Information and Software Technology 73 (2016) 81–100

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Understanding the API usage in Java

Dong Qiu[a], Bixin Li[a,*], Hareton Leung[b]

[a] School of Computer Science and Engineering, Southeast University, Nanjing, China
[b] Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 7 September 2015
Revised 25 January 2016
Accepted 25 January 2016
Available online 3 February 2016

Keywords:
API usage
Empirical study
Java

ABSTRACT

Context: Application Programming Interfaces (APIs) facilitate the use of programming languages. They define sets of rules and specifications for software programs to interact with. The design of language API is usually artistic, driven by aesthetic concerns and the intuitions of language architects. Despite recent studies on limited scope of API usage, there is a lack of comprehensive, quantitative analyses that explore and seek to understand how real-world source code uses language APIs.

Objective: This study aims to understand how APIs are employed in practical development and explore their potential applications based on the results of API usage analysis.

Method: We conduct a large-scale, comprehensive, empirical analysis of the actual usage of APIs on Java, a modern, mature, and widely-used programming language. Our corpus contains over 5000 open-source Java projects, totaling 150 million source lines of code (SLoC). We study the usage of both core (official) API library and third-party (unofficial) API libraries. We resolve project dependencies automatically, generate *accurate resolved* abstract syntax trees (ASTs), capture used API entities from over 1.5 million ASTs, and measure the usage based on our defined metrics: *frequency, popularity* and *coverage*.

Results: Our study provides detailed quantitative information and yield insight, particularly, (1) confirms the conventional wisdom that the usage of APIs obeys Zipf distribution; (2) demonstrates that core API is not fully used (many classes, methods and fields have never been used); (3) discovers that deprecated API entities (in which some were deprecated long ago) are still widely used; (4) evaluates that the use of current compact profiles is under-utilized; (5) identifies API library coldspots and hotspots.

Conclusions: Our findings are suggestive of potential applications across language API design, optimization and restriction, API education, library recommendation and compact profile construction.

1. Introduction

Syntax and semantics define a programming language. Application Programming Interfaces (APIs) facilitate its use. Most of today's software projects heavily depend on the use of API libraries [1]. They improve code reuse, reduce development cost and promote programmers' productivity. However, API design has been artistic and biased, driven by aesthetic concerns and the intuitions of API designers. They usually have limited knowledge on how programmers actually use the API, which leads to many unnatural and rarely used API features being introduced, while not some expected ones [2,3]. Meanwhile, the ever-growing APIs (increasing features have been introduced) remain a significant barrier to novice programmers [4]. In addition, API libraries have become one of the most influential factors for the choice of programming languages [5]. Poor design of the APIs increases the learning curve for developers and greatly influence their productivity. Therefore, it is significant to understand the actual usage of the current API libraries, and optimize the designs to promote API *usability* for programmers.

Studying how a large number of real-world programs use APIs can help validate or disprove the many popular "theories" concerning what APIs are most adopted, most useful, easiest to use; whether APIs have been fully used by the programmers, *etc.* that abound concerning programming in popular literature and on the Internet. For language education, the gap between APIs and their actual usage may guide pedagogy, giving teachers insight into what is common (and perhaps should be) and rare (and perhaps should not be). It also guides novice programmers to select a proportionally smaller fraction, *i.e.* most essence of the entire APIs to reduce the cost of learning. Language API designers may leverage data on

* Corresponding author. Tel.: +86 25 52090877; fax: +86 25 52090879.
E-mail addresses: dongqiu@seu.edu.cn (D. Qiu), bx.li@seu.edu.cn (B. Li), hareton.leung@polyu.edu.hk (H. Leung).

http://dx.doi.org/10.1016/j.infsof.2016.01.011
0950-5849/

actual API usage to optimize the design of API libraries, *e.g.* simplifying unpopular APIs and identifying unused APIs that could be eliminated. In addition, API usage analysis is crucial in mining API usage patterns [6–9], and offers supports for API migration [10,11]. It also produces a positive effect in software maintenance [12].

To this end, we perform a large-scale empirical study on a diverse corpus of over 5000 real-world Java projects to gain insight into how APIs are used in practice. We retrieve project dependencies with the aid of Maven [13], generate *accurate resolved* abstract syntax trees (ASTs) for approximately 150 million SLoC, capture used API entities (*i.e.* packages, classes, methods and fields) from over 1.5 million ASTs, and measure the usage based on our defined metrics: *frequency* (whether an API has been frequently used), *popularity* (whether an API has been widely used) and *coverage* (whether an API has been fully used). We analyze almost all the API libraries that are adopted by practical projects, including both core API and third-party APIs. Besides, we investigate some extra issues, *e.g.* construction of API subsets and selection of the versions of the third-party APIs. In summary, this paper makes the following contributions:

- It presents a large-scale, comprehensive, empirical analysis of the use of APIs in a modern programming language, namely Java;
- This is the first work to deeply study both core API and third-party APIs, including the use of deprecated API entities. It is also the first to study how API usage guide the design of the compact profiles (*i.e.* subset of APIs);
- Some interesting results are demonstrated: (1) 1% of the most-used packages account for 80% of all API usage, while 70% least-used packages are used $<$ 0.5% of all API usage and 50% only $<$ 0.1%; (2) 15.3% of the classes, 41.2% of the methods and 41.6% of the fields from the core API are never used; (3) 9.5% of the packages have all subordinative methods never used and 29.2% of the classes have all subordinative methods never used; (4) 51.1% of deprecated classes, 43.5% of the deprecated methods and 18.1% of the deprecated fields from the core API have been adopted.

Taken together, our results permit API designers to empirically consider whether the design of the API facilitates programmers' development based on their actual usage. Our study also identifies both *hotspots* (*i.e.* frequently and widely used APIs) and *coldspots* (*i.e.* rarely and narrowly used APIs) to inform programmers to selectively learn and adopt the APIs. For example, if the APIs are never used, alerting programmers to use them cautiously in practical development is indispensable. In addition, the results assist to construct appropriate subsets of the APIs, that can be employed in either resource-constrained devices or high security environment. We believe that our work enables data-driven language API design, optimization and simplification, analogous to how Cocke's study at IBM in the 1970s on the actual usage of CISC instructions eventually led to the RISC architectures [14].

2. Methodology

This section first discusses the research questions studied, presents the basic information of the corpus used in this study then, and illustrates the process of how we set up and perform the experiments.

2.1. Research questions

The goal of this study is to answer the key research question: *How programming language APIs are used in real open-source projects*. To better investigate the question, we focus on the following dimensions:

Table 1
An overview of the Java corpus.

Corpus summary	
Repository	Github
No. of projects	5185
No. of files	1,595,600
Source lines of code	152,341,840
No. of imports	12,518,834
No. of class use	75,076,400
No. of field use	34,149,616
No. of method use	59,225,800
No. of unique class use	2,034,177
No. of unique field use	5,031,510
No. of unique method use	5,403,540

Global view of API usage. Most of current software projects heavily depend on the use of API libraries. Understanding the API usage *provenance* can provide an overview of API use distribution, *i.e.* how much of the API entities are reused from existing APIs (core APIs or third-party APIs) and how much of them are designed and created specific to projects. We are also interested in investigating how much of the API entities are adopted to construct a project in general and further validate whether the scale of the software is correlated with the API usage. In addition, we desire to confirm the conventional wisdom that the use of API entities obeys Zipf distribution.

Core API usage. Core API library is essential API that facilitates the use of the programming language, which is ordinarily developed by official organizations which maintain such programming language (*e.g.* Java SE Development Kit, *i.e.* JDK from Oracle [15]). However, as new features have been introduced increasingly, the scale of the core API library is growing rapidly, consuming more resources for devices and increasing the learning curve for novice programmers. It is significant to understand the *utilization* of the core API, *i.e.* whether all API entities from the core library have been fully used. The introduction of a new concept, *compact profiles*, which are subsets of the entire core API, motivates us to inspect the *utilization* of compact profiles analogously. In addition, identifying *hotspots* and *coldspots* can be suggestive of optimizing the design of current core APIs and guiding novices to learn the essence preferentially. We also investigate the use of *deprecated* API entities.

Third-party API usage. Third-party API libraries are supplements to the core API library, providing extra functionalities that are not supported by the core API or analogous functionalities with preferable implementations. Many of them are developed and maintained by reputable commercial companies (*e.g.* `guava` from Google) or open-source communities (*e.g.* `commons-*` from Apache). We are interested in investigating how heavily a project depends on third-party APIs, *i.e.* how much of third-party API libraries are required to construct a project in general. In addition, a library is available in multiple versions. It would be interesting to figure out how many distinct versions a typical library has in general. It is also significant to investigate how programmers select and adopt concrete versions.

2.2. Gathering the corpus

Our large-scale corpus consists of 5185 (including over 1.5M Java files and 15M non-comment lines of code) open-source and real-world Java projects whose source code is available from Github, one of the most popular repository hosting services. We rigorously select applications based on the *popularity* by synthetically considering their size of *watchers, stars* and *forks* provided by Github. Table 1 lists the corpus summary information. The corpus is diverse, covering various application domains and size. It