Discover Knowledge on FLOSS Projects Through RepoFinder

Francesca Arcelli Fontana[1], Riccardo Roveda[2] and Marco Zanoni[1]
[1] *University of Milano-Bicocca, Milano, Italy*
[2] *KIE S.r.l, Milano, Italy*

Keywords: Knowledge Discovery, Project Discovery, FLOSS Projects, Full-text Search.

Abstract: We can retrieve and integrate knowledge of different kinds. In this paper, we focus our attention on FLOSS (Free, Libre and Open Source Software) projects. With this aim, we introduce RepoFinder, a web application we have developed for the discovery, retrieval and analysis of open source software. RepoFinder supports a keyword-based discovery process for FLOSS projects through google-like queries. Moreover, it allows to analyze the projects according to well-known software metrics and other features of the code, and to compare some structural aspects of the different projects. In the paper, we focus on the discovery capabilities of RepoFinder, evaluating them on different project categories and comparing them with a well-known search engine as Google.

1 INTRODUCTION

Through the Web we are able to retrieve a huge amount of data and information of different kinds. Among them FLOSS (Free, Libre and Open Source Software) projects and applications are receiving a continuous increasing interest across different communities, both in the industry, public administration and research institutions. FLOSS projects are managed on dedicated web platforms, called Code Forges (Squire and Williams, 2012), designed to collect, promote, and categorize FLOSS projects, and to support development teams. Some examples are Launchpad, Google Code, SourceForge, Github, Bitbucket and GNU Savannah. Code Forges provide tools and services for distributed development teams, e.g., version control systems, bug tracking systems, web hosting space. Currently, Github hosts more than ten million software components, and is probably the largest. SourceForge is one of the first Code Forges, created in 1999, with more than 425,000 projects, all of them fitting a categorization, while Google Code hosts about 250,000 projects.

To find a project satisfying specific requirements, we can start from a simple query on a generic search engine or on each Code Forge. In this case, the risk is receiving many useless answers, because FLOSS project names are often ambiguous and difficult to recover. Even retrieving the source code of a particular project can be difficult, because it may be managed by different portals and with different versioning technologies, e.g., Git, SVN, Mercurial (HG), CVS, GNU Bazaar. Moreover, to choose a project for integrating a FLOSS component, it is often necessary to spend significant time inspecting the project activity, releases, and web forums or communities explaining strength and weaknesses of these projects. In this traditional way of evaluating new components to integrate in a software, there is very little knowledge of the quality of the project, design, code and development team.

We started facing these issues, by developing a web application, called RepoFinder. Its aim is to be a platform able to support the discovery, retrieval, analysis and comparison of FLOSS projects. In the current development stage, we started supporting FLOSS project discovery, retrieval and analysis. RepoFinder is currently able to perform the following tasks:

- Crawling of widely known Code Forges (Github[1], SourceForge[2], Launchpad[3] and Google Code[4]), through their APIs or by parsing their web pages. During this activity, projects contained in the Code Forges are indexed, and tagged with labels and other available metadata (e.g., tags, forks, stars), depending on the particular Code Forge.

[1] https://github.com
[2] http://sourceforge.net
[3] https://launchpad.net
[4] https://code.google.com

Arcelli Fontana F., Roveda R. and Zanoni M..
Discover Knowledge on FLOSS Projects Through RepoFinder.
DOI: 10.5220/0005156704850491
In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval* (KDIR-2014), pages 485-491
ISBN: 978-989-758-048-2

Currently, we indexed more than 450,000 projects and found more than 20,200 labels.

- Browsing projects through a search engine based on simple google-like queries. It is possible to search into the projects' descriptions, names and all the available metadata. The code of the found projects can be retrieved through their version control system repository.
- Analysis of the projects' code. We integrated external software for the computation of metrics and the detection of code smells (Fowler, 1999; Arcelli Fontana et al., 2012) in Java code. Currently, the following tools are integrated: Checkstyle (Ivanov and Sopov, 2014), PMD (Dangel, 2014) for code smell detection, NICAD (Roy and Cordy, 2008) for clone detection, JDepend (Clarkware Consulting Inc., 2014), Understand (Scientific Toolworks, Inc., 2014), JavaNCSS (Lee, 2014) and another tool developed at our Laboratory (ESSeRE-Evolution of Software Systems and Reverse Engineering Lab), called DFMC4J, for metrics computation.
- Integration of data gathered by the code analyzers in a dataset, to be used for statistical analyses. In the resulting dataset, data extracted by the analyzers is merged in a uniform format, allowing, e.g., to compare the analyses performed by different tools on the same code artifact.

In this paper, we describe the functionalities and the architecture of RepoFinder, we summarize the amount and characteristics of the projects we indexed during the crawling phase, and we evaluate the discovery capabilities of the tool, by simulating a FLOSS project discovery use case, repeated on different project categories.

The paper is organized through the following sections: In Section 2 we describe some related work, in Section 3 we introduce the main functionalities offered by RepoFinder and its software architecture; in Section 4 we show the data we collected in the crawling phase; in Section 5 we evaluate a project discovery use case. Finally, in Section 6 we conclude and outline some future developments.

2 RELATED WORK

Other applications exist with functionalities similar to the ones of RepoFinder. We cite below some of them, outlining the main similarities and differences with our tool.

- SonarQube[5] (Arapidis, 2012; Campbell and Papapetrou, 2013)
  - Similarities: performs different types of analysis, integrating different external tools; exploits also Checkstyle and PMD.
  - Differences: it can be used like a plugin of Eclipse; SonarQube does not support projects discovery, but every project must be added to the system. In addition to Java, it allows analyzing other languages by integrating different plugins.
- SECONDA (Prez et al., 2012)(Software Ecosystem Analysis Dashboard)
  - Similarities: performs analyses and computes metrics on locally stored Git repositories. Uses a web GUI to visualize the results.
  - Differences: analyzes C code and allows to download code only using Git.
- Ohloh.net (Black Duck Software, 2014)
  - Similarities: provides a search engine that use tags and labels to improve performance.
  - Differences: computes few metrics (LOC and number of commits). In Ohloh, any user can freely modify labels, tags and other information about the indexed site. RepoFinder, instead, directly reports the information published on the official Code Forge page.
- GlueTheros (Robles et al., 2004)
  - Similarities: calculates metrics code repositories repository and shows results using a Web GUI.
  - Differences: supports CVS only. In addiction to Java, it analyzes other languages. It does not support the discovery of new projects.
- Complicity (Neu et al., 2011)
  - Similarities: computes metrics from a project's repository. Extracts all projects from a single repository. Uses a dashboard to visualize results.
  - Differences: computes only few repository's metrics (e.g., number of commits, number of files). It does not support project discovery, but is initialized with an input project ecosystem. The supported analyses focus on the contributions the developers give to the ecosystem, rather than the source code and its quality.

There are also a number of datasets (Van Antwerp and Madey, 2008; Howison et al., 2006; Gousios, 2013) that gather data coming from different Code

[5] http://nemo.sonarqube.org/

Forges, which could be integrated in the crawling and analysis processes of RepoFinder. However, their availability is variable and, unfortunately, some of these data-collection projects are closed.

3 REPOFINDER

In this section, we introduce the overall architecture of RepoFinder and we briefly describe its main functionalities.

3.1 RepoFinder Functionalities

RepoFinder provides five different functionalities:

- *Crawling*: it allows to find FLOSS projects on the different Code Forges, and to save, update and index their information in a data store local to the application.
- *Indexing*: it allows to obtain better results during the projects search and has been developed using the Lucene external library.
- *Pre-Analysis (Selection)*: it allows finding projects through simple queries submitted to the application; found projects can be selected for being analyzed.
- *Analysis*: allows to analyze and store data gathered with different analysis tools. The projects are built before the analysis, because some tools need projects that can be compiled.
- *Post Analysis*: analyses results are graphically visualized; it is possible to make comparison among the data extracted on different analyzed projects. Extracted data can be exported for external processing.

3.2 RepoFinder Architecture

RepoFinder allows searching FLOSS projects in Code Forges and to analyze Java projects. The overall architecture of RepoFinder is represented in Figure 1.

The architecture is client-server. The server side, implemented in Java and hosted on Tomcat 7, does all the heavy jobs and is made of different components:

- Crawler: this component performs and manages the scheduled crawling jobs. There are different crawling strategies to extract the projects' information from HTML pages, because there can be different HTML project page structures in the same Code Forge.

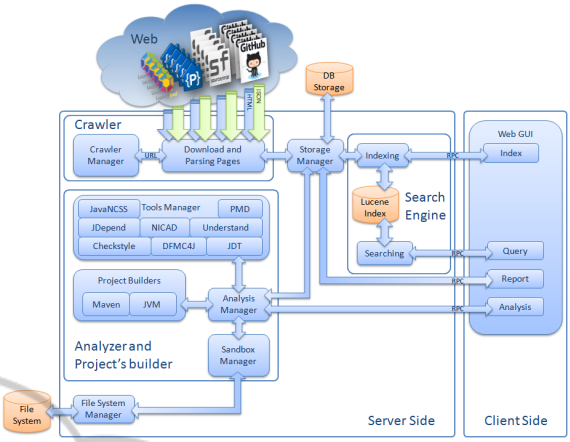

Figure 1: RepoFinder architecture

- Search engine: indexes the projects collection and performs the query execution. It exploits Lucene[6] to create and query the projects' index.
- Analyzer and project builder: downloads source code from the projects repositories; builds the projects and, after running all tools, saves all the gathered data.
- Storage manager: manages all the stored data gathered from both crawling and analyses.

The client-side is a Web GUI. It enables the user interaction with the application using a web browser. The user can search projects, run the analyses on selected projects, and show reports of the extracted data. Vaadin[7] has been used to create the GUI, and to realize the client-server communication through RPC.

Currently, analyses can be applied only to Java systems having a working Maven[8] configuration. Maven ensures that all relevant sources are included in the analyses. It also retrieves from its repositories all the library dependencies needed to compile the systems and to gather the required metrics correctly. It can provide only projects' source code, without test cases, and executing code generation tools (e.g., ANTLR, JOOQ) if required. These features allow to provide the analysis tools a consistent and compilable codebase to analyze, minimizing possible errors during the analyses. Currently, RepoFinder supports six different tools for metrics computation and code smell detection (Understand, DFMC4J, JDepend, JavaNCSS, PMD, Checkstyle, Nicad). Code smells have been defined by Fowler (Fowler, 1999) and are synthoms of problems at code or design level, that can be removed through refactoring steps.

[6]https://lucene.apache.org/
[7]https://vaadin.com
[8]http://maven.apache.org

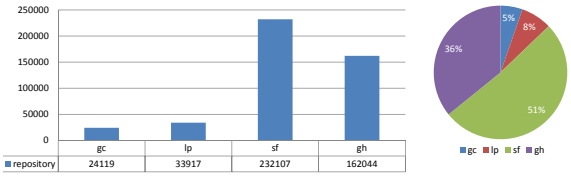

Figure 2: Number of project and division.

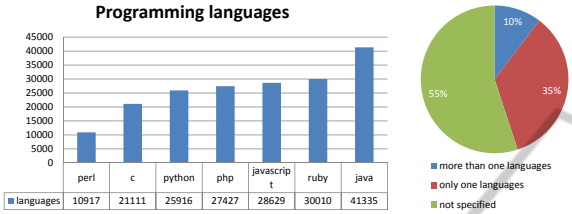

Figure 3: Languages.

4 COLLECTED DATA

Crawling Code Forges, we obtained different numbers of projects. In Figure 2 we show the number of the projects we obtained. The collected projects are written in many different programming languages. In Figure 3, we outline the number of projects we found for each language. Projects can be written in more than one language. In that case, they are counted into each language. Only 10% of the projects contain more than one language. We considered only tags and labels to determine the programming languages, so the language of about one half of the projects is unknown.

The crawling phase of the discovery process has performances that vary a lot with respect to the crawled forge and the number of projects it exposes. In Figure 4, we report the time needed for crawling in the different Code Forges. GoogleCode is the faster to index, but exposes a smaller number of projects. Github is the slowest. In fact, we were able to collect only a small fraction of Github projects, because its APIs limit the number of request for listing projects metadata to 5,000 request per hour. We are continuously updating our database by crawling the four Code Forges we support. The usage of Github is increasing, and many projects we found in other Code Forges are totally or partially migrated to Github. This appears to happen for different reasons, often for achieving more visibility.

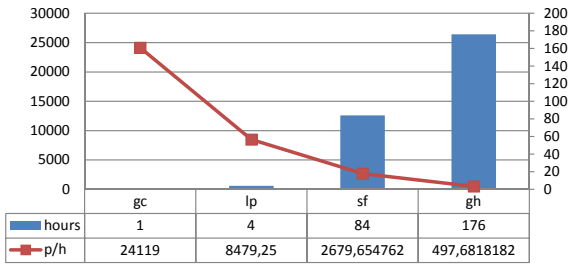

Figure 4: Crawling speed and duration.

Table 1: Keywords used for the use case experiment.

Category	Query
Office	office suites spreadsheet word processor
Java-IDE	editor text programming java ide program
Full-text	text search engine library java
Java-ORM	orm database mapping java library
Algebra	linear algebra math library java

Table 2: Results of the use case experiment.

				Google			RepoFinder		
Category	G	RF	$\wedge$	P_1	P_2	P_3	P_1	P_2	P_3
Office	23	18	4	0.57	0.30	0.30	0.61	0.22	0.22
Java-IDE	22	51	1	0.68	0.59	0.59	0.39	0.24	0.20
Full-text	44	50	5	0.84	0.52	0.39	0.50	0.22	0.18
Java-ORM	54	51	5	0.85	0.54	0.54	0.90	0.69	0.43
Algebra	42	49	9	0.79	0.26	0.26	0.73	0.61	0.49

G, RF: Num. prjs returned by Google and RepoFinder;
$\wedge$: Prjs returned by both tools; P_x: precision on criterion x

Another important source of FLOSS projects are foundations like Eclipse, Mozilla and Apache. These organizations tend to use their own infrastructure for project pages, bug tracking and code repositories. These projects are not indexed by RepoFinder, while many of them can be found anyway as mirrors on Github.

5 USER EXPERIENCE REPORT

To understand the effectiveness of RepoFinder's discovery capabilities, we simulated a use case where a user is looking for projects and uses some keywords as a search query. We used the same query in RepoFinder and in Google. For RepoFinder, we considered the first 50 results, while for Google we collected all the projects mentioned in the first 10 returned links. We recorded the time needed to understand if the collected projects are pertinent to the query and if they are active. For Google, we listed also in which position we found the references to new projects.

We replicated the use case for five different queries, reported in Table 1. The five queries are targeted to different categories of projects, each one having at least a famous FLOSS project. We sorted the

categories from the wider, addressing more general functionalities, to the narrower, addressing more specific functionalities.

For every query, we measured how many projects were discovered in common, and how many were discovered by only one of the two sources. We also measured the precision of the results using different criteria. The criteria are defined in more detail for each considered software category, but can be summarized as:

1. the project is correct if it covers the needs expressed by the query, even if in a different technology;
2. if criterion 1 is met, consider the project as correct if it is active; we considered active projects the ones having significant activity on the source code, or having releases, in year 2012;
3. if criterion 2 is met, consider the project as correct if it is implemented using the technology specified in the query, or it fits any other specific requirements of the query.

We defined these different levels of criteria because some of these considerations are difficult or impossible to be made by an automatic search engine, but are relevant to the user. Assessing precision from these three points of view, we want to give an idea of the effort the user has to put in evaluating the results. The collected results are summarized in Table 2.

A general consideration we can formulate is that the intersection between the results returned by Google and RepoFinder is very small. In the following, we discuss the results obtained on each category. For each category, we also give a specialized version of the applied criteria, where the general criteria are detailed with respect to the defined software category and the experimented query. Criterion 2 is never reported, because it does not need specialization. Specialized criteria are of increasing strictness, as in the generalized ones.

5.1 Considerations on Office Results

This software category contains open source office suites, or single word processors and spreadsheets. The most famous project in the category is LibreOffice/OpenOffice. The specialized criteria for this software category are: *1)* we consider a correct project every open source application providing at least a word processor or a spreadsheet; *3)* no restriction applied in this case.

This category is one of the oldest and most popular software categories at all. The alive projects in this category have typically a long history, and many FLOSS projects have been stopped or discontinued due to the consolidation of the main players in both the FLOSS and commercial world. The results of both Google and RepoFinder confirm this analysis. RepoFinder returned 18 projects only, while Google returned 23, but nearly half of them are commercial projects. The low precision of RepoFinder is influenced mainly by the generality of the keywords, and to the fact that many FLOSS projects in this category are not updated anymore.

5.2 Considerations on Java-IDE Results

This category contains text editors and Integrated Development Environments (IDEs) written in Java or supporting the development of Java software. Popular IDEs in this category are Eclipse, Netbeans and IntelliJ IDEA, but also smaller projects like JEdit. For this category, we specialized our evaluation criteria in this way: *1)* correct projects are text editors, programming editors, IDEs, or any software supporting editing and development; we exclude plugins for other editors or IDEs, and commercial projects; *3)* correct projects are the ones written in Java or dealing with Java development.

The results obtained on this project category are slightly better than previous ones. The search made on Google returned 22 projects only, and only one is in common with the results provided by RepoFinder. This category of projects include major players in the whole software development area, like the aforementioned ones. Eclipse, Netbeans and IntelliJ IDEA are not hosted on Code Forges, but on their own platforms. In this case, Google gives better results than RepoFinder, and this was expected, given the popularity of the returned software. Moreover, the precision on criterion 1 for Google is due only to the inclusion of commercial software in the proposed results.

RepoFinder has low performance on this query. The motivation is basically that the query is very general, and both the description and the labels of other project kinds match the same query. Examples of projects matching the query are plugins for other editors or IDEs (7 projects), editors for special files (5 projects) or miscellaneous Java libraries (4 projects). The only project in common with Google is `drjava`, an actual Java IDE. JEdit, one of the project we mentioned, is not present in the first 50 results of RepoFinder. By extending the result list, we found JEdit in the next 50 projects. This placement is due to the result contamination we just discussed.

5.3 Considerations on Full-text Results

This category contains Java libraries dealing with the indexing and search of textual data, or services having the same purpose. One of the most famous members of this category is Lucene. For this category, we specialized our criteria in this way: *1)* a project is correct if it is an open source library or service helping with the search and indexing of textual data, at any scale or complexity; *3)* a project is correct if it is a Java library or a platform-independent service.

The performances for this category are aligned to the already reported ones. Criteria 3 for Google is lower than in the previous categories. Returned results are contaminated by other kinds of text processing libraries, inactive clones or ports of Lucene, and API support for languages different form Java. The last issue is very relevant also for Google results, in addition to the presence of commercial software in the results. The impact of inactive projects in the two cases is comparable, instead.

5.4 Considerations on Java-ORM Results

This category addressed Java libraries for Object-Relational Mapping. One of the most famous projects of this category is Hibernate. For this category, we specialized the evaluation criteria in this way: *1)* we consider correct projects all the ones providing libraries exposing an API for working with a database of any kind, for any programming language; *3)* we consider correct projects only the ones providing Java libraries for mapping objects to a relational database.

The precision of both tools is comparable for this category, especially considering the amount of non-active projects returned, highlighted by the difference between the precision of criteria 1 and 2 in both cases. As for the projects missed by RepoFinder, we tested that only 22 of the 49 reported only by Google are currently indexed in RepoFinder. Among the projects we do not index there are Apache and Eclipse projects, which are hosted on their own forge. The analysis of Google results took one hour and twenty minutes. Most of the projects (51 out of 54) have been found in the first and third result, which are two pages containing categorized lists (directories) of project names. First result is a Wikipedia entry, and the third is from java-source.net. From the results, we can see that these pages contained outdated information, demonstrated by the precision change between criterion 1 and criterion 2. Finally, both Google and RepoFinder reported Hibernate in the considered result set; we consider Hibernate the most famous library in this category.

This query confirms the increasing trend in the performances of RepoFinder along with the increase of the specialization of the query. The query is similar to the Full-text one, but identifies a slightly more precise area. In both categories, there is a very famous project (Hibernate for Java-ORM and Lucene for Full-text).

5.5 Considerations on Algebra Results

The last project category we tested contains Java libraries dedicated to linear algebra, math and matrices. Some known projects in the category are JAMA, Colt, jblas. We specialized our evaluation criteria for this category as follows: *1)* a project is correct if it is a library for solving any kind of mathematics, algebra or matrix calculation; *3)* a project is correct if it provides a Java library.

This is the narrowest category we defined in this evaluation. The results are the best obtained in all categories for both Google and RepoFinder. In this case, the precision of RepoFinder is superior the precision of Google, which found a high number of inactive projects. This category does not contain a project representing a standard-de-facto, but is populated by many medium-small projects, often addressing only particular tasks. A peculiarity of this category is that some popular projects, e.g., JAMA and Colt, are actually inactive, but they keep a wide user base.

6 CONCLUSIONS AND FUTURE DEVELOPMENTS

In this paper, we described a web application we developed, called RepoFinder, which gives automated support to the discovery and selection of FLOSS projects. We reported the current stage of the crawling and indexing process we did on four Code Forges. Currently, we index more than 450,000 projects, labeled with more than 20,200 labels/tags, and written more than one hundred languages. The crawling process is more effective on some Code Forges than others. For example, Github imposes artificial speed limits to its APIs, slowing down the crawling process.

We performed some evaluations with RepoFinder, to understand its behaviour with respect to the traditional way of discovering FLOSS project through a general purpose search engine. In our evaluation, we simulated a project discovery use case, where a developer defines some keywords to look for a certain project category, and uses them to discover FLOSS

projects. We defined five categories, and verified the correctness of the discovered projects using three criteria. The discovered performances have great variability, depending also on the query. The most popular project categories we defined have low performances in RepoFinder, while, increasing the specificity of the query, performances raise.

The discovery time is one of the quality factors in RepoFinder. To collect the results on RepoFinder, it took from 5 to 15 minutes, while the same task on Google took from 1 to 2 hours. The motivation is very simple: RepoFinder reports projects in a structured and uniform way, showing the name, labels and description of projects. Web pages, instead, mostly report the name of the project, and sometimes the link to the official site, making the discovery process slower.

We did not describe in this paper the analyses we can perform on the retrieved projects, by exploiting the different tools we have integrated for metrics computation and code smell detection. We wanted to focus instead on the discovery functionalities of RepoFinder, leaving the demonstration of the analysis phases for another future work.

Regarding future work, we identified some directions in which RepoFinder's discovery functionalities can be extended:

- Online query support. As we outlined, the crawling process can be slow, also because of limitations imposed by Code Forges, and can lead to a partial exploration of the available projects. We are planning to combine our local index search with online queries submitted to the supported Code Forges. This hybrid solution should increase the chance of discovering new projects, and will leverage the existing search engines available on each Code Forge.
- Similar projects search. As we outlined in Section 5, software categories often have one or more famous projects. It happens many times that developers look for alternatives to an existing project, rather than directly searching a software by identifying its category. This is a common use case, and we plan to integrate automated support for it.

ACKNOWLEDGEMENTS

This work was partly funded by KIE S.r.l. of Milano, Italy (http://www.kie-services.com/). The authors kindly thank this society.

REFERENCES

Arapidis, C. S. (2012). *Sonar Code Quality Testing Essentials*. Packt Publishing.

Arcelli Fontana, F., Braione, P., and Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*, 11(2):5:1–38.

Black Duck Software (2014). Ohloh. www.ohloh.net.

Campbell, G. A. and Papapetrou, P. P. (2013). *SonarQube in Action*. Manning Publications Co.

Clarkware Consulting Inc. (2014). JDepend. clarkware.com/software/JDepend.html.

Dangel, A. (2014). PMD. pmd.sourceforge.net.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA. http://www.refactoring.com/.

Gousios, G. (2013). The GHTorrent dataset and tool suite. In *Proc. 10th Working Conf. Mining Software Repositories (MSR '13)*, pages 233–236, San Francisco, CA, USA. IEEE.

Howison, J., Conklin, M., and Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analyses. *Intl J. Information Technology and Web Engineering*, 1:17–26.

Ivanov, R. and Sopov, I. (2014). CheckStyle. checkstyle.sourceforge.net.

Lee, C. C. (2014). JavaNCSS. www.kclee.de/clemens/java/javancss.

Neu, S., Lanza, M., Hattori, L., and D'Ambros, M. (2011). Telling stories about GNOME with Complicity. In *Proc. 6th Intl Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2011)*, pages 1–8, Williamsburg, Virginia, USA. IEEE.

Prez, J., Deshayes, R., Goeminne, M., and Mens, T. (2012). SECONDA: Software ecosystem analysis dashboard. In *Proc. 16th European Conf. Software Maintenance and Reengineering (CSMR 2012)*, pages 527–530, Szeged, Hungary. IEEE.

Robles, G., Gonzlez-Barahona, J. M., Ghosh, R. A., and Carlos, J. (2004). GlueTheos: Automating the retrieval and analysis of data from publicly available software repositories. In *Proc. Intl Workshop on Mining Software Repositories (MSR 2004)*, pages 28–31, Edinburgh, UK. IET.

Roy, C. and Cordy, J. (2008). NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proc. 16th IEEE Intl Conf. Program Comprehension (ICPC 2008)*, pages 172–181.

Scientific Toolworks, Inc. (2014). Understand. www.scitools.com.

Squire, M. and Williams, D. (2012). Describing the software forge ecosystem. In *Proc. 45th Hawaii Intl Conf. Systems Science (HICSS-45 2012)*, pages 3416–3425, Grand Wailea, Maui, HI, USA. IEEE.

Van Antwerp, M. and Madey, G. (2008). Advances in the sourceforge research data archive (SRDA). In *Proc. 4th Intl Conf. Open Source Systems (WoPDaSD 2008)*, Milan, Italy.