SEAgle: Effortless Software Evolution Analysis

Theodore Chaikalis, Elvis Ligu, George Melas, Alexander Chatzigeorgiou
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
Email: {chaikalis, ligu, melas, achat}@uom.edu.gr

***Abstract*—The analysis of software evolution by means of mining public repositories has been established as one of the dominant approaches for empirical studies in software engineering. However, even the investigation of the simplest research question demands a mazy process involving installation and configuration of tools, climbing their learning curve and tedious collection of desired information. Acknowledging the need for effortless querying of remote repositories we introduce a Web-based 'one-click approach' to perform software evolution analysis of Git projects.**

Keywords—software evolution analysis; software engineering platforms; mining software repositories;

I. INTRODUCTION

Empirical studies in software engineering have been leveraged by the availability of public repositories hosting open source software projects. Nowadays, Distributed Version Control Systems (DVCS) are the preferred choice for collaborative software development with Git being the dominant version control software [2], [5]. At the time of writing, the GitHub hosting service for Git projects hosted over 13.8 million repositories [9].

To serve the needs of the software engineering research community various platforms have been proposed to facilitate the collection, analysis and reporting of data retrieved from public software repositories. However, based on our experience the existing approaches suffer from various limitations and problems such as difficulty of installation, configuration and use, superficial information or inability to choose specific projects.

Acknowledging the need for easy access and inquiring of software repositories we propose a Web-based, "one-click" approach for mining source code information, named *SEAgle*.

II. RELATED WORK AND EXISTING TOOLS

The increasing interest on software repository mining led to the creation of several tools, frameworks and techniques that facilitate the overall process. Most of the existing approaches has been recorded by Chaturvedi et al. [3] who reviewed all papers published in conferences related to repository mining since 2007. In more than half of the papers the proposed approach is backed up by a tool developed for this purpose.

One of the most prominent tools is SonarQube [10], which is an online platform that evaluates software quality and through a reporting mechanism it provides an overview of the project state as well as the estimated technical debt. This type of continuous analysis can certainly be applied on projects retrieved from public repositories. However, it has not been designed with the software engineering researcher in mind, as each project has to be downloaded individually.

More targeted to software engineering research is Ohloh [11], a public directory of open source projects that provides basic information about the size and developer contribution among others. A notable tool for Automated Software Engineering called Kenyon has been developed by Bevan et al. [1]. Its main feature is the ability to facilitate the creation of new evolution analysis tools as well as the data sharing among them. Gousios and Spinellis [6] introduced the "Alitheia Core" platform that automates metric collection from online repositories and provides a programing interface for querying the available results. A domain-specific language and infrastructure to test hypotheses related to Mining Software Repositories (MSR) is Boa [4], which enables querying through a web-based interface. Deep IntelliSense [7] is a Visual Studio plugin that can provide information related to dependencies among software modules in order to help developers better understand the way that each artifact has evolved. Linstead et al. [8] proposed "Sourcerer", an infrastructure that automatically parses and analyzes online software repositories in order to provide information about the program functions and source code similarities as well as developer activities and similarities among developer programming styles.

However, notwithstanding this abundance of tools and platforms, the systematic presentation of the evolution of size properties, software metrics and repository activity together in a single dashboard and without the need for human intervention, is still not available. Therefore, software engineers are still compelled to collect data from many different sources, transform the data in a common format and finally combine information from each field to perform meaningful queries.

III. CONCEPTS BEHIND THE PROPOSED PLATFORM

The development of the proposed platform was driven by the following key issues and decisions:

- the platform should be easy to use. To this end we opted for a Web based platform enabling users to analyze a repository by a single click (either selection of an already analyzed project or by providing the git repository URI).

- software repositories encompass a project's history. As a result, all reported information spans across all available versions, i.e. constitutes a form of software evolution analysis.
- Any software system has several facets. Therefore, we offer multiple views concerning commit-related metrics, source code metrics and graph based metrics.
- Empirical studies very often focus on the investigation of relations among variables. To satisfy this need we offer direct correlation analysis between any two monitored variables. (for this reason the *x*-axis is common on all diagrams and represents software versions)
- Contemporary software repositories are extremely large in size. To confront this challenge, we optimized the process of extracting commit-related metrics, which are demanding since they involved the analysis of thousands of commits.

IV. Architecture and Employed Technologies

The architecture of SEAgle is outlined in Fig. 1. In the left hand side components offering core services are shown, such as the API taking care of communication with VCS and the API responsible for metrics. For the latter two components the architecture is highly extendible in the sense that a clear separation between abstraction and implementation has been adopted. The Software Evolution Analysis Engine, running in Java EE, exploits services provided by individual components and stores the calculated results in a MySql database. Moreover, the engine provides Web Services (SOAP/REST), which are accessed by the presentation tier in order to trigger the analyses and retrieve the results which are then displayed in the form of charts and tables.

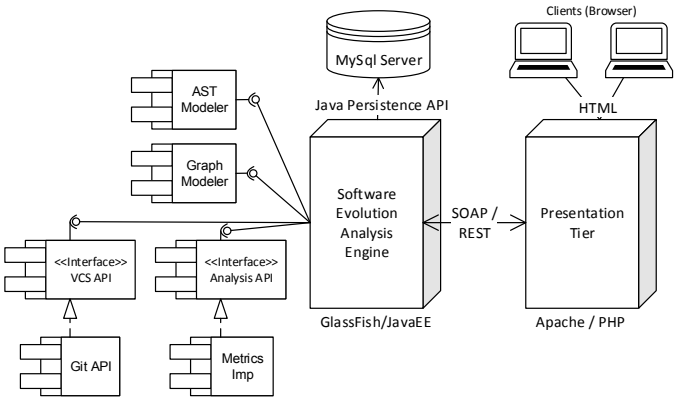

Fig. 1. Architecture of the SEAgle platform.

The proposed platform employs a plethora of state-of-the-art technologies, which are outlined in Table I.

It should be mentioned that in order to minimize human intervention versions are automatically determined based on tags explicitly contained within the git repository. This is in alignment with common practices in software development where tags delineate different software releases.

TABLE I. Employed Technologies

Core Components	
JGit	• To access git source code management (SCM) systems
Guava	• Extended Java Collections • Caching
Software Evolution Analysis Engine	
Java EE 7 / Glassfish	• provides an API and runtime environment to run on a Web Server
Web sockets	• interaction with presentation tier to provide progress monitoring
JAX/WS	• provides an access point to the analysis engine
Java Persistence API (JPA)	• facilitates object-relational mapping and storage of analysis results to the database
R	• Calculation of Statistical Measures
Presentation Tier	
PHP	• SOAP Client implementation
HTML5, CSS3	• Local storage, Adaptive screen controls, etc.
Bootstrap	• Responsive design
JavaScript + Flot, JQuery, Sparkline libraries	• Chart creation technologies • Data manipulation

V. Usage Scenario

A. Selection of Project to be analyzed

The homepage of SEAgle, shown in Fig. 2, awaits a single user input, which may be either:

- a project name
- a git URI

Fig. 2. Main search page.

In case a project name is entered, it will be looked for in the already analyzed projects for which results are available. If the user types in a git URI, it is also being checked whether the corresponding repository has been analyzed. If not, the user request triggers the analysis.

Since the analysis of large repositories can be time and resource consuming, the user is notified on the progress of processing. Moreover, the system can notify the user by email

when the analysis has been completed. The home page offers (on the right hand side) a timeline overview of the recently analyzed projects (shown in Fig. 3).

Fig. 3. Timeline of recently analyzed projects.

B. *Dashboard and Results*

For each analyzed project a dashboard containing a metrics overview is displayed (Fig. 4).

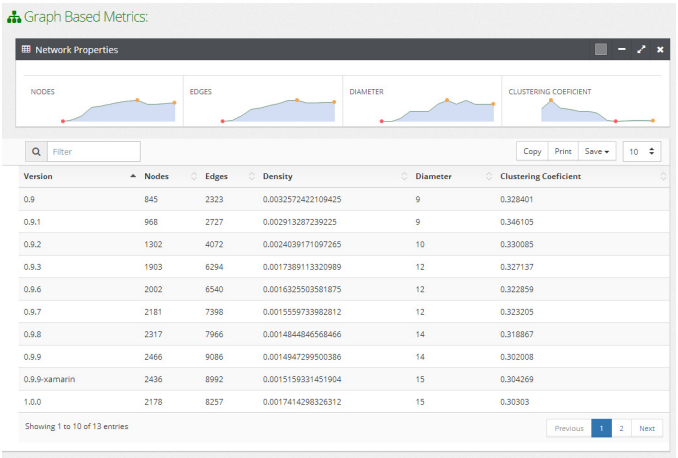

Fig. 4. Overview of project metrics.

Detailed information concering the evolution of metrics for the three examined views (commit, source code and network metrics) can be displayed by selecting "Evolution Analysis" on the left menu. As an example, in Fig. 5 the evolution of commit-related metrics over the examined versions of a project are shown. The results are also shown as Tables with columns corresponding to metrics, and rows to examined versions.

By clicking the "Save" button on every tabular representation, the corresponding data can be exported in CSV, Excel or PDF format to allow further experimentation. The available metrics of SEAgle are summarized in Table II.

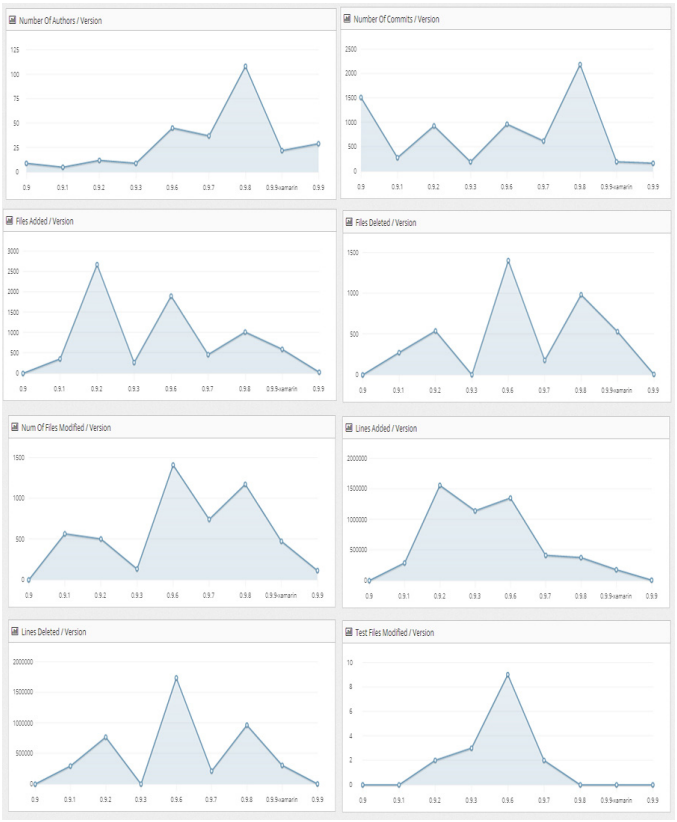

Fig. 5. Diagrams that depict metrics related to repository activity.

TABLE II. SEAGLE METRICS

Commit-Related Metrics	
Authors	Added Lines
Commits	Deleted Lines
Added Files	Added Test Files
Deleted Files	Modified Test Files
Modified Files	
Source-Code Metrics	
Coupling Between Objects (CBO)	Lack of Cohesion of Methods (LCOM)
Number of Attributes (NOA)	Number of Methods (NOM)
Weighted Method Complexity (WMC)	
Graph-Based Metrics	
Number of nodes	Number of edges to new nodes
Number of edges	Number of edges between existing nodes
Diameter	Number of edges between new nodes
Density	Number of edges to existing nodes
Clustering Coefficient	Number of deleted edges

C. *Correlation Analysis*

This feature allows the user to select any two monitored variables (from the entry "Correlation Analysis" in the left-hand menu) and investigate the way that the corresponding measures co-evolve over time. Both trends are shown on a common chart (employing a secondary y-axis) for improved readability (Fig. 6). Moreover, the Pearson correlation coefficient as well as the corresponding significance level (p-value) are shown. Currently, the platform is capable of calculating the correlation of 153 different metric pairs.

Fig. 6. Correlation analysis between two selected metrics (here: Number of Nodes and WMC).

VI. Experiments

To investigate the performance of the proposed platform we have tested a number of projects available in Git repositories. The experiment was carried out in a Windows 8 PC with Intel Core i7-3770K processor running at 3.5 Ghz, 8 GB DDR 3 RAM, 64 GB Solid State Drive on eSata port, Java 1.7.0_40 64bit, Java EE 7, Glassfish 4 and MySql Server Community Edition 5.6. Information of the project name, size, number of Java files in the first and last version, the number of analyzed versions as well as the repository size is shown in Table III.

TABLE III. Analyzed Projects

	LibGdx	Hystrix	Mongo Java Driver	RxJava	GitHub android app
Size (MB)	122 - 158	1.0 - 2.6	1.2 – 5.4	0.5 - 4	1.4 – 2.3
Java files	685 - 1807	74 - 187	107 - 360	55 - 494	216 - 279
Versions	13	53	70	72	19
Repo Size (MB)	812	4	26	13.8	6.9

The time (in secs) required for each step of the proposed process is shown in Table IV. As it can be observed cloning the remote repository, employing the JGit API is extremely fast, even for large repositories. Execution time for the rest of the steps is strongly dependent upon the number of versions and unavoidably of the number of Java files.

TABLE IV. Execution Time per Step (secs)

	LibGdx	Hystrix	Mongo Java Driver	RxJava	GitHub android app
Cloning of the Git repository	52	5	3	50	7
Reconstruction of the source code/ versions on the file system	324	49	63	250	42
Graph creation and calculation of graph-based metrics	270	28	108	211	15
Calculation of source code metrics	210	110	150	314	12
Calculation of commit-related metrics	197	185	225	373	33
Total Time	1053	377	549	1198	109

VII. Future Work

Since SEAgle is an ongoing project, we aim at extending its features towards all aspects which can make the platform more valuable to software engineering researchers: support of other version control systems, additional metrics, programming languages and statistical analyses. Currently, software evolution is examined at the system level of the analyzed projects. Deeper insight can be obtained by performing the analysis at finer levels of granularity such as the package and class level. Moreover, we aim at providing all extracted metrics through a public REST API to facilitate the collaboration with other tools.

In case SEAgle is embraced by software researchers we consider it valuable to systematize the collection of feedback from its users (e.g. in the form of a discussion forum). Establishing a means of communication with potential users is vital to seek from the software engineering community suggestions for further extensions.

VIII. Website

SEAgle is available at http://se.uom.gr/seagle. The website contains also a screencast describing the main features and the usage of the platform.

Acknowledgment

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thalis – Athens University of Economics and Business - SOFTWARE ENGINEERING RESEARCH PLATFORM.

References

[1] J. Bevan, E. J. Whitehead,Jr., S. Kim, and M. Godfrey, “Facilitating Software Evolution Research with Kenyon,” in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2005, pp. 177–186.

[2] C. Brindescu, M. Codoban, S. Shmarkatiuk, and D. Dig, “How Do Centralized and Distributed Version Control Systems Impact Software Changes?,” in *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, 2014, pp. 322–333.

[3] K. K. Chaturvedi, V. B. Sing, and P. Singh, “Tools in Mining Software Repositories,” in *2013 13th International Conference on Computational Science and Its Applications (ICCSA)*, 2013, pp. 89–98.

[4] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, “Boa: A Language and Infrastructure for Analyzing Ultra-large-scale Software Repositories,” in *Proceedings of the 2013 International Conference on Software Engineering*, Piscataway, NJ, USA, 2013, pp. 422–431.

[5] K. Finley, “Github Has Surpassed Sourceforge and Google Code in Popularity,” *ReadWrite*, 02-Jun-2011. [Online]. Available: http://readwrite.com/2011/06/02/github-has-passed-sourceforge. [Accessed: 30-Jun-2014].

[6] G. Gousios and D. Spinellis, “Alitheia Core: An Extensible Software Quality Monitoring Platform,” in *Proceedings of the 31st International Conference on Software Engineering*, Washington, DC, USA, 2009, pp. 579–582.

[7] R. Holmes and A. Begel, “Deep intellisense: a tool for rehydrating evaporated information,” 2008, p. 23.

[8] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, “Sourcerer: mining and searching internet-scale software repositories,” *Data Min. Knowl. Discov.*, vol. 18, no. 2, pp. 300–336, Apr. 2009.

[9] “GitHub - Features,” *GitHub*. [Online]. Available: https://github.com. [Accessed: 30-Jun-2014].

[10] “SonarQube™ - Open source platform to manage code quality.” [Online]. Available: http://www.sonarqube.org/. [Accessed: 23-Jun-2014].

[11] “Ohloh,” *Ohloh, the open source network*. [Online]. Available: https://www.ohloh.net/. [Accessed: 30-Jun-2014].