LibVCS4j: A Java Library for Repository Mining

Marcel Steinbeck
University of Bremen
marcel@informatik.uni-bremen.de

Abstract

Analyzing the evolution of software systems has become an emerging field of research. In the last couple of years, different studies investigated the evolution of various systems by utilizing version control systems and issue trackers. Due to the absence of suitable libraries, however, most of these studies implemented their own data extraction tools. This paper presents LibVCS4j, a Java programming library that tries to accommodate this shortcoming. The library integrates existing software to access repository routines, adds additional features for data analysis, and defines a common API to make subsequent analyses independent from particular repository systems.

1 Introduction

Due to the complexity of modern systems, the circumstance that software is developed together in teams distributed around the world, and for data backup reasons, version control systems (VCS), such as Git, Mercurial, and Subversion, are likely used to manage middle to large sized software projects. Furthermore, issue trackers are incorporated to put down bugs in writing, emphasize different strategies to solve issues, and, as a result, simplify communication between developers. Using repository mining techniques, different studies have been conducted to analyze the evolution of a multitude of software systems by processing the data provided by version control systems, for instance, the change history of files containing source code, as well as issue trackers, for instance, issues referenced in commit messages [1], [2], [3]. However, due to the absence of self-contained and freely accessible libraries that assist in extracting the required information, tools to gather the necessary data are developed over and over again.

In this paper, we present the Java programming library *LibVCS4j*[1]. The library serves as a tool for processing different version control systems as well as issue trackers under a common API. The project is divided into two submodules: (1) a lightweight and interface-only API that is supposed to be integrated into existing analysis tools, and (2) an implementation providing the features. The remainder of this paper describes the data model and extraction engines of the API submodule.

[1] https://github.com/uni-bremen-agst/libvcs4j

2 Data Model

The entire data model of LibVCS4j is depicted by the UML diagram in figure 1. In the following, only the most essential components are described.

Commit This interface provides basic access to the data of an individual commit applied to a repository. Each instance contains the list of files reported as changed by the underlying VCS engine.

FileChange The *FileChange* interface allows to examine the changes applied to a single file. Accordingly, it provides access to a file's state before and after the corresponding commit has been applied.

Revision A *Revision* represents the analyzed repository at a certain point in time, and is the result of a sequence of commits applied to a predecessor revision.

Size, Complexity LibVCS4j may not only be used to process a repository system, but also adds methods to compute several size and complexity related metrics. We believe that these metrics are useful for analysis tools using LibVCS4j and, thus, integrated them into our data model. In order to compute the metrics for an individual file, the ConQAT scanner library[2] is used to generate its token stream which, in turn, is parsed to collect the necessary data. ConQAT is well suited for this task as it supports several programming languages, amongst others, ADA, C/C++, Java, Python, and Ruby, and is capable of classifying tokens, for instance, comment and literal tokens.

Version The *Version* interface forms the basis of our data model and reflects the file changes of the referenced commits. By referencing more than one commit, the interface provides a mechanism to merge consecutive commits on, for instance, a monthly basis. Accordingly, the actual list of *FileChange* objects is generated based on the commits a version subsumes. For example, if a version v references two commits c_1 and c_2 where c_1 adds a new file f which is removed with c_2 afterwards, the list of file changes reported by v does not include f.

Issue This interface provides basic access to issues extracted from an issue tracker as well as the comments attached to an issue.

[2] https://www.cqse.eu/en/products/conqat/overview

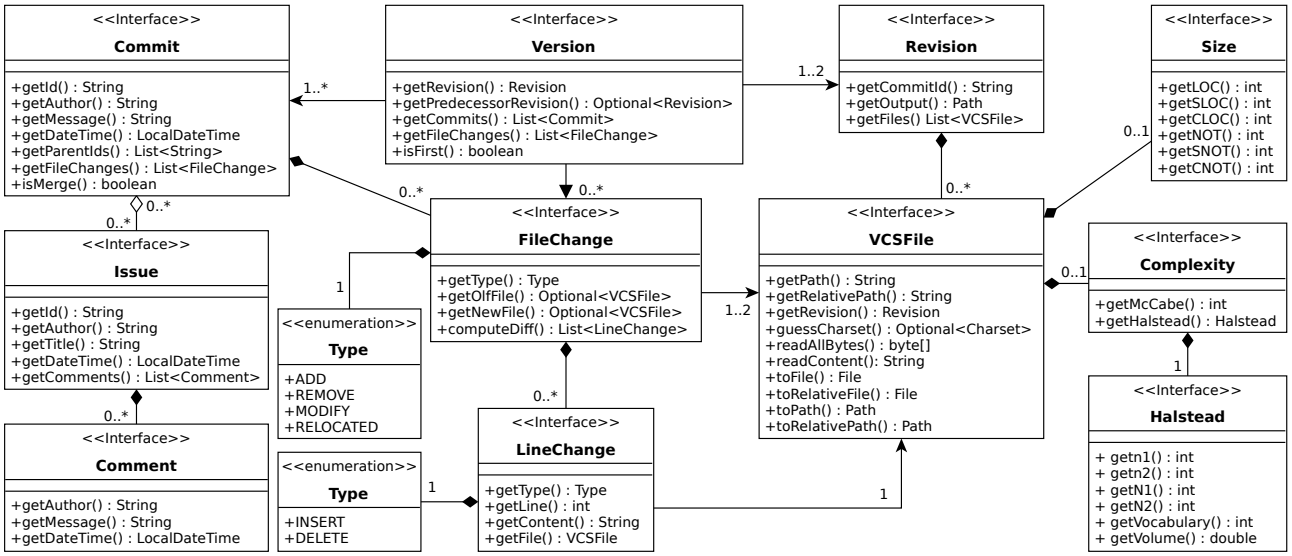

Figure 1: The data model of LibVCS4j.

3 Extraction Engines

By using extraction engines, analysis tools are able to query different repository systems under a common API and parse the output provided by LibVCS4j to perform the actual analysis for each revision desired. Two different engines are available: (1) the *VCSEngine* that allows to process version control systems and, to that effect, supply instances of the data model described in section 2, and (2) the *ITEngine* that is capable of extracting issues from an issue tracker. Figure 2 shows an UML diagram depicting the provided extraction engines. In the following, we discuss the engines in more detail.

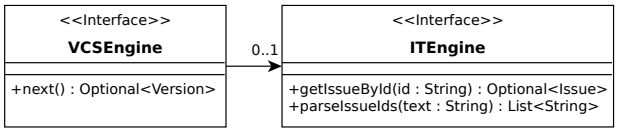

Figure 2: The extraction engines provided by LibVCS4j. A *VCSEngine* allows to process a version control system and may use an *ITEngine* to automatically extract issues referenced in commit messages.

VCSEngine This engine is used to abstract from the concrete version control system and provides methods to checkout a repository commit wise. In order to process a version control system, the required repository routines are delegated to the underlying VCS engine. Afterwards, the returned output is mapped to our data model. For example, the Java library JGit[3] is used to clone Git repositories from external URLs, checkout commits according to the user configuration, and map the reported file changes to *FileChange* instances.

[3]https://www.eclipse.org/jgit

ITEngine While a *VCSEngine* delivers one version after another, an *ITEngine* provides access to the corresponding issue tracker. As with the *VCSEngine*, the required operations to parse issues from an issue tracker are delegated using existing libraries and APIs. *ITEngines* may be used as a standalone tool or as part of a *VCSEngine* to automatically extract issues referenced by commits when processing a VCS.

4 Conclusion

We presented LibVCS4j, a Java programming library for repository mining with a common API for different version control systems and issue trackers. The library allows existing tools to make subsequent analyses independent from particular repository systems and provides support for Git, Mercurial, Subversion, Github, and Gitlab. However, it is not limited to this engines. In future work, we plan to extend LibVCS4j to cover further systems, such as CVS, Bazaar, Bugzilla, and Jira.

References

[1] N. Göde. Evolution of type-1 clones. In *2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 77–86, Sept 2009.

[2] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk. When and why your code starts to smell bad. In *Proc. of the 37th ICSE*, pages 403–414. IEEE Press, 2015.

[3] C. Vendome, M. Linares-Vasquez, G. Bavota, M. D. Penta, D. German, and D. Poshyvanyk. License usage and changes: A large-scale study of java projects on github. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 218–228, May 2015.