API-related Developer Profiling
(Extended Abstract)*

Hakan Aksu and Ralf Lämmel
Software Languages Team, University of Koblenz-Landau, Germany

Abstract

We analyze the version history of software projects to determine API-related profiles of software developers. To this end, we identify API references in source-code changes and aggregate such references through suitable metrics that provide different views on the API usage per developer so that certain conclusions regarding developer experience or comparisons between developers become feasible. We apply this approach in a case study for the open-source project JHotDraw.

1 Motivation

Knowledge of developer profiles (or experience or skills) is clearly valuable, e.g., for hiring or program managers. For instance, a hiring manager who is filling a position in a specific project may want to match the developer skills with the skills required for the project. A program manager who is reassigning sparse resources across several projects may want to validate that developers are reassigned in a way that all projects are still sufficiently staffed in terms of required skills.

Developer skills may be determined, in principle, by means of interviews, questionnaires, assignments, or analysis of available social (coding) network information (such as GitHub, topcoder, or StackOverflow). We describe an approach that analyzes API usage in source-code changes over the timeline of a project so that API-related developer profiles can be aggregated on the grounds of suitable metrics. The individual APIs in a project may also be mapped to more abstract domains [4] (such as GUI or XML or database programming), thereby permitting a discussion of developer profiles at a higher level of abstraction.

2 API-usage analysis

Without loss of generality, our approach has been implemented for Java as the source-code language and subversion as the version control system. We use the metamodel of Fig. 1 for data extraction. For what it matters, it is implemented as a relational database (relying on *Java DB*). The metamodel shows how APIs consists of API packages, how these packages declare certain API elements (e.g., classes and methods), how APIs are associated with domains, how repositories consist of files, which files have changed, which specific lines have changed, and how these changes are associated with APIs and elements thereof on the grounds of analyzing the changed lines.

The analysis relies on several extractors. The *SVNRepositoryExtractor* iterates over all commits and extracts *Version* information like developer name, revision number, and commit message, the *ChangedFile* information, and the *ChangedLine* information. As a result, every changed line can be associated with a specific developer. Data cleaning is applied so that, for example, bulk moves are excluded, as such changes would otherwise lead to severely imprecise results.

The *ClassExtractor* extracts API packages and elements from a given *.jar* file for an API. In our implementation, as a concession to scalability, we only maintain information about the latest version of an API, which may affect precision and recall.

The *APIUsageExtractor* extracts API package imports from changed files and 'potentially' referenced API elements from changed lines. A lexical approach is used in that changed lines are tokenized and the extracted names are intersected with API elements from imported API packages, as known due the *ClassExtractor*. In this manner, the underlying software projects do not need to be built, which is often difficult for larger projects and specifically older versions thereof. However, the lexical approach also challenges the precision of the analysis; this is not a severe problem in the case study that we performed.

3 Profiling metrics

For brevity, we only mention a few per-developer metrics, also without motivating them deeply:

- Given an API, the number of distinct API elements that are referenced by changed lines, over all commits by the developer.
- Given an API, the largest number of methods changed in a single commit by the developer and referencing elements of the API.
- The number of APIs of which elements were referenced in changed lines, over all commits by the developer.

*A comprehensive description of this research is being prepared [1, 2] and made available online: `http://softlang.uni-koblenz.de/apidevprof/`

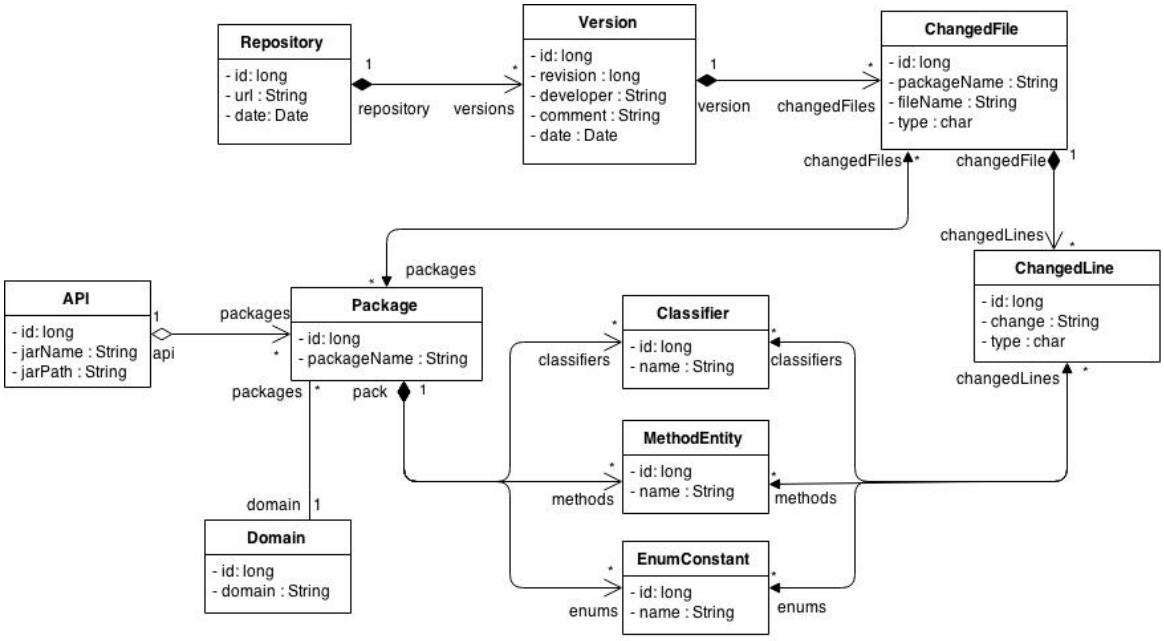

Figure 1: Metamodel for the underlying API-usage analysis.

The idea is that developers are to be compared in terms of these metrics. Clearly, these metrics need to be configured with thresholds and normalized in certain ways to permit useful comparisons.

4 Related Work

Software analysis related to APIs has received much attention by research in the last few years. For example, our team and collaborators have analyzed API usage to understand what API facets are used to what extent in what parts of a project and also the combination of APIs involved [4]. Analyzing API usage may also be useful to complement API documentation [6]. Evolution-aware analyses are also common. For instance, the evolution of an API may be analyzed to guide the implied migration work on projects that use an API [5].

Our current work is particularly concerned with linking API usage to developers and aggregating profiling information regarding experience or skills. Thus, our work is closely related to any effort on mining software repositories that takes properly into account developers. For instance, there is research on analyzing interactions between distributed open-source software developers and leveraging data mining techniques so that developer roles can be derived [7]. Another approach [3] applies statistical topic modeling to source code, thereby providing a basis for determining developer competencies, developer similarity, and others.

5 Case study

We analyzed the subversion repository of *JHotdraw* with its version history of 15 years (2000-2015) and 800 versions. There are more than 17K changed files and more then 650K changed lines. We identified 47 APIs and grouped them in 18 programming domains. We associated changes with 11 developers. We determined the profiling metrics such as those mentioned above. These metrics provide API-related quantitative insight into developer activities, ultimately profiling ('sorting') the developers in terms of their API-related skills; see [1, 2] for details.

References

[1] H. Aksu. Evolution-aware API analysis of developer skills, Mar. 2015. Master's thesis. University of Koblenz-Landau. Computer Science Department.

[2] H. Aksu and R. Lämmel. API-related Developer Profiling. Draft. Unpublished. To be submitted., 2015.

[3] E. Linstead, P. Rigor, S. K. Bajracharya, C. V. Lopes, and P. Baldi. Mining Eclipse Developer Contributions via Author-Topic Models. In *Proc. of MSR 2007*, page 30. IEEE, 2007.

[4] C. D. Roover, R. Lämmel, and E. Pek. Multi-dimensional exploration of API usage. In *Proc. of ICPC 2013*, pages 152–161. IEEE, 2013.

[5] W. Wu, B. Adams, Y. Guéhéneuc, and G. Antoniol. ACUA: API Change and Usage Auditor. In *Proc. of SCAM 2014*, pages 89–94. IEEE, 2014.

[6] T. Xie and J. Pei. MAPO: mining API usages from open source repositories. In *Proc. of MSR 2006*, pages 54–57. IEEE, 2006.

[7] L. Yu and S. Ramaswamy. Mining CVS Repositories to Understand Open-Source Project Developer Roles. In *Proc. of MSR 2007*, page 8. IEEE, 2007.