A Dataset for Maven Artifacts and Bug Patterns Found in Them

Vaibhav Saini, Hitesh Sajnani, Joel Ossher, Cristina V. Lopes
Bren School of Information and Computer Sciences
University of California, Irvine, USA
{vpsaini, hsajnani, jossher, lopes}@uci.edu

ABSTRACT

In this paper, we present data downloaded from Maven, one of the most popular component repositories. The data includes the binaries of 186,392 components, along with source code for 161,025. We identify and organize these components into groups where each group contains all the versions of a library. In order to asses the quality of these components, we make available report generated by the FindBugs tool on 64,574 components. The information is also made available in the form of a database which stores total number, type, and priority of bug patterns found in each component, along with its defect density. We also describe how this dataset can be useful in software engineering research.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Miscellaneous

General Terms

Experimentation; Measurement; Reliability

Keywords

Software Quality; FindBugs; Maven; Empirical Software Engineering; Empirical Research

1. INTRODUCTION

The growth of the open source software movement has greatly enhanced the opportunities for reuse in software development, as it has generated a large amount of freely available and high quality source code [1, 6]. While reuse occurs in many forms, the most widely accepted and promoted form is software component reuse, in which developers reuse existing binary components. These components are specifically designed with reuse in mind, and their reuse can decrease development time while improving product quality [7, 2]. When developing new software, it is now routine to search for open source components to reuse instead of implementing functionality from scratch [8].

The management of these external dependencies can be a significant challenge, and is complicated by components that themselves use other components. The result is that if a project depends on one component directly, it depends on many more indirectly. Locating all of these transitive dependencies can be difficult, as they are not always packaged with or clearly indicated by their consumer component.

Artifact repositories are a popular component or library management solution, as they provide a framework for collecting and organizing external artifacts. The research community has conducted empirical studies from studying selection criteria of these artifacts, to their usage, to asses their quality and popularity, and also to generate these libraries automatically.

Although these studies are useful, conducting them on a large-scale study has always been a challenge. The primary challenge is simply collecting the raw data. Components are hosted in a variety of different locations, requiring custom full- edged web-crawling infrastructures to locate. Collecting even a small fraction of the ecosystem takes a significant development investment, in addition to computing, storage and temporal resources. Due to this, few research groups have the necessary data to conduct such large scale empirical studies on these libraries.

In this paper, we present data downloaded from Maven, one of the most popular component repositories. The data includes the binaries of 186,392 components, along with source code for 161,025. We identify and organize these components into groups where each group contains all the versions of a library. In order to assess the quality of these components, we make available report generated by the FindBugs tool on 64,574 components. The information is also made available in the form of a database which stores total number, type, and priority of bug patterns found in each component, along with its defect density. We also describe how this dataset can be useful in software engineering research. We describe couple of studies where the dataset has been used and posit that this dataset will enable researchers to conduct plethora of large scale empirical studies involving components.

2. BACKGROUND

2.1 Maven Artifact Repository

The Maven Central Repository is a large collection of categorized Java artifacts, containing many popular third-party Java components. Maven's structure is essentially that seen in Figure 1. At the top level, the repository contains compo-

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597134

nents, such as junit. Each component is broken into multiple versions, such as junt 4.9 and junit 4.10. Finally, each component version is associated with some number of artifacts, in this case jar files like junit-4.9.jar and junit-4.9-dep.jar.

Each component is uniquely identified by a groupId and an artifactId, and contains multiple versions. Each version can be associated with an arbitrary number of artifacts, such as jar files.

2.2 FindBugs

FindBugs is an open source static analysis tool that analyses Java class files looking for programming defects. We use FindBugs to assess the quality of Maven Components. FindBugs reports nearly 300 different bug patterns by analysing the bytecode of the Maven components.

Each bug pattern is grouped into a category which determines the nature of bug pattern and its level of threat (see Table 1 for details). The bug pattern is also assigned a high, medium or low priority determined by heuristics unique to each pattern, and are not necessarily comparable across bug patterns.Priority describes how likely the bug pattern instance is to actually be a bug.

3. DESCRIPTION OF DATASETS

The dataset consists of 3 major parts as follows:

First, a tarball containing 186,392 Maven artifacts. It contains binaries for all and source code for 161,025 artifacts. The size of tarball is ≈ 80 GB.

Second, a tarball containing bug reports generated by the FindBugs tool for 67,273 artifacts. The report is in XML format and contains detailed description about each bug instance found in the artifact. The description includes type, priority, source location where the bug was found, method/field associated with the bug. Figure 2 shows the snapshot of the of xml bug report generated by FindBugs tool.

This tarball also includes a file named *jar.properties* for each artifact which contains the following meta-information about the artifact.
name: The name of the artifact as given in Maven repository.
source: The source repository of the artifact.
group: The group name of the artifact as given in Maven repository. The *group* and *name* together can uniquely identify a component.
hash: The MD5 hash generated using *group* and *name* to uniquely identify the artifact.
has_source: Flag indicating if source code is available for this artifact.

Third, a MySql database which contains two tables:
(i) maven_artifacts (ii) maven_artifacts_findbugs_bug_pattern. Figure 3 shows the schema of these two tables along with the relationship between them.
maven_artifacts contains information about Maven artifacts. Most of the fields in this table correspond to the fields in jar.properties file described above. However, there are two new fields *version* and *description.*
version: The version number of the artifact as given in the Maven repository.
description: This is an optional field containing the original project description as given in the Maven repository.
maven_artifacts_findbugs_bug_pattern table contains the information about the bug patterns identified by the FindBugs tool for 64,574 Maven artifacts. Each row in this table represents a bug-pattern found in a Maven artifact. There are total 10,308,818 rows in this table, each with the following fields.
id: The unique identifier of a row. This is the primary key for the table.
type: Type is a string indicating what kind of bug it is as given by FindBugs tool.
priority: Priority indicates how likely this instance is to actually be a bug. Bug instances with priority 1 (high) and 2 (medium) are captured in this table.
abbrev: It is a short alphanumeric code for the bug as given by FindBugs tool.
category: The category of a bug instance as given by the FindBugs Tool (see Table 1).
project_hash: md5 hash of the Maven artifact. This is a foreign field referencing the *hash* field in the *maven_artifact* table.
project_version: Version number of the artifact given in the Maven repository.
project_size: Lines of Code (non-whitespace) in the project.
project_bug_count: Total number of bug instances found in a Maven artifact.

As shown, a simple group by *project_hash* operation will give all the bug patterns found in the artifact. However, note that the total bug count is already present in *project_bug_count* field.

4. DATASET APPLICATIONS

In this section, we describe how we have used the above dataset in couple of studies to provide readers with some intuition about the utility of this dataset in software engineering research.

4.1 Component Popularity and Quality

One of the perceived values of open source software is the idea that many eyes can increase code quality and reduce the amount of bugs. This perception, however, has been questioned by some due the lack of supporting evidence.

We recently performed an empirical study [3], focusing on the relationship between the usage of open source components and their engineering quality using the above dataset. In this study, we determine the exposure (popularity) of 2,232 Maven components by calculating their usage across 47,801 open source Java projects. As a proxy of code quality, we use defect density of those 2,232 Maven components using the set of bug patterns reported by FindBugs. We then looked for correlations between popularity and defect density.

We were unable to find significant correlations between the Maven components' popularity and their defect density. Surprisingly, we found a strong positive correlation (0.42) between popularity and defect density in the subset of the 24 most popular Maven components. We also found a weak positive correlation (0.13) between popularity and defect density for the 1,569 Maven components with defect density lower than 10. Finally, we found a moderate correlation (0.32) between popularity and size.

4.2 Bottom-up Construction of Structured Artifact Repository

Artifact repositories, such as the Maven Central Repository, are manually curated, and constructed in a top-down

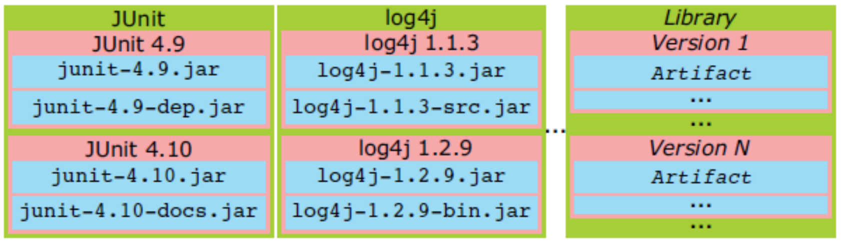
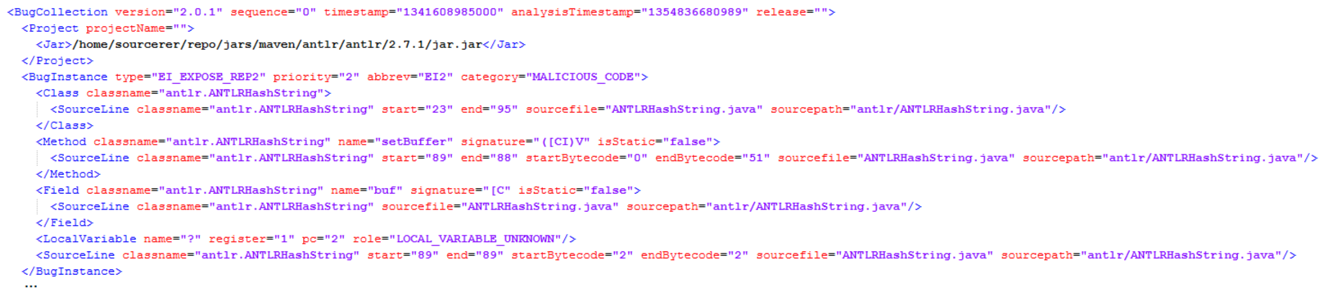


Figure 1: Artifact Repository Structure



```xml
<BugCollection version="2.0.1" sequence="0" timestamp="1341608985000" analysisTimestamp="1354836680989" release="">
  <Project projectName="">
    <Jar>/home/sourcerer/repo/jars/maven/antlr/antlr/2.7.1/jar.jar</Jar>
  </Project>
  <BugInstance type="EI_EXPOSE_REP2" priority="2" abbrev="EI2" category="MALICIOUS_CODE">
    <Class classname="antlr.ANTLRHashString">
      <SourceLine classname="antlr.ANTLRHashString" start="23" end="95" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Class>
    <Method classname="antlr.ANTLRHashString" name="setBuffer" signature="([CI)V" isStatic="false">
      <SourceLine classname="antlr.ANTLRHashString" start="89" end="88" startBytecode="0" endBytecode="51" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Method>
    <Field classname="antlr.ANTLRHashString" name="buf" signature="[C" isStatic="false">
      <SourceLine classname="antlr.ANTLRHashString" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Field>
    <LocalVariable name="?" register="1" pc="2" role="LOCAL_VARIABLE_UNKNOWN"/>
    <SourceLine classname="antlr.ANTLRHashString" start="89" end="89" startBytecode="2" endBytecode="2" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
  </BugInstance>
  ...
```

Figure 2: Xml Schema for report generated by FindBugs

Table 1: Categories of bug patterns reported by FindBugs

Category	Description	Example
Bad Practice	Violations of recommended and essential coding practice. Examples include hash code and equals problems, cloneable idiom, dropped exceptions, serializable problems, and misuse of finalize	Using “==” to compare string objects
Style	Code that is confusing, anomalous, or written in a way that leads itself to errors	Dead local stores, switch fall through, unconfirmed casts, and redundant null check of value known to be null
Correctness	An apparent coding mistake resulting in code that was probably not what the developer intended	Returning a reference to mutable object may expose internal representation
Multithreaded correctness	Thread synchronization issues	Method does not release lock on all or code paths that end up in lock not being released
Internationalization	Use of non-localized methods	Use of non-localized String.toUpperCase
Performance	Inefficient memory usage/buffer allocation, usage of non-static classes	Creating a new String(String) constructor
Malicious code	Variables or fields exposed to classes that should not be using them	Returning a reference to mutable object may expose internal representation
Security	Similar to malicious code vulnerability	Passing a dynamically created string to a SQL statement

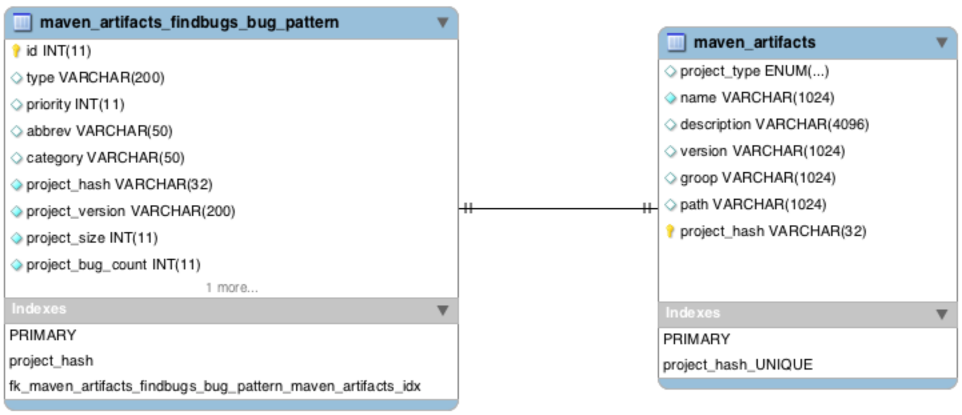


Figure 3: Database Schema for Maven artifacts

manner. This makes expanding their scope, and keeping them up to date, a time consuming process. For example, our investigation of the scope of the Maven Central Repository showed that it is missing over 60% of externally referenced types in a large collection of 48,000 open source Java projects.

To address this issue, we created Astra [5], an algorithm for the automated bottom-up construction of structured artifact repositories. Astra takes as input a collection of unknown library artifacts, such as jar files, and returns a structured artifact repository. The resulting repository contains components, which are divided into versions and associated with individual artifacts. This bottom-up construction is accomplished by analysing the co-occurrence of types between the component artifacts. Our evaluation demonstrates that Astra generates a repository with 77% similarity to the Maven Central Repository when provided the same base artifacts. We used the above dataset to evaluate the accuracy of Astra.

While this is not the complete list of work where this dataset has been useful, we believe that the above described works give a flavor of how this dataset has been useful in both empirical and algorithmic work in software engineering research.

5. DATA ACCESSIBILITY

All the datasets are accessible through our website: http://mondego.ics.uci.edu/projects/mavenpopandqual/ In the past researchers have also sent us hard drives or disks and we have provided them with the data. We will continue doing this as well as sometimes due to slow connections downloading such huge dataset becomes difficult.

6. CHALLENGES AND LIMITATIONS

In order to download the Maven repository, we wrote a basic crawler to perform the download. However, we had to soon make it fault resilient because the process ran for few days with intermittent failures. We made use of rsync utility, as it is good at picking up from where it left off. However, since rsync option is no longer available, it makes getting an updated version virtually impossible.

To produce the bug reports by FindBugs on the repository, we wrote java tools to iterate through the repository, run FindBugs on a jar (using Runtime.exec utility), and then parse the output XML to pull out the information in the database. It took days, with intermittent failures, to execute FindBugs on such a large repository.

7. ACKNOWLEDGMENTS

This material is based upon the work supported by the National Science Foundation under Grant No. 1018374

8. REFERENCES

[1] A. Deshpande and D. Riehle. The total growth of open source. In *Open Source Development, Communities and Quality*, volume 275 of *The International Federation for Information Processing*, pages 197–209. Springer US, 2008.

[2] P. Devanbu, S. Karstu, W. Melo, and W. Thomas. Analytical and empirical evaluation of software reuse metrics. In *Software Engineering, 1996., Proceedings of the 18th International Conference on*, pages 189–199, Mar 1996.

[3] J. Ossher. *Software Component Utilization and Software Quality Metrics.* PhD thesis, University of California, Irvine, 2013.

[4] J. Ossher, S. Bajracharya, E. Linstead, P. Baldi, and C. Lopes. SourcererDB: an aggregated repository of statically analyzed and cross-linked open source java projects. In *Mining Software Repositories*, pages 183–186, 2009.

[5] J. Ossher, H. Sajnani, and C. Lopes. Astra: Bottom-up construction of structured artifact repositories. In *Proceedings of WCRE*, 2012.

[6] D. Spinellis and C. Szyperski. How is open source affecting software development? *IEEE Software*, 21(1):28–33, January/February 2004.

[7] C. Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[8] M. Umarji, S. Sim, and C. Lopes. Archetypal internet-scale source code searching. In *Open Source Development, Communities and Quality*, volume 275 of *The International Federation for Information Processing*, pages 257–263. Springer US, 2008.