositories contain useful information, which can be explored to comprehend the software evolution and contribute to its development.

MetricMiner is a web application that aims to support researchers when working with mining software repositories. As mentioned before, when a researcher needs to do a study like that, s/he needs to install many different tools, libraries, etc., and spend many computational resources. As a web application, MetricMiner makes use of the power of cloud computing to scale. This way, researchers do not have to worry about resources. Currently, MetricMiner is running over a cloud infrastructure and it is currently available at http://metricminer.org.br/.

MetricMiner was based on rEvolution[1], a command-line tool that extracts data from a local repository and persists them in a database. rEvolution was limited to collect data from just one project, requiring researchers to execute it manually for each repository. In addition, the configuration of the tool was complex. It was necessary to configure database, source control tools, and all external applications that the tool uses in a single XML file.

[1]http://github.com/mauricioaniche/rEvolution.

	Quantity
Total of projects	317
Total of committers	5,274
Total of commits	887,493
Total of artifacts	1,453,123
Minutes of work spent with tasks	39,203

TABLE I
CURRENT NUMBERS OF PROJECTS IN METRICMINER

The tool automatically clones source code repositories, processes all repository metadata, stores it in a database, allows researchers to create queries and to manipulate data, and even to run statistical tests on the data set.

In the subsections below, we better explain the functionalities as well as the architectural decisions. All the source code and development history is hosted on Github, under an open source license[2].

A. Current Features

When researchers decide to do a study in MSR, they first choose projects. When using MetricMiner, the researcher only needs to insert the name of the project and the URL to its repository into the web application. Currently, MetricMiner supports Git and SVN repositories. As soon as the researcher inserts the project, the tool automatically starts to clone the repository, and to run all the existent metrics for each version of the project. In addition, researchers can access repositories that were previously inserted by other researchers. As it is all done in the cloud, MetricMiner currently contains a reasonable number of projects that were already processed. For example, all Apache open source projects are already there. In Table I, we show the current numbers of projects and commits in MetricMiner.

As all steps in the process usually take time, all tasks are executed asynchronously. MetricMiner has an internal queue, in which tasks are executed. Researchers can monitor the web application to find out when their tasks finished.

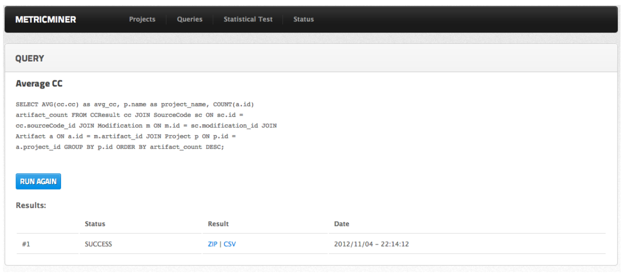
At the end of the cloning process, all relevant information, such as source code, list of committers, date/time information, code metrics, are persisted in the database. MetricMiner already contains several metric implementations: cyclomatic complexity [6], lack of cohesion of methods (LCOM) [3], efferent coupling [5], amount of lines of code[1], and amount of method invocations[4].

Researchers are able to query all the data that is in the tool by just typing a SQL query. MetricMiner execute the query asynchronously, store the result, and send the researcher an email notification. Then, the researcher can download the results in a standard CSV format[3], and refer to the results in their papers by an unique URL that MetricMiner provides.

As new projects can be added at any time, queries can be re-executed at any moment. However, no data is lost; older results are saved and their unique URL persists, even after the query is executed again. In Figure 1, we show the screen in which developers can create SQL queries.

[2]http://github.com/metricminer-msr/metricminer. Last access on June, the 9th, 2013.

[3]To preserve the identity of the developers, the tool replaces the developers' name and emails by a hash string.



Average CC

```
SELECT AVG(cc.cc) as avg_cc, p.name as project_name, COUNT(a.id)
artifact_count FROM CCResult cc JOIN SourceCode sc ON sc.id =
cc.sourceCode_id JOIN Modification m ON m.id = sc.modification_id JOIN
Artifact a ON a.id = m.artifact_id JOIN Project p ON p.id =
a.project_id GROUP BY p.id ORDER BY artifact_count DESC;
```

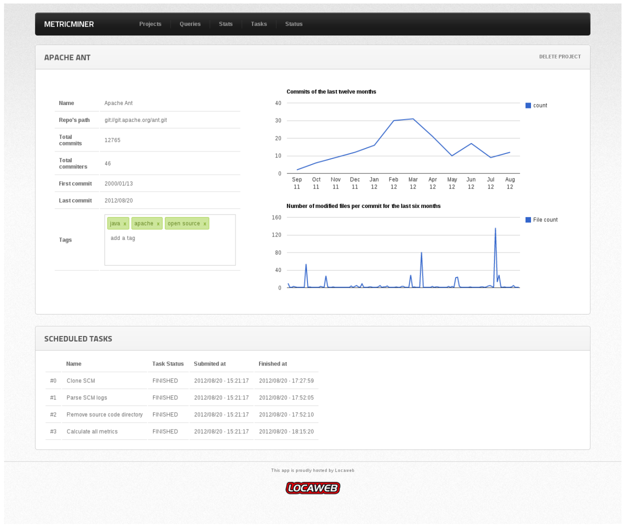
Results:

	Status	Result	Date
#1	SUCCESS	ZIP | CSV	2012/11/04 – 22:14:12

Fig. 1. A query in MetricMiner

Besides downloading it as a CSV, researchers can also use the result of a query as an input to a statistical test. To do so, they choose two datasets and the statistical test (for example, T Student, Wilcoxon, and so on). MetricMiner then dynamically writes and executes an R script [4], and stores the result in an unique URL.

Researchers can also navigate into each project, as they have their own page. In Figure 2, we show the homepage for the Ant project, which contains more than 12,000 commits. The tool presents basic information, such as the total number of commits, committers, first and last commit's date/time. The tool also shows a few charts, showing the number of commits in the last twelve months, and the number of files changed in each commit for the last six months. Researchers can also add tags to the project to group similar projects in a future data extraction.



Name	Apache Ant
Repo's path	git://git.apache.org/ant.git
Total commits	12765
Total commiters	46
First commit	2000/01/13
Last commit	2012/08/20
Tags	java, apache, open source

	Name	Task Status	Submited at	Finished at
#0	Clone SCM	FINISHED	2012/08/20 - 15:21:17	2012/08/20 - 17:27:59
#1	Parse SCM logs	FINISHED	2012/08/20 - 15:21:17	2012/08/20 - 17:52:05
#2	Remove source code directory	FINISHED	2012/08/20 - 15:21:17	2012/08/20 - 17:52:10
#3	Calculate all metrics	FINISHED	2012/08/20 - 15:21:17	2012/08/20 - 18:15:20

Fig. 2. Ant homepage, available on http://metricminer.org.br/project/12

B. Architecture and Design Decisions

In Figure 3, we show the basic flow in MetricMiner. MetricMiner was developed in Java, and can be deployed in any Java web container. Currently, the application is running in *Apache Tomcat*. The tool also uses MySQL to store the data.

[4]The R Project for Statistical Computing. http://www.r-project.org/.

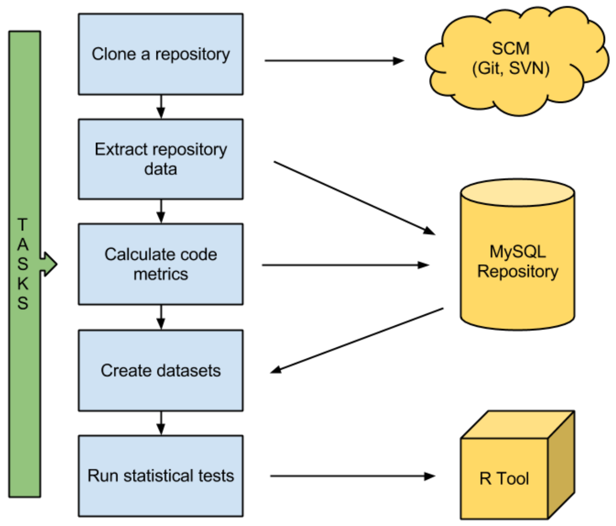


Fig. 3. Basic flow of working in MetricMiner

The domain model contains all information from the source control repository. It stores commits, authors, dates, and source codes. Each commit is related to a set of modifications. These modifications can be classified as a "change," "new file," or "deleted." For each file, MetricMiner stores the current source code and the difference (*diff*) to the last commit. The tool does not store binary artifacts, such as images, zip files, etc. By storing all the information, we enable researchers to create new metrics, and run over old repositories, without the need of restarting the whole process.

Internally, the tool uses a queue to organize the tasks it needs to execute. Tasks represent the steps that MetricMiner takes to extract information from a project: repository cloning, extract repository metadata, run code metrics, and run a statistical test. The task system is extensible. In Figure 4, we show an UML diagram that represents the tasks. If one needs to create a new task, s/he basically needs to create a concrete implementation of the interface.

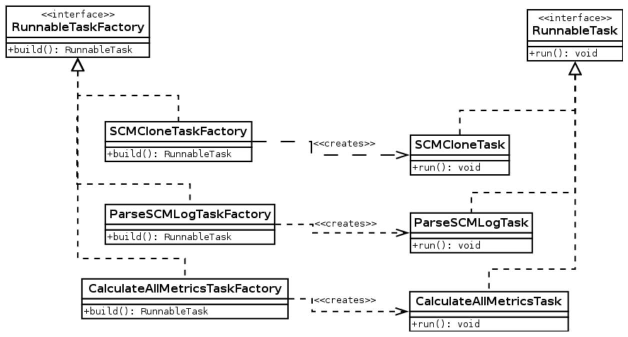


Fig. 4. UML that represents a Task in MetricMiner

Analogously, new code metrics can be inserted into MetricMiner. Researchers only need to implement a set of interfaces, and the metric will be ready to be executed in all source code. The interface is simple. There are three methods that need to be implemented: one that calculates the metric based on the source code, one that returns one (or many) results based on the calculation, and one that returns if that metric should be executed for that file (Java metrics should run only in *.java files, for example).

Not all code metrics require compiled code: they use statically analysis. The process of compiling a project can be costly and tricky, as each project contains its own way to be built. The existent metrics of MetricMiner use the *javaparser*[5] library. With it, the abstract syntactic tree is built and each metric is implemented as a visitor of this tree.

Many examples of it can be seen in the MetricMiner source code. This link[6] points to the implementation of the McCabe metric, using the mentioned visitor.

It is important to highlight the fact that, as soon as a new metric is implemented, MetricMiner automatically detects it, and schedules the execution of it to all repositories that exist in the tool.

The tool also abstracts the persistence problem: researchers are able to create the domain object the way they want, and MetricMiner, by using Hibernate, persists it, without the need of writing any SQL Insert statement. It means that researchers should only worry about creating an object-oriented model of their metrics. In Figure 5, we show the UML diagram that represents the internals of two code metrics in MetricMiner.

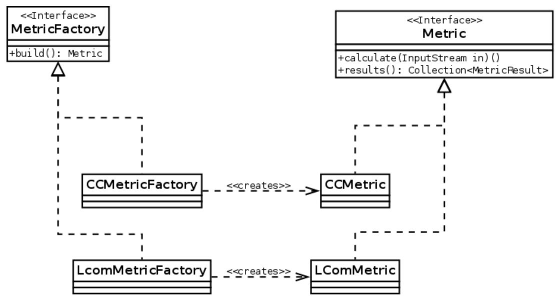


Fig. 5. UML that represents a Metric in MetricMiner

Performance is also something important for researchers. During the development, we tried different strategies to make the execution of all tasks faster. At the end, the current implementation makes use of first and second level caching, stateless Hibernate sessions, and data pagination. In numbers, we improved the average time of task execution from 5.3 minutes to 3 minutes.

C. *Evaluation of the Tool*

We evaluated the capability of the tool by mining all source code repositories available in the Apache Software Foundation. All repositories were accessed via Git. The complete list of projects can be found at http://git.apache.org/. There were 307 different projects in the website. At the end of the process, MetricMiner was able to process and store more than 800k commits from more than 2k different committers. There were more than 1.5 million different artifacts and 5 million different versions of source code. All these data are stored in a database

[5]https://code.google.com/p/javaparser/. Last access on June, the 5th, 2013.
[6]https://github.com/metricminer-msr/metricminer/blob/master/src/main/java/org/metricminer/tasks/metric/cc/CCVisitor.java.

	Decreased CC	Did not change CC	Incremented CC
Documented refactorings	14 / 1,504	7 / 1,603	12 / 3,230
No refactoring documented	27 / 30,145	580 / 99,580	136 / 121,239

TABLE II
FINDINGS IN SOETENS AND DEMEYER STUDY / OUR REPLICATION

with more than 180 gigabytes and available to researchers by means of the web application. All the process, from the beginning of the first project up to the metrics of the last one, took 90 hours.

In addition, to exemplify the value of the tool to researchers in the field, we reproduced a published study. The work by Soetens and Demeyer [8] was replicated and extended using MetricMiner. In this paper, the authors studied the effects of refactoring over the complexity of the system. They analyzed 776 versions that were extracted from PMD[7]. To do the analysis, the authors made use of two Eclipse plugins: the *SVNKit*, to extract data from the SVN repository, and the *Eclipse Metrics*, to calculate the cyclomatic complexity of the code. With these two plugins, they developed their own plugin to do all the work of loading the version of code, calculate the metrics, and save them into an XML file. After that, the XML files were processed and the associated metrics were related to a commit message.

To reproduce this work, we have done a SQL query to the MetricMiner database. The SQL query is represented in the box below. This query extracts the cyclomatic complexity of all classes that were updated in all projects. Together with the cyclomatic complexity, we also extracted the name of the project, the date of the modification, and the commit message. The resulting dataset can be found online [8]. With the dataset in hands, we developed a small Java program that sums up the effect of the refactoring on the cyclomatic complexity. The source code of this program can be found online [9]. In Table II, we show the results found in their study versus our replication. One can notice that the amount of data analyzed was much higher than in the original paper.

The extension of the original paper was facilitated by having all the data already available and common metrics, like cyclomatic complexity, already calculated for all source code artifacts. In addition, it is important to notice that we executed our study in more than 300 projects, while the original authors did it in only one project. The use of MetricMiner enabled us to run the study over many projects at once.

III. RELATED WORK

A similar tool to MetricMiner is Sonar[10], a web application that analyzes the source code and extracts a huge variety of code metrics and structural dependencies among classes. The focus of this tool is to support the development team to keep track of the code quality. Sonar does not store metadata from the code control system and does not provide a way to extract its data. However, the number of different data visualizations is noticeable. Because of that, Sonar is frequently used in industry, but not for academic purposes.

Eclipse Metrics[11] is an Eclipse plugin that calculates code metrics. Developers can keep track of their code evolution while they are programming. However, the plugin does not execute metrics on the whole repository, but only in the current codebase. The tool requires compiled code, which may be a problem depending on the repository being analyzed.

Kalibro[12] and Analizo[13] are tools that calculate different code metrics. While Analizo calculates metrics in many different languages, Kalibro focuses on giving support to developers, giving them reference values to the metrics and pointing out potential problems. The tool is very flexible: developers can configure the reference values, and compose new metrics using JavaScript. However, Kalibro does not analyze the whole repository history. The user needs to select versions and recalculate the metrics manually.

Mezuro[14] is a web application built over Kalibro and Analizo. Through its web interface, the user adds software projects and the metrics are calculated over the code (using the Analizo tool). Reference values are presented to the users, suggesting good and bad points in the source code.

EvolTrack is a software evolution visualization tool. Developed as an Eclipse plugin, EvolTrack processes the history in a source code control, and enables users to see the evolution of the classes over time. The tool shows the class diagram of the project and allows the user to go forward or backwards, seeing the classes that were added or removed. Currently, the tool supports only SVN repositories. EvolTrack also has plugins for different data visualizations, such as EvolTrack-SocialNetwork [9], which shows the relationship among the developers during the evolution of the software.

ArchView [7] is a visualization tool to analyze the software evolution. The tool extracts information from the source control (CVS) and from the bug tracking (BugZilla). After processing the information, ArchView enables users to visualize different metrics from an artifact in any version of the software, using Kiviat diagrams.

CodeCity [10] makes use of the metaphor "software as a city," and generates a city based on the source code. Each class is a building. Classes are grouped in neighborhoods. The height and weight of the building depends on the number of methods and attributes of the class.

Boa [2] is a domain-specific programming language for analyzing ultra-large-scale software repositories. Boa makes use of distributed computing techniques to execute queries against software repositories in an efficient way. As soon as the researcher learns the DSL, they can extract interesting information in a simple way. By February of 2013, Boa had almost 700,000 projects in its repository.

[7]http://pmd.sourceforge.net/
[8]http://metricminer.org.br/query/1
[9]https://github.com/csokol/refactoring-cc
[10]http://www.sonarsource.org/
[11]http://metrics.sourceforge.net/
[12]http://www.kalibro.org/
[13]http://www.analizo.org/
[14]http://mezuro.org/

	MetricMiner	Sonar	Eclipse Metrics	Kalibro/Analizo	Mezuro	Evoltrack	ArchView	CodeCity	Boa
Web application	X	X	-	-	X	-	-	-	X
Interface to query data	X	-	-	-	-	-	-	-	X
Code metrics	X	X	X	X	X	-	X	-	-
Code metrics in non-compiled source-code	X	-	-	X	X	-	-	-	X
Graphic interface to visualize data	-	-	-	-	-	X	X	X	-
Statistical tests	X	-	-	-	-	-	-	-	-
Git repositories	X	-	-	-	X	-	-	-	-
SVN repositories	X	-	-	-	X	X	-	-	-
CVS repositories	-	-	-	-	X	-	X	-	-

TABLE III
COMPARISON BETWEEN METRICMINER AND OTHER TOOLS

In Table III, we compare the related tools. The main difference of MetricMiner is that it stores the calculated value of the metrics so that the queries may be executed very fast and researchers may adopt an exploratory approach in the large amount of data, rapidly prototyping their study, without needing to install anything in their workstations.

IV. CONCLUSION AND FUTURE WORK

In this paper, we described MetricMiner, the web application that facilitates the work of researchers involved in mining software repositories. The tool aids researchers in all steps during a MSR study, such as cloning the repository, extracting data, creating specific datasets, and running statistical tests. Using the tool, we were able to extract a huge quantity of data from many repositories. We were also able to replicate a study of the literature and extend it to more than 300 projects. The tool's extension points support the creation of specific metrics or tasks, enabling researchers to deal with the data in a personalized fashion.

More improvements will be done in MetricMiner in the future. Parallelization of the tasks execution, for instance, may improve more the performance of the application. In terms of metrics, it is possible to implement more metrics. Collecting data from other repositories, such as bug tracking and mailing lists would provide researchers the possibility of triangulating their findings. We also plan to gather more studies from the MSR literature and replicate them in MetricMiner as a way of evaluating and extending the tool support. These replication may be useful to identify data analysis patterns which may be made available in MetricMiner.

ACKNOWLEDGMENT

We thank Locaweb for sponsoring the project, providing us their cloud infrastructure. Marco Gerosa receives individual grant from CNPq.

REFERENCES

[1] C. Chidamber, S.; Kemerer. A metrics suite for object oriented design. pages 476–493. IEEE TSE, Vol. 20 (6), 1994.

[2] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. International Conference on Software Engineering (ICSE2013), 2013.

[3] B. Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, 1996.

[4] W. Li; S. Henry. Object-oriented metrics that predict maintainability. J. Systems and Software, vol. 23, no. 2, 1994.

[5] J. Lorenz, M.; Kidd. *Object-oriented metrics: A Practical Guide*. Prentice-Hall, 1994.

[6] T. McCabe. A complexity measure. pages 308–320. IEEE TSE, SE-2, Vol. 4, 1976.

[7] Martin Pinzger. *ArchView - Analyzing Evolutionary Aspects of Complex Software Systems*. PhD thesis, 2005.

[8] Quinten David Soetens and Serge Demeyer. Studying the effect of refactorings: a complexity metrics perspective. In *QUATIC 2010: The 7th International Conference on Quality in Information and Communications Technology*. IEEE Computer Society Press, IEEE Computer Society Press, 2010.

[9] C. M. Vahia, A. M. Magdaleno, and C. M. L Werner. Evoltrack-socialnetwork: Uma ferramenta de apoio à visualização de redes sociais. In *Congresso Brasileiro de Software (CBSoft) – Sessão de Ferramentas*, 2011.

[10] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In Jonathan I. Maletic, Alexandru Telea, and Andrian Marcus, editors, *VISSOFT*, pages 92–99. IEEE Computer Society, 2007.