



UNIVERSIDAD NACIONAL DE COLOMBIA

Automatic Multi-label Categorization of Java Applications Using Dependency Graphs

Santiago Vargas Baldrich

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2015

Automatic Multi-label Categorization of Java Applications Using Dependency Graphs

Santiago Vargas Baldrich

A thesis submitted in partial fulfillment of the requirements for the degree of:
Maestría en Ingeniería - Ingeniería de Sistemas y Computación

Advisor:

Mario Linares Vásquez, Ph.D (c)

Co-advisor:

Jairo Hernán Aponte Melo Ph.D

Research Line:

Software Engineering

Research Group:

ColSWE: Software Engineering Research Group

Universidad Nacional de Colombia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial

Bogotá, Colombia

2015

Dedication

To Mario Linares for his continuous guidance, support and patience. To my teachers at Universidad Nacional de Colombia whom I deeply respect and admire. To my competitive programming team and my friends.

To those who are no longer with me but were there just when I needed them, and to those who still are.

To my mother and brother, the two strongest persons in the world.

Abstract

Automatic approaches for categorization of software repositories are increasingly gaining acceptance because they reduce manual effort and can produce high quality results. Most of the existing approaches have strongly relied on *supervised machine learning* –which requires a set of predefined categories to be used as training data– and have used source code, comments, API Calls and other sources to obtain information about the projects to be categorized. We consider that existing approaches have weaknesses that can have major implications on the categorization results and haven't been solved at the same time, namely the assumption of non-restricted access to source code and the use of predefined sets of categories. Therefore, we present *Sally*: a novel, unsupervised and multi-label automatic categorization model that is able to obtain meaningful categories without depending on access to source code nor the existence of predefined categories by leveraging on information obtained from the projects in the categorization corpus and the dependency relations between them. We performed two experiments in which we compared *Sally* to the categorization strategies of two widely used websites and to MUDABlue, a categorization model proposed by Kawaguchi *et al.* that we consider to be a good baseline. Additionally, we assessed the proposed model by conducting a survey with 14 developers with a wide range of programming experience and developed a web application to make the proposed model available to potential users.

Keywords: closed-source, open-source, software categorization, machine learning.

Resumen

La categorización automática de repositorios de software ha ido ganando aceptación debido a que reduce el esfuerzo manual y puede generar resultados de alta calidad. La mayoría de los modelos existentes dependen fuertemente del *aprendizaje de máquina supervisado* – que necesita de un conjunto predefinido de categorías para ser usado como datos de entrenamiento– y han usado código fuente, comentarios, llamadas de API y otras fuentes para obtener información sobre los proyectos a categorizar. Consideramos que los modelos existentes tienen debilidades que pueden tener implicaciones importantes en el resultado de la categorización y no han sido resueltas al mismo tiempo, específicamente la suposición de que el código fuente de los proyectos se encuentra completamente disponible y la necesidad de conjuntos predefinidos de categorías. Por esto, presentamos el modelo *Sally*: Un enfoque de categorización automática de software novedoso, no supervisado y multi-etiqueta que es capaz de generar categorías descriptivas sin depender del acceso al código fuente ni a categorías predefinidas usando información obtenida de los proyectos a categorizar y las relaciones entre ellos. Realizamos dos experimentos en los que comparamos a *Sally* con las estrategias de categorización automática de dos herramientas online ampliamente utilizadas y con MUDABlue, un modelo de categorización automática de software propuesto por Kawaguchi *et al.* que consideramos una buena base de comparación. Adicionalmente, evaluamos el modelo propuesto por medio de un caso de estudio llevado a cabo con la participación de 14 desarrolladores con un amplio rango de experiencia en programación y desarrollamos una aplicación web para poner el modelo propuesto a disposición de usuarios potenciales.

Palabras clave: código propietario, código abierto, categorización de software, aprendizaje de máquina

Contents

Abstract	vii
Contents	ix
1. Introduction	2
1.1. Objectives	5
1.2. Contribution	5
1.3. Document structure	6
I. Background	7
2. Maven and the POM	8
3. Related Work	11
3.1. Software Categorization	11
3.2. Maven	13
II. The Sally software categorization approach	15
4. Description of the proposed categorization approach	16
4.1. Identifier extraction and filtering	17
4.2. Dependency resolution	20
4.3. Generation of Categories	21
4.3.1. Primary categories	22
4.3.2. Secondary categories	22
4.4. Concept definition	24
4.5. Tag Cloud Output	26
4.6. Sally: The application	26

III. Experimentation and analysis of results	30
5. Experimentation	31
5.1. Experiment 1: Comparison With Online Tools	31
5.1.1. Experiment setup	34
5.1.2. Results	34
5.1.3. Conclusion	40
5.2. Experiment 2: MUDABlue	40
5.2.1. Preliminary Results	42
5.2.2. User study	44
6. Conclusions	50
A. Appendix: Experiments corpus	52
References	57

List of Tables

4-1. Identifiers removed from project <i>htmlunit-core-js-2.15</i> after filtering.	19
4-2. Extracted categories for project <i>batik-css-1.7.jar</i>	23
5-1. Computed categories for project <i>jTransforms-2.4</i>	35
5-2. Computed categories for project <i>bcprov-jdk15</i>	35
5-3. Computed categories for project <i>stringtemplate-3.2</i>	36
5-4. Computed categories for project <i>annotations-3.0.0</i>	36
5-5. Computed categories for project <i>semargl-sesame-0.6.1</i>	37
5-6. Number of projects without categories per approach.	39
5-7. Excerpt of titles obtained for categories by MUDABlue	43
5-8. Excerpt of libraries belonging to cluster 205	43
A-1. Categorization corpus	56

List of Figures

1-1. An example of category propagation. Boxes contain the categories for each project.	4
4-1. Components of the proposed categorization model	17
4-2. A sample question on Stack Overflow	19
4-3. Dependency graph calculation	21
4-4. Output of <i>Sally</i> 's definition module for category <i>RGB</i>	25
4-5. Tag Cloud output for project <i>StringTemplate-3.2</i>	26
4-6. Home page	27
4-7. Project browsing view	28
4-8. Tag cloud output	28
4-9. Definition for tag <i>analysis</i>	29
5-1. The project categorization page from SourceForge for a newly created project.	32
5-2. The first search result for query "JUnit" on MVNRepository	33
5-3. <i>Summary of the distribution of categories given by MUDABlue to the libraries in the corpus. Figures 5-3a and 5-3b show how many libraries belong to each cluster (are labeled with each category). Figures 5-3c and 5-3d show how many categories were assigned to the libraries in the corpus.</i>	42
5-4. Sample question taken from the survey.	46
5-5. Summary of reponses to demographic questions.	47
5-6. Amount of evaluations per rating for both approaches	48

1. Introduction

Software reuse is the process of using existing software components to build software systems instead of creating them from scratch. Reuse has proven to be an important part of the software development lifecycle because its application on the planning and development of software projects can provide significant benefits to organizations on aspects such as cost, time, reliability, time needed for testing, etc.[12]

Currently it would be virtually impossible to build large-scale systems without reusing software to some extent; developers often use libraries such as `JUnit` or `TestNG` for testing, `JDBC` drivers for database connectivity, `Log4j` or `SLF4J` for logging, etc. These become dependencies for projects and given their importance for any software development process, tools as `Maven`¹ and `Gradle`² for handling these dependencies (and in general all aspects of the build process, even documentation) have been created.

Several open and closed-source software repositories are available for developers to take advantage of software components, which address particular needs their projects may have. However, in order to be able to take advantage of a repository, there should be ways for efficiently locating assets in it[11]. If for example, a user knows it is possible that a software repository contains useful assets but has no efficient way to access them, it may be preferable to implement the necessary functionality to avoid manually searching through the repository and the cost associated with this search.

Acknowledging this need for tools that help locating software projects in a repository, research has been made on the topic of *Software Categorization*. Categorization is the process of assigning one or more tags—which describe categories—to items in order to group them by their common properties (for example, books in a library are usually categorized by subject). It is usual for the set of categories to be defined before having knowledge of the items that are going to be categorized. However, research has been made in models that automatically create categories by leveraging on the properties of the set of items that are to be categorized.

¹<http://maven.apache.org/>

²<http://www.gradle.org/>

Software categorization is the process of assigning categories to software projects, libraries, binary files or in general any software asset. These categories could denote the application domain, programming language, hardware platform on which the software has to operate or any other feature that may be shared by groups of assets. The main objective of software categorization is to improve the browsing/searching process that a user of a repository has to make in order to find relevant assets. An effective categorization that improves browsing is particularly important since finding a software asset that perfectly matches a user's query is rare [21]. In [1], it is shown that developers often perceive awareness (the knowledge of the existence of a reusable asset) and acceptability (the asset must be acceptable to the developer for use in a new project) as impediments for software reuse. A proper categorization combined with search tools would address both aspects by allowing developers to determine if a particular functionality is available on projects belonging to the repository.

The categorization process could be done manually, for example by asking developers to categorize their projects upon uploading them to the repository. However, this could lead to misclassification and additionally one cannot assume that developers will always take the time to do it. The task could be left for repository owners or administrators, however, software repositories are enormous and keep growing at a very fast pace, which could make a manual categorization approach unpractical.

Automatic software categorization has progressively gained importance because of the benefits that derive from its use. Research has been made on various approaches and methods where the use of *Machine Learning* techniques for classification is common among the majority of them. Most of these approaches rely on the use of identifiers extracted from source code, assuming there is full access to the repository [13, 14, 38]. This situation is not common on environments different to those of repositories maintained by the Open Source community; it is known that many companies often work under high security practices to protect organizational secrets, which limits the access to source code thus giving no opportunity for a categorization model that relies on such sensitive information to work. Attending this concern about closed source repositories, approaches that rely not on the source code but on the Application Programming Interface (API) calls have arisen with positive results [19, 24].

Among the reviewed literature it is found that on most approaches, category labels are created manually by domain experts [5] or selected from a set of previously defined categories [24, 19]. These approaches rely on *supervised learning* [2], thus require a previously categorized set of projects to be used as training data. This assumes that the set of categories is sufficient to classify any new project that enters the repository although that may not necessarily be the case. Also, a predefined set of categories is limited by the knowledge of available domain experts or by the decisions made by repository administrators.

In some cases, category labels are created automatically by analyzing the information of hosted projects [13, 14, 36]. This automatic creation of category labels has the advantage of relying only on information mined from the corpus, thus decoupling category names from specific knowledge of the available domain experts. However, closed source repositories can not be categorized using these approaches since they rely on source code, which is obviously not available.

Although the use of API-calls for categorization (which solves the problem of closed-source repositories) has been used previously [24, 19] by analyzing calls to the Java SDK, information about external libraries has not been used as a source of information for the categorization process. This means that relations among projects in the categorization corpus were not taken into account as input for the categorization process. We hypothesize these relations could provide valuable information for determining the domain application of software projects.

Consequently, the main hypothesis of this thesis is that *application domains (categories) of a software project depend to some extent on those of the projects it depends on*. In order to test the hypothesis, an approach that makes use of the dependency relations between projects is proposed to propagate categories between projects.

Figure 1-1 shows a simplified example of the proposed approach. Projects x and w have no dependencies and have been labeled with categories *logging* and *database* respectively. Project v depends on w , therefore the category *database* is propagated to it. In the same fashion, categories from x and v are propagated to u , which has by itself the category *html*. This example propagates all categories and ignores the fact that the reuse of a certain library does not directly imply that the application domain of the library and its dependencies is the same; the details of the approach used for category propagation is described in chapter 4.

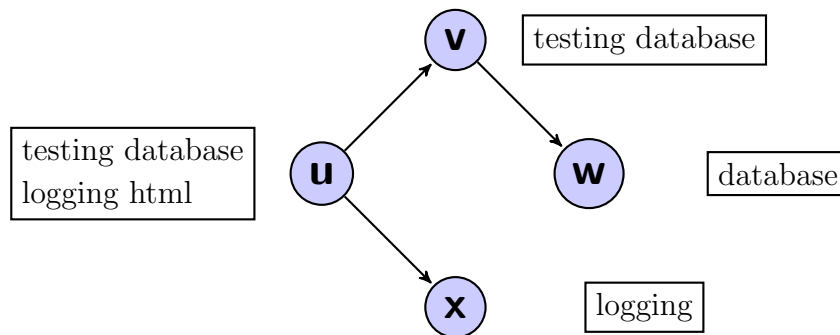


Figure 1-1.: An example of category propagation. Boxes contain the categories for each project.

1.1. Objectives

The main objective of this thesis is to design and implement an automatic categorization model that does not rely on access to the source code but rather the bytecode and harnesses the dependency relations between projects to assign categories. Also, the set of categories should not be predefined but generated from the projects that belong to the repository.

The specific goals and their contribution to the thesis project are the following:

- Goal 1:** To define and implement a procedure to construct dependency graphs from projects in the repository. In order to test the hypothesis, a procedure that allows to efficiently obtain information from the dependency relations is necessary.
- Goal 2:** To define and implement a categorization model based on the constructed dependency graphs and the information extracted from each particular library. By using information obtained from dependency graphs, the categories assigned to a particular project can be weighted and propagated to others that depend on it. Additionally, it is important to define the procedure under which projects that do not have dependencies will be categorized.
- Goal 3:** To compare the proposed model with a state of art approach and currently available alternatives. In order to assess the utility of the proposed model, a comparison against the alternatives is necessary.
- Goal 4:** To build an application that uses the proposed categorization model and harnesses the findings obtained from experimentation. The creation of an application will allow potential users to harness the results of this research on their own repositories.

1.2. Contribution

This thesis presents *Sally*, a novel, multi-label and unsupervised approach for categorization of Maven projects. By extracting identifiers from source code or bytecode and harnessing the dependency relations between projects, *Sally* is able to produce a set of weighted communicative tags and present them in a useful way for the user.

Because of the way *Sally* is designed, the approach is capable of dealing with common problems that previous work on the subject of automatic software categorization faced; some of the features of the approach and their importance are:

- **Does not depend on source code** : Unlike some of previously presented approaches [14, 36, 38], *Sally* does not need access to source code in order to work. This makes the approach a feasible alternative for closed-source and organizational repositories where access to information is restricted due to security reasons.
- **Does not need a predefined set of categories** : The use of predefined sets of categories for software categorization, although being successful in controlled environments [19, 24, 38], greatly limits the range of possible domain categories software projects can belong to. The use of these rigid sets of categories is likely to misclassify projects that do not fit into the predefined categories regardless of the used approach for their selection.
- **Provides meaningful labels**: Thanks to a category-filtering process based on tags from StackOverflow³, a large knowledge base related to software development, the labels obtained by *Sally* provide descriptive information to its users. Additionally, in order to deal with cases where users do not know the meaning of a particular category, *Sally* provides a simple way to obtain its definition from various popular information sources.

1.3. Document structure

The document is organized as follows:

Chapter 2: presents an introduction to Maven.

Chapter 3: presents related work on software categorization and empirical studies using Maven.

Chapter 4: describes the proposed approach in detail.

Chapter 5: presents two experiments that were performed to assess the proposed model.

Chapter 6: presents conclusions and future work.

³<http://stackoverflow.com>

Part I.

Background

2. Maven and the POM

*“Maven is a project management tool which encompasses a Project Object Model, a set of standards, a project lifecycle, a dependency management system and logic for executing plugin goals at defined phases in a lifecycle”*¹.

Maven is a tool mainly used with Java projects (although it can be used with other programming languages) that favors the concept of *convention over configuration* to help developers perform common tasks such as building, deploying, generating reports and even creating web sites by running simple commands. It mainly revolves around the concepts of the *build lifecycle* and the *Project Object Model*. A lifecycle is a clearly defined set of phases (or goals) that are performed to fulfill tasks related to the project; by default, Maven includes lifecycles for deploying, cleaning and generating sites. The *Project Object Model* is an XML file named *pom.xml* that fully describes Maven projects and can be used to customize Maven’s default behavior. It contains general information, build settings, environment and most importantly the declaration of dependencies to other projects or POM files. A minimal POM file should at least contain the `groupId`, `artifactId` and `version` attributes —known as the **GAV coordinates** — to allow Maven to uniquely identify the project.

One of the main reasons organizations and development teams choose to adopt Maven is for its dependency management scheme: new dependencies can be obtained by simply declaring them in the **dependencies** section of the POM file, each one uniquely described by its GAV coordinates. Also, when a developer builds a project, it is by default copied to the local repository and is ready to be reused by other projects. This scheme makes the development environment specially easy to replicate for new team members.

Besides from reusing projects available on local repositories, developers can reuse projects published in online repositories by simply adding the dependency declarations to the pom file. By default, dependencies that are not found in the local repository are searched for in *The Maven Central Repository (MCR)*². However, alternative online repositories can also be used.

¹Maven: The Complete Reference. <http://books.sonatype.com/mvnref-book/reference/>

²<http://search.maven.org/>

Listing 1 shows an abridged version of the POM file for project `spring-social-google`³ taken from the MCR. General information about the project as its GAV coordinate, developers, url, etc. is present. Also, it can be seen that dependencies to projects `javax.servlet-api`, `jackson-annotations` and others are declared. It is worth noticing that the declared dependencies also have a `scope` tag defined, this tag allows Maven to know whether the dependency should be packaged with the application, is only necessary for testing (`<scope>test</scope>`) or if the jar file should be searched for in the file system (`<scope>provided</scope>`) instead of the configured repositories.

By simply declaring GAV coordinates in its POM file, it is possible for Maven to obtain all the dependencies a project requires to function properly. However, since these dependencies are also Maven projects themselves, they can also have their own declared dependencies. All dependencies that are listed in the POM file are known as **direct dependencies** and the dependencies required by them are known as **transitive dependencies**. Maven seamlessly resolves transitive dependencies without any special intervention from the developer.

Maven by itself is a plugin execution framework. This is, most of the functionality provided by Maven is actually executed by plugins that can be retrieved from the central repository. This allows both to add functionalities to a Maven installation and to make upgrades without affecting the current ones. The **Maven Dependency Plugin** provides a set of goals related to the manipulation of artifacts and allows the developer to perform various tasks related to the dependencies of a project. These tasks include (but are not limited to) listing the set of resolved dependencies, finding unused ones, copying all dependencies to a specified location and listing all used repositories.

³<https://github.com/GabiAxel/spring-social-google>

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>org.springframework.social</groupId>
  <artifactId>spring-social-google</artifactId>
  <version>1.0.0.RELEASE</version>
  <name>Google API</name>
  <description>Google API</description>
  <url>https://github.com/GabiAxel/spring-social-google</url>
  <organization>...</organization>
  <licenses>...</licenses>
  <developers>...</developers>
  <scm>...</scm>
  <dependencies>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.0.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-annotations</artifactId>
      <version>2.3.3</version>
      <scope>compile</scope>
    </dependency>
    <dependency>...</dependency>
    <dependency>...</dependency>
    <dependency>...</dependency>
  </dependencies>
</project>
```

Listing 1: Abridged version of the POM file of project spring-social-google

3. Related Work

3.1. Software Categorization

Most of the previous work on the topic of software categorization has strongly relied on Machine Learning algorithms and mainly differ in the way features are extracted and the specific classification algorithm employed.

Source code [13, 14, 36], comments [36], README files [38], online repository profiles [40] and API calls [24, 19] have been used as input for feature extraction. Besides source code, Ugurel *et al.*[38] include comments and README files, finding that the use of comments can have a negative effect on the classification for some languages. Kawaguchi *et al.* [13, 14] extract identifiers only from source code arguing that the use of design documents, build scripts or other software artifacts is not convenient because although these artifacts can contain highly abstracted information, their quality can vary greatly from project to project thus affecting the categorization results. Later on, Tian *et al.* present in [36] a similar approach that also takes into account comments in source code and has a more strict filtering scheme for identifiers.

In [24, 19], the authors ignore source code and make use of API calls for feature extraction acknowledging the cases in which the availability of source code cannot be counted on. By using their approach, closed-source and organizational software repositories can be subject to automatic software categorization. It is known that several companies work under highly restricted environments to protect organizational secrets so a categorization approach that uses source code as input cannot work. In [40], Wang *et al.* use online repository profiles (descriptions and collaborative tags) from various sites as input as well as a method to aggregate them effectively.

For classification, most of the approaches treat software projects as documents and textual classification approaches are used to obtain categories. After extracting identifiers, Ugurel *et al.* [38] use Expected Entropy Loss to select the most important features, Support Vector Machines are used to categorize projects by programming language and application topic. In [13], the authors use Decision Trees for categorization with a reported error rate of less than 5%. To deal with the need of predefined categories required to use Decision Trees (and in general any supervised learning algorithm), the authors propose the use of LSA as a way to obtain similarities between software systems and to use this information for classification. Their work is extended in [14] where MUDABlue is proposed: after obtaining similarities between projects, MUDABlue uses cluster analysis to find sets of related software systems. Tian *et al.* propose in [36] a similar approach that uses LDA as a way to obtain topics and also applies cluster analysis to determine the software clusters. In [24], the authors use Decision Trees, Naive Bayes and Support Vector Machines (SVM) for classification, finding the SVM approach to be the most effective. Wang *et al.* also propose in [39] an approach based on similarity of software systems and clustering using these similarities to build a taxonomy for 40744 projects from Freecode¹.

Regarding topic detection and automatic labeling of software components, Wang *et al.*[41] use L-LDA [31] to solve the problem of software topic detection by mining software tags and profiles from three large open source repositories and compare their approach to LACT [36] and LDA-TR [15] with positive results. Kuhn proposes in [16], an approach based on log-likelihood ratio to automatically retrieve labels from source code and applies it to label the Java API and to compare different revisions of the JUnit software system. Later on, Kuhn *et al.* present in [17] an LSI-based semantic clustering approach to produce characterization of software systems which can help new developers to familiarize with unknown systems.

The work done by Bruno *et al.* [3] and Di-Lucca *et al.* [5] shows that not only software projects can be subject to categorization. In [3], the authors use an SVM-based approach to classify web services using identifiers extracted from WSDL files. Their approach produced good results generating a limited set of classes to which the services could belong. Di-Lucca *et al.* present in [5] a comparison of different Machine Learning algorithms to classify software maintenance requests.

¹<http://freecode.com>

3.2. Maven

Due to the widespread use of Maven by software developers, various aspects of Maven and the Maven Central Repository have been studied by researchers. These include trends of library usage [25], project versioning practices and their relation to backward compatibility [29], the relation between the popularity of components and their quality [34], bugs in the Maven build system [42] among others. Furthermore, the Maven Central Repository has served the purposes of multiple research projects by giving researchers a simple way to obtain projects to form a corpus. Migration graphs [35], software bertillonage² [4], the evaluation of a genetic algorithm to refactor interfaces using ISP principles [33] and an automatic bottom-up approach for generating software repositories [26] are some of the projects that the MCR has leveraged. Since it has proven to be a valuable resource for research on multiple subjects, a particular project was focused on providing a set of characteristics of the MCR to facilitate their use in future research [28].

Mileva *et al.* [25] analyzed 250 open source projects from the Apache Software Foundation³ to find trends of library usage. Dependencies were collected over a period of two years to find interesting behaviors of development teams regarding the release of new versions and an Eclipse⁴ plugin as well as a web tool called AKTARI were presented as a result of their findings.

Raemaekers *et al.* mined in [29] 150000 binary jar files to analyze various aspects related to the use of semantic versioning practices and their implications regarding backward compatibility. It was found that currently used mechanisms to indicate interface instability are not applied properly, which increases the amount of work necessary to upgrade project dependencies.

Sajnani *et al.* analyzed in [34] if there was a relation between the popularity of Maven components and their quality. Analyzing bug patterns and various software quality metrics, they found no evidence to support the commonly believed claim that states that popular components tend to have greater quality and less bugs than non popular ones.

Teyton *et al.* [35] mined a set of approximately 39000 projects from the MCR to extract migration graphs and produce visual patterns that can help developers find feasible options whenever a particular library must be replaced. By using these graphs, developers can quickly identify candidate libraries to migrate to and see how many projects have migrated from them.

²Bertillonage was originally a system based on anthropometric measurements for the identification of criminals invented by Alphonse Bertillon.

³<http://www.apache.org/>

⁴<http://eclipse.org/>

Davies *et al.* made use of the MCR on [4] to evaluate a *Software Bertillonage* approach for narrowing the search space for determining the provenance of Java software assets.

Romano *et al.* used in [33] information of public API usage from the Maven Central Repository to evaluate a genetic algorithm to refactor interfaces by applying the Interface Segregation Principle. The algorithm was shown to outperform other search based approaches.

Ossher *et al.* proposed in [26] a bottom-up approach for automatically constructing software repositories and performed a comparison between the structure of a generated repository with the structure of the Maven Central Repository. They found that although the Maven Central Repository is manually maintained and curated, the proposed approach generated a competitive and in some aspects superior structure.

Xia *et al.* performed in [42] an empirical study to determine various aspects of bugs in software build systems including Maven. Bug densities, categories and severities were analyzed.

In [28], Raemaekers *et al.* present the Maven Dependency Dataset, a set of code metrics, dependencies, breaking changes between library versions and a call-graph for the repository with the intention of providing easy access to the Maven Central Repository for research.

Part II.

**The Sally software categorization
approach**

4. Description of the proposed categorization approach

On this chapter we formally present *Sally*, a novel, multi-label and unsupervised approach for automatic categorization of Java Maven projects. *Sally* is capable of producing categories for projects in a software repository by analyzing identifiers and information about the dependency relations extracted from bytecode.

The main features of the proposed approach are the following:

- **Does not depend on source code** : Since *Sally* uses information extracted from bytecode, it is possible to use it to categorize closed-source and organizational Maven repositories.
- **Does not need a predefined set of categories** : *Sally* is capable of generating categories without requiring any special action from developers, this allows the model to work on repositories that do not have a defined categorization scheme and on projects that do not provide a description of their purpose.
- **Provides meaningful labels**: Thanks to a category-filtering process based on tags from StackOverflow¹, *Sally* is able to provide descriptive information about the projects in a repository. Additionally, by mining information from widely used sources, definitions for the presented labels can be obtained.
- **Provides information beyond category names**: Besides generating categories based on identifiers and dependency relations, *Sally* is able to produce a measure of how relevant is a category for a project.

¹<http://stackoverflow.com>

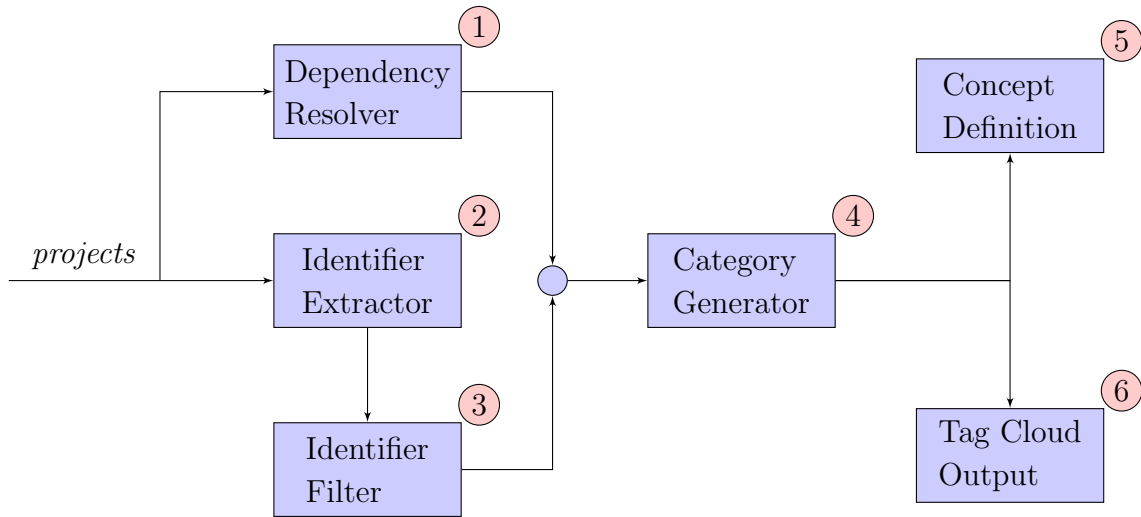


Figure 4-1.: Components of the proposed categorization model

Figure 4-1 shows a diagram of the different phases of the *Sally Categorization Model*. A resolved set of Maven projects² is used as input for the **Dependency Resolver** (1) and **Identifier Extractor** (2) modules. Then, extracted identifiers are filtered using the **Identifier Filter** (3) module. The information obtained from these three modules is used by the **Category Generator** (4) to be able to select a set of identifiers that become categories. This module is also in charge of harnessing the dependency information obtained from the Dependency Resolver to propagate categories. The output of the model is a set of categories whose context is augmented by the **Concept Definition** (5) module and presented to the users using a **Tag Cloud** (6). The depicted phases of the process are explained in the remainder of this section:

4.1. Identifier extraction and filtering

Extraction

The use of identifiers extracted from software for automatic categorization is common among various approaches [5, 36, 38, 14, 24]. This is because identifiers extracted from internal documentation of software such as comments, variable names and class names can clearly reflect human concepts [7]. Source code, comments, bytecode, README files, etc. have been successfully used for feature extraction in the past, therefore, we also make use of this approach in *Sally*.

²We refer to a resolved set of projects as a set in which all dependencies of the members of the set also belong to the set.

The **Identifier Extractor** module makes use of the ASM Bytecode Manipulation Framework ³ to obtain class names, class fields, method names and method arguments from bytecode. Then, by using Apache Lucene ⁴, the obtained identifiers are stemmed to prepare them for future computations. The output of this module is a set of $(root, term)$ pairs, where *root* is the stemmed identifier and *term* is the original identifier. For example, the word `stemming` would produce the pair $(stem, stemming)$ after going through the extraction process.

Filtering

The identifiers obtained as output from the Identifier Extraction Module are filtered in the **Identifier Filter Module**. This module is in charge of retaining only the identifiers that fulfill certain conditions in order to produce a set that is useful for computations in further steps and that also provides information that is descriptive for humans.

The first step of the filtering process removes identifiers with less than 3 characters, identifiers that appear on more than 50% of the projects and treats all Java keywords as stopwords. The lower bound on the number of characters for each identifier was chosen because in software development there are various widely known acronyms and abbreviations that describe high level concepts and are composed of three characters, e.g., XML, POM, RSS, etc. Removing these kind of identifiers could also remove a large amount of useful information for the categorization model.

The second step is to filter the identifiers by using tags found on StackOverflow ⁵, a popular question and answer site where programmers with a wide range of experience help each other by asking and answering questions. With a community of over 3.5 million users and more than 8.4 million questions, the site has become a valuable resource for programmers as well as an important knowledge base for research [37]. We consider this filtering necessary because the set of identifiers obtained from bytecode can be polluted with names and instructions that do not convey any useful information for humans.

Figure 4-2 shows a question taken from the StackOverflow site. It contains a title (1), a description (2), a set of tags (3), zero or more answers (4) and votes (5) that are given by the community to rate the importance and quality of questions and answers. All content-related aspects from the site are maintained and filtered by the community, which gives a high level of quality to the information present on the site.

³<http://asm.ow2.org/>

⁴<http://lucene.apache.org>

⁵<http://stackoverflow.com>



Figure 4-2.: A sample question on Stack Overflow

The filtering process works by comparing the obtained identifiers to the tags available in StackOverflow and removing the ones that do not appear in this set. Since these tags are maintained and filtered by the community to ensure that they provide useful information about the questions to which they are assigned, they represent concepts related to Software Development, Computer Science, Programming and General Tasks. This makes the set of StackOverflow tags ⁶ a diverse and curated list of categories, concepts and terms which we consider to be valid descriptors for the software projects to be categorized. In the same way a tag on a question in StackOverflow can give a reader an idea about the subject of the question, we expect it to give information about the project it is assigned to.

To illustrate the benefits of filtering using the StackOverflow set of tags, Table 4-1 shows 25 (out of 442 total) identifiers that were filtered out by applying the described process to identifiers extracted from project *htmlunit-core-js-2.15*.

⁶<http://stackoverflow.com/tags>

prologue	tza	fsub	getelem	bitnot
significand	fneg	wide	nclass	dtostr
iproxy	impdep	dstore	cpbegin	aastore
arrayobj	pos	withexpr	frac	actual
arraylength	getprop	iushr	descendants	dude

Table 4-1.: Identifiers removed from project *htmlunit-core-js-2.15* after filtering.

The description for project *HtmlUnit*⁷ is the following: “*HtmlUnit is a GUI-Less browser for Java programs. It models HTML documents and provides an API that allows you to invoke pages, fill out forms, click links, etc... just like you do in your normal browser. It has fairly good JavaScript support (which is constantly improving) and is able to work even with quite complex AJAX libraries, simulating either Firefox or Internet Explorer depending on the configuration you want to use. It is typically used for testing purposes or to retrieve information from web sites.*”

It can be seen from table 4-1 that the removed terms do not provide much information on what the purpose of the library is, they instead might confuse an observer and make the task of understanding it significantly more difficult.

4.2. Dependency resolution

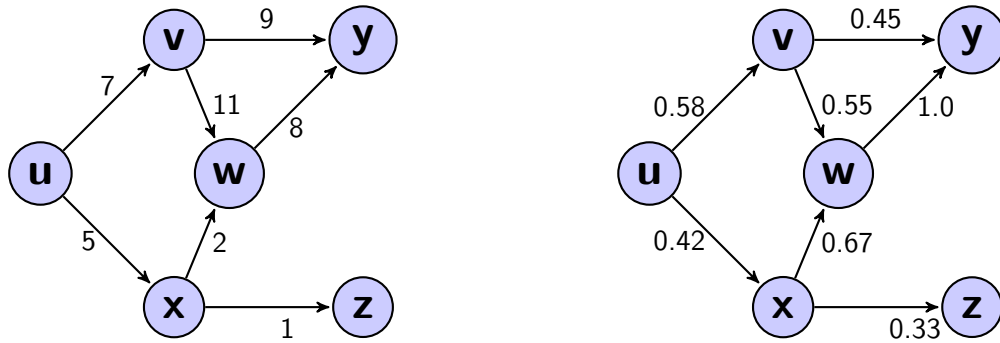
Software reuse can be seen as the inclusion/adaptation of previously developed software artifacts into new projects with the intention of using their functionality; this inclusion creates a dependency relation between the projects. More formally, it is said that a project u depends on another project v when there is at least one method invocation, object instantiation or inheritance relation from classes in u to classes in v . We will refer to each of these situations that create dependency relations as *dependency calls*.

The use of graphs for modeling various kinds of relations between software projects or components is a commonly used technique for extracting information from them [23, 20, 27]. Given a set of projects in a repository, a dependency graph can be obtained by modeling projects as nodes and dependency relations as directed edges. Each edge can optionally be weighted by the number of dependency calls between the pair of nodes it connects.

On the **Dependency Resolver Module**, a dependency graph whose edges are weighted by the number of dependency calls is obtained from the projects to be categorized. This graph is used to compute a *dependency measure* defined as:

$$D_{uv} = \frac{dc(u, v)}{\sum_{i=1}^{i=n} dc(u, i)}$$

⁷<http://htmlunit.sourceforge.net/>



(a) Edges represent the number of dependency calls between connected nodes (projects). (b) Edges represent the computed measure between connected nodes (projects).

Figure 4-3.: Dependency graph calculation

Where $dc(x, y)$ is the number of dependency calls made from project x to project y and n is the number of projects on which x depends on. Using this measure we can define the dependency between projects as a tuple $(source, destination, weight)$ where $source$ and $destination$ are both projects and $weight$ is the dependency measure between them.

Figure 4-3a shows a simple dependency graph where the weight of each edge represents the number of dependency calls between pairs of projects. In this graph, project u has a total of 12 outgoing dependency calls, 5 to x and 7 to v . Figure 4-3b depicts the same graph with its edges displaying the computed dependency measure for each pair of connected nodes. The dependency measure between u and x is then $5/12 = 0.42$ (the number of outgoing dependency calls from u to x divided by the total number of dependency calls made by u).

4.3. Generation of Categories

For each project, a set of categories is found and assigned to it. A category is defined as a tuple $(name, relevance)$ where $name$ is the name of the category and $relevance$ is a value that describes how relevant is the category for the particular project. The relevance measure is subject to

$$0 \leq r_i \leq 1.0 \wedge \sum r_i = 1.0$$

where r_i is the relevance value of category i . For example, if we were examining the project `JUnit` we would expect to have some categories whose names are related to testing and have high relevance measures. Moreover, the **Category Generator Module** divides categories into primary and secondary categories, both types are described below:

4.3.1. Primary categories

Primary categories arise from the analysis of a particular project by itself, i.e., only taking into account the identifiers extracted from bytecode and ignoring dependencies. To obtain the primary categories, **Gensim** [32] is used to compute TF-IDF [22, 30] values for the identifiers obtained from the Identifier Filter Module described in section 4.1.

TF-IDF is a term weighting scheme that combines the concepts of *term frequency (TF)* and *inverse document frequency (IDF)* to find how important is a term for a document in a corpus while reducing the effect of terms that occur very often across documents; it is calculated as follows:

$$tf_{t,d} \times \log \frac{N}{df_t}$$

where $tf_{t,d}$ is the number of occurrences of term t in document d (*term frequency*) and df_t refers to the number of documents in the corpus that contain term t (*document frequency*).

The computed values are used to obtain *relevance measures* for each one of the terms, calculated as:

$$\frac{tfidf(t_i)}{\sum_{k=1}^m tfidf(t_k)}$$

where $tfidf(t_i)$ is the TF-IDF value for term i and m is the number of identifiers obtained from the project. Finally, identifiers are sorted in descending order by relevance and the roots (recall that the Identifier Filter Module outputs a set of $(term, root)$ pairs) of the identifiers become the primary categories.

4.3.2. Secondary categories

The secondary categories of a project are the primary categories of its direct dependencies⁸ with their relevance measures scaled using the *dependency measure* that was computed by the Dependency Resolver Module. Once again, categories are sorted in descending order by their relevance values.

If we define each project p to be a 3-tuple (D_p, C_p, S_p) where D_p is the set of projects on which p depends on, C_p is the set of primary categories of p , and S_p its set of secondary categories; the category propagation algorithm can be described by the following pseudo-code:

⁸A deeper exploration of the dependency graph (i.e. using transitive dependencies) was tried but the results were not satisfactory for most of the projects.

Data: A project $p = (D_p, C_p, S_p)$

Result: Project p with its primary and secondary categories fully resolved.

```

1 begin
2   for  $d \in D_p$  do
3     for  $c \in C_d$  do
4       | add new category ( $c.name, c.relevance * d.weight$ ) to  $S_p$  ;
5     end
6   end
7 end

```

Listing 2: Category propagation algorithm

To illustrate the output of the **Category Generator Module**, Table 4-2 shows the top 5 primary and secondary categories (as well as their relevance measure) obtained for project *batik-css-1.7.jar*.

Primary	Secondary
rgb - 0.5123	svg - 0.5585
color - 0.1494	css - 0.2847
style - 0.1304	smil - 0.0925
cssom - 0.1130	tag - 0.0349
selector - 0.0949	feature - 0.0293

Table 4-2.: Extracted categories for project *batik-css-1.7.jar*

The following is the description for project *batik-css-1.7.jar*: “*Batik CSS Engine. Batik is a Java-based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG) format for various purposes, such as display, generation or manipulation*”.

Project *batik-css-1.7.jar* depends on project *batik-util-1.7.jar* from where the secondary categories are extracted. It can be seen that both projects are related to graphics. Moreover it can be seen by looking at their relevance measures, that the secondary categories *svg* and *css* are significantly more important than the other three, which helps support the conclusion that the project’s domain application is related to graphics.

The number of categories that are obtained for projects can be customized either by the number of categories desired or by establishing a relevance threshold, e.g. only obtain categories with a relevance measure greater than 15%.

4.4. Concept definition

Depending on the background and programming experience of users of *Sally*, it is possible to perceive that the generated categories do not provide useful information because the concepts are unknown. To deal with this possibility, a **Concept Definition Module** was developed; given a term, it is capable of searching for its definition on various information sources to provide detailed information about it.

For example, identifiers `tiff`, `glyph` and `rgb` were extracted by *Sally* as categories for project *xmlgraphics-commons-1.3.1.jar*. Given these identifiers, it is possible for a user without prior knowledge about graphics to perceive that the information provided by the model is not useful enough. Since each of these identifiers are related to concepts, we can provide the user with a better understanding of the identifiers by defining what these concepts mean.

The following are excerpts from the definitions obtained by the module for the aforementioned concepts:

tiff: *TIFF (Tag Image File Format) is a common format for exchanging raster graphics (bitmap) images between application programs, including those used for scanner images. A TIFF file can be identified as a file with a “.tiff” or “.tif” file name suffix.*

rgb: *RGB (red, green, and blue) refers to a system for representing the colors to be used on a computer display. Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum. Levels of R, G, and B can each range from 0 to 100 percent of full intensity.*

glyph: *In information technology, a glyph (pronounced GLIHF ; from a Greek word meaning carving) is a graphic symbol that provides the appearance or form for a character. A glyph can be an alphabetic or numeric font or some other symbol that pictures an encoded character. The following is from a document written as background for the Unicode character set standard. In the Unicode standard, a character is stated to be an abstract entity and not a glyph (some visual representation of a character).*

Since the most relevant identifiers become categories for projects, we expect users to find the assigned categories more useful by giving them detailed information on what the concepts behind these categories mean. Currently, *StackOverflow*⁹, *Wikipedia*¹⁰, *Wiktionary*¹¹ and *TechTarget*¹² are the supported sources for concept definition.

⁹<http://stackoverflow.org>

¹⁰<http://www.wikipedia.org/>

¹¹<https://www.wiktionary.org/>

¹²<http://whatis.techtarget.com>

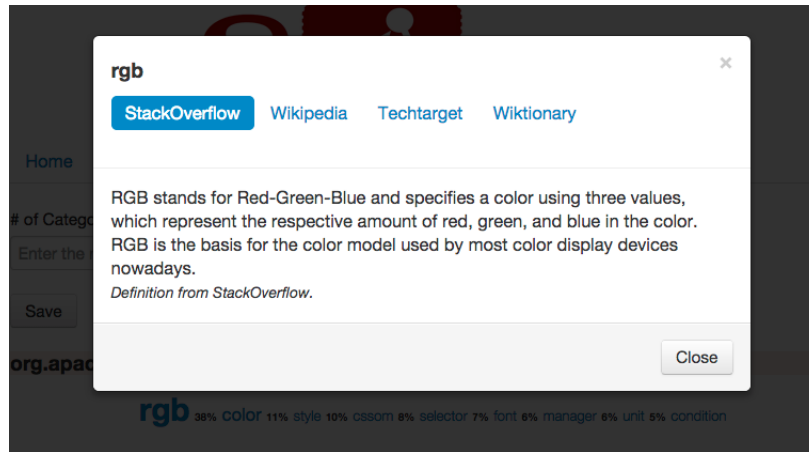


Figure 4-4.: Output of *Sally*'s definition module for category *RGB*

To obtain definitions from the mentioned sites, the *JSoup HTML Parser*¹³ was used. We manually obtained the html anchors that are used on each of the sites to visually present results from queries. For example, *TechTarget* uses “*articleBody*” as the `id` attribute of the paragraph that contains the text of a definition. Using *JSoup* we can find these elements by `id` and extract their contents. Figure 4-4 shows the output of the Concept Definition Module for category *RGB*.

Having an automatic way to obtain these definitions aids users without prior knowledge about some particular application domains to get a better understanding of them. However, the main reason for developing the definition module is that although the selected identifiers are related to the application domains of projects, they are not necessarily enough to describe them; this is because identifiers present low-level information about projects, e.g. frameworks, communication protocols, related programming languages, technologies, etc. In order for a user to find the concepts that describe what a project's application domain is, it is necessary to go beyond this specific details that identifiers by themselves provide and relate them to form more general concepts. We expect definitions to be the bridge between identifiers and the complex concepts that serve to describe the application domains of software projects.

¹³<http://jsoup.org/>

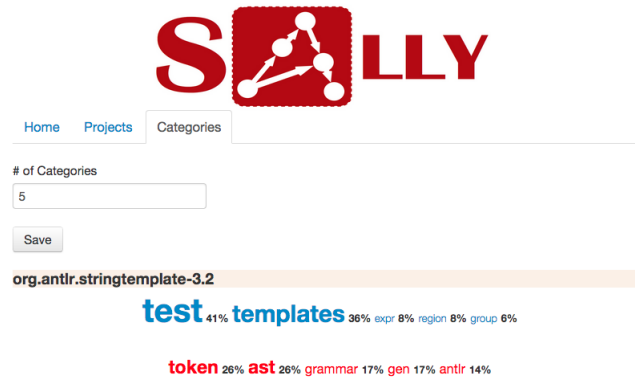


Figure 4-5.: Tag Cloud output for project *StringTemplate-3.2*

4.5. Tag Cloud Output

In order to visually present the obtained information in a useful way, Sally’s user interface presents the extracted categories and their relevance measures as a **tag cloud**, where the size of each tag is directly related to the relevance of the category for the project that is being visualized. The intention behind this decision is to provide visual aid to users in finding the most relevant categories for a particular project, i.e, although all extracted categories are related to a project in one way or another, there are some of them that are more relevant than the others.

As an example, figure 4-5 depicts the generated tag cloud for project *Stringtemplate-3.2*. The description obtained from the official website¹⁴ is the following: “*StringTemplate is a java template engine for generating source code, web pages, emails, or any other formatted text output. StringTemplate is particularly good at code generators, multiple site skins, and internationalization localization. StringTemplate also powers ANTLR.*”

4.6. Sally: The application

Sally is composed by two main components, one is responsible for core functionality and the other is in charge of making the information obtained by the core component available to users. All modules in charge of information extraction (i.e. identifier extraction / filtering, dependency resolution, category generation, definition mining) were developed as a Java Maven project that makes use of Python and Bash scripts.

¹⁴<http://www.stringtemplate.org/>

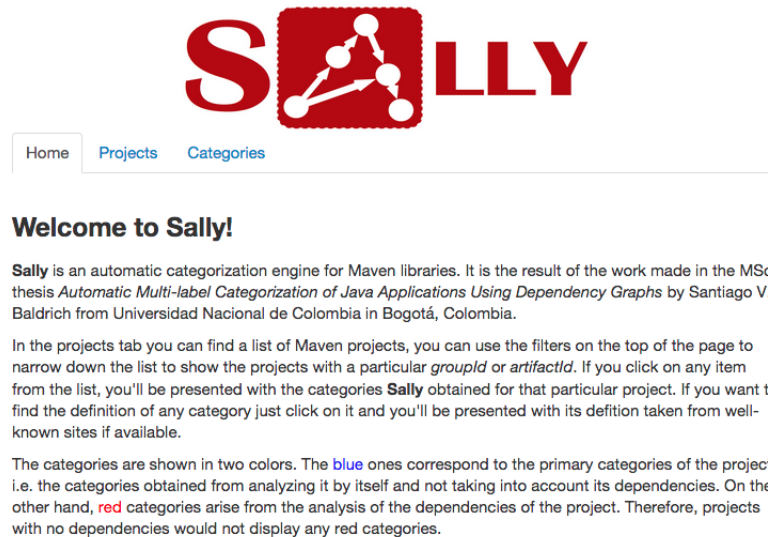


Figure 4-6.: Home page

The web application was developed using the Meteor¹⁵ Javascript platform, which allowed for a very fast development process and all information is stored in a MongoDB¹⁶ database. *Sally* is currently hosted at <http://sally.meteor.com>.


As Figure 4-6 shows, users are greeted with a description of *Sally* and an explanation of how to use the site and the meaning of the colors of categories on the tag cloud.

On the project browsing view shown in Figure 4-7, users can use filters to reduce the number of projects that are displayed to help them find the one they are looking for more easily.

When a project is selected, the user is presented with the tag cloud for it as shown in Figure 4-8. The number of displayed tags can be increased or decreased by inputting a number on the upper box. By clicking on a tag, the mined definition is presented as a modal dialog. As an example, the definition for tag *analysis* is presented on Figure 4-9.

¹⁵<https://www.meteor.com/>

¹⁶<https://www.mongodb.org/>



Home Projects **Categories**

GroupId ArtifactId

GroupId	ArtifactId	Version
com.google.code.findbugs	annotations	3.0.0
com.google.code.findbugs	bcel-findbugs	6.0
com.google.code.findbugs	findbugs	3.0.0
com.google.code.findbugs	jFormatString	3.0.0
com.google.code.findbugs	jsr305	3.0.0

Figure 4-7.: Project browsing view

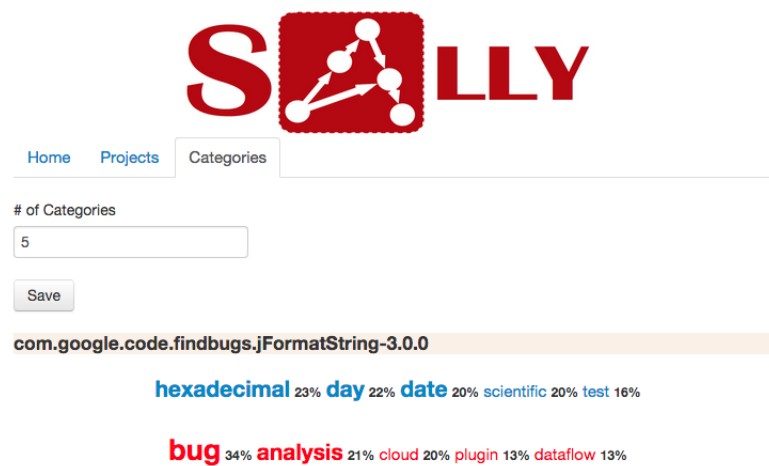
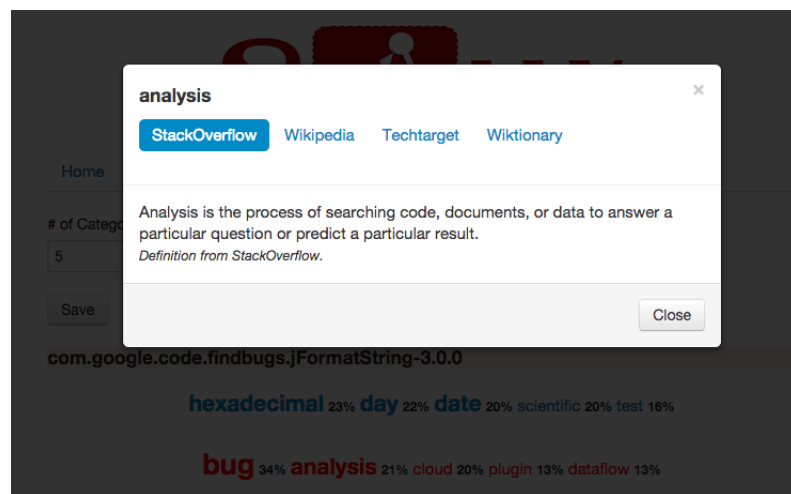


Figure 4-8.: Tag cloud output

Figure 4-9.: Definition for tag *analysis*

Part III.

Experimentation and analysis of results

5. Experimentation

In order to assess the quality of the proposed model, two different experiments were performed. The first one was a comparison against the categorization schemes of two widely used tools: **SourceForge**¹, an online repository where developers can share their software projects and **MVNRepository**², a search engine that allows users to look for characteristics (mainly the GAV coordinate) of Maven projects. The second experiment consisted on implementing **MUDABlue**[14], an unsupervised multi-label categorization approach for software projects developed by Kawaguchi *et al.* and comparing it to *Sally*.

The set of projects for both experiments was obtained by using MVNRepository to find the GAV coordinates of 68 Maven projects with the `net.sourceforge` groupId. These projects were declared as dependencies on the pom file of an empty project and the **Maven dependency plugin** (described in Chapter 2) was used to resolve all transitive dependencies. This selection process produced a set of 167 jar files and the rationale behind it was to obtain a corpus composed by projects that were present on both MVNRepository and SourceForge, thus making a comparison possible. **Appendix A** shows the complete set of projects that were used for the experiments.

5.1. Experiment 1: Comparison With Online Tools

Currently, there are multiple online software repositories for developers to publish their own projects or to search for previously developed ones for reuse. SourceForge, Google Code³, Github⁴ and IBiblio⁵ are just some of many available alternatives. SourceForge has been used in previous occasions either as a baseline for comparison or as the categorization corpus in various research projects [24, 9, 40, 14, 19, 36]. It is considered for comparison with *Sally* as well.

¹<http://sourceforge.net>

²<http://mvnrepository.com>

³<http://code.google.com>

⁴<http://github.com>

⁵<http://www.ibiblio.org/>

Regarding Maven libraries, alternatives are not as numerous as they are for self-contained applications. Although there exist some available online Maven repositories (with the Maven Central Repository being by far the most prominent one), there are not many ways to search for particular projects by purpose nor domain application. **MVNRepository**, a site that indexes projects belonging to the Maven Central Repository and presents information about them by using their pom file is one of the most widely used alternatives.

The categorization schemes used by SourceForge and MVNRepository are described below:

SourceForge

SourceForge is an open source project hosting site with over 3.7 million registered users and is one of the most widely used alternatives by developers for publishing software projects. After registering a project, the developer can optionally add tags and select topics that describe its characteristics. The set of topics is built as a 4-level hierarchy with 21 general topics on the top that are specialized to form a set of 754 possible topics.

Figure 5-1 shows the screen that is presented to a developer for categorizing a new project. The developer can add any number of free text tags (1) to the project and select up to three categories from the predefined set of topics (2).

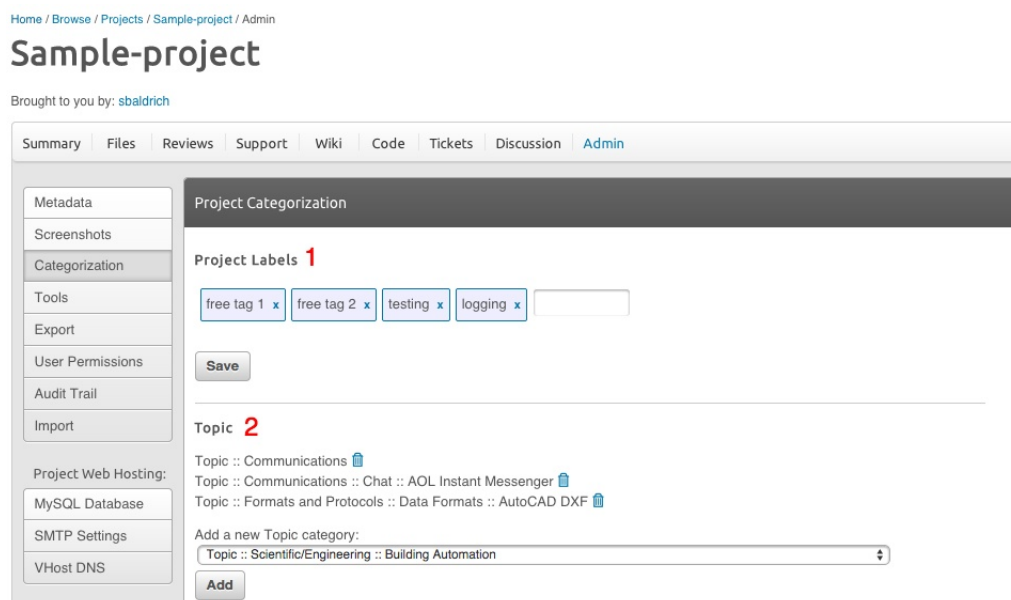


Figure 5-1.: The project categorization page from SourceForge for a newly created project.

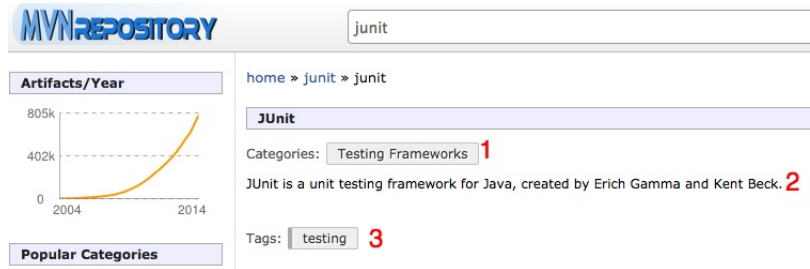


Figure 5-2.: The first search result for query “JUnit” on MVNRepository

MVNRepository

MVNRepository is a search tool for Maven projects. It helps its users find the GAV coordinates of a project by only providing its `groupId` or `artifactId`. Additionally, it shows the content of the project’s `description` tag –if available– and displays it on the website. The categorization scheme from MVNRepository is divided into tags and categories, this means projects are labeled with both. Tags are extracted from the text in the pom file and we believe categories are assigned manually⁶. To differentiate categories from tags in the following section, the same convention from MVNRepository is used: categories are presented with initial capital letters and tags are shown in lower case.

Figure 5-2 is an excerpt of what a user of MVNRepository sees after performing a query with the term *junit*. The user is presented with the assigned categories (1), the content of the `description` tag from the pom file (2) and with the tags (3) assigned to the project.

⁶We tried multiple times to contact the creator of the search tool in order to fully understand the scheme used for classification and tagging but we never received any reply from him.

5.1.1. Experiment setup

Our goal was to compare the categories generated by *Sally* to those from SourceForge and MVNRepository in terms of their availability (whether the approaches have categories for all of the projects in the corpus) and how descriptive they are. Therefore, the following research questions were addressed in this experiment:

- *RQ₁: How do the categories assigned by developers in SourceForge compare to the ones generated by Sally in terms of their descriptiveness?*
- *RQ₂: How do the categories generated by MVNRepository compare to the ones generated by Sally in terms of their descriptiveness?*
- *RQ₃: How do Sally, SourceForge and MVNRepository compare in terms of the availability of categories?*

In order to answer these questions, categories and tags were manually obtained from the SourceForge and MVNRepository websites and *Sally* was used to generate 5 primary and 5 secondary categories for each project. A manual examination was then conducted to compare the categories assigned by each approach.

5.1.2. Results

In this section, an excerpt of the results obtained in the experiment is presented, the full table with results is available at <http://bit.ly/categresults>. For each of the following tables, categories assigned by *Sally* (primary and secondary) as well as the relevance measure described in section 4.3 are shown on the first and second columns, the third column contains tags and categories assigned by MVNRepository and the last one contains the categories from SourceForge. After each table, the description for the analyzed project is shown.

1. Results for project `net.sourceforge.jtransforms:jtransforms-2.4.jar`

Description:⁷ *JTransforms is the first, open source, multithreaded FFT library written in pure Java. Currently, four types of transforms are available: Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Discrete Sine Transform (DST) and Discrete Hartley Transform (DHT). The code is derived from General Purpose FFT Package written by Takuya Ooura and from Java FFTPack written by Baoshe Zhang.*

⁷<https://sites.google.com/site/piotrwendykier/software/jtransforms>

<i>Sally</i>		MVNRepository	SourceForge
fft - 0.4968	None	None	Mathematics
dct - 0.1857			
dht - 0.1701			
dst - 0.1010			
benchmark - 0.0464			

Table 5-1.: Computed categories for project jTransforms-2.4

In Table 5-1 it can be seen that MVNRepository has no categories for project *jtransforms* and that although SourceForge does assign the category *Mathematics* to it, only *Sally* provides specific categories for the project at hand. Although the first four categories assigned to the project by our approach are precise and specific, the last one (*benchmark*) does not seem to be as useful as the other four. This less relative importance can also be seen in the relevance value assigned to it.

2. Results for project `bouncycastle:bcprov-jdk15.jar`

<i>Sally</i>		MVNRepository	SourceForge
jce - 0.3482	test - 0.4469	Encryption Libraries	Missing
certificate - 0.2066	suite - 0.1794		
cipher - 0.1588	runner - 0.1794		
revocation - 0.1442	failures - 0.1315		
rsa - 0.1422	defect - 0.0628		

Table 5-2.: Computed categories for project bcprov-jdk15

Description:⁸ *The Bouncy Castle Java API for handling the OpenPGP protocol. This jar contains the OpenPGP API for JDK 1.5 to JDK 1.8. The APIs can be used in conjunction with a JCE/JCA provider such as the one provided with the Bouncy Castle Cryptography APIs.*

Table 5-2 shows the categories assigned by the three approaches to project *bcprov-jdk15*, which is basically a cryptography API. All primary categories are closely related to the project at hand but the secondary ones are not. This is the result of considering all the declared dependencies from the pom file of the projects instead of those only under the `compile` scope.

⁸<https://repo1.maven.org/maven2/org/bouncycastle/bcpg-jdk15on/1.51/bcpg-jdk15on-1.51.pom>

3. Results for project `org.antlr.stringtemplate-3.2.jar`

<i>Sally</i>		MVNRepository	SourceForge
test - 0.4125	token - 0.2660	Template Engines	Missing
templates - 0.3639	ast - 0.2638		
expr - 0.083	grammar - 0.1721		
region - 0.081	gen - 0.1659		
group - 0.057	rule - 0.1319		

Table 5-3.: Computed categories for project `stringtemplate-3.2`

Description:⁹ *StringTemplate is a java template engine for generating source code, web pages, emails, or any other formatted text output. StringTemplate is particularly good at code generators, multiple site skins, and internationalization / localization. StringTemplate also powers ANTLR.*

Table 5-3 shows how both primary and secondary categories found by *Sally* contain terms associated to regular expressions and grammars, which directly relate to the application domain of the project under analysis. MVNRepository has a relevant category as well, however it does not present any automatically generated tags. The project is not available in SourceForge.

4. Results for project `com.googlecode.findbugs:annotations-3.0.0.jar`

<i>Sally</i>		MVNRepository	SourceForge
confidence - 0.4905	None	analysis	Software Development
desired - 0.1426		annotations	
annotation - 0.1280		Defect Detection Metadata	
priority - 0.1197			
warning - 0.1192			

Table 5-4.: Computed categories for project `annotations-3.0.0`

Description:¹⁰ *Annotation support for the FindBugs too, a program which uses static analysis to look for bugs in Java code.*

⁹<http://www.stringtemplate.org/index.html>

¹⁰<http://findbugs.sourceforge.net/index.html>

Table 5-4 shows that none the approaches is successful at describing the application domain of the project at hand. *Sally* produces a set of categories that despite being related to the relevant application domain, are too specific and fail to provide the necessary context to understand it. This situation arises because of the way the dependency relations in the graph are analyzed and used as input for the approach (e.g. only outgoing connections in the graph are taken into account). MVNRepository also shows a similar situation with its tags although the terms are better at describing the domain of the project than the ones presented by *Sally*. SourceForge on the other hand, presents a very wide category which does not provide useful information to describe the project.

5. Results for project `org.openrdf.sesame:sesame-rio-rdfjson-2.7.12.jar`

<i>Sally</i>		MVNRepository	SourceForge
rdf - 0.4558	rdf - 0.3630	json	Storage
json - 0.3234	datatype - 0.2123		Libraries
graph - 0.0780	owl - 0.1915		Semantic Web
literal - 0.073	statement - 0.1184		(RDF, OWL, etc.)
statement - 0.069	axiom - 0.1145		

Table 5-5.: Computed categories for project `semargl-sesame-0.6.1`

Description:¹¹ *Rio parser and writer implementation for the RDF/JSON file format included in the Sesame Framework, a powerful Java framework for processing and handling RDF data. This includes creating, parsing, storing, inferencing and querying over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions.*

Table 5-5 shows that the automatic tagging approach from MVNRepository does not extract all possible information for the analyzed project and is outperformed by both *Sally* and SourceForge. It is important to recall that although both SourceForge and *Sally* produced similar results, the categories presented by SourceForge were manually assigned, while the ones presented by *Sally* were automatically obtained without any interaction from developers.

¹¹<http://rdf4j.org/>

RQ₁: SourceForge

Categories in SourceForge are selected manually by developers, which allows them to have some control over the indexing of their projects on the site. However, this allows for ambiguous categorizations and the assignation of categories that do not correspond to the real purpose or domain application of projects. Also, since the set of categories is predefined, there are categories in the site that are too broad to give useful information, (e.g. Software Development, Framework, Libraries, etc.). It is important to keep in mind that SourceForge is not commonly used to host projects as a set of libraries that depend on each other as the Maven Central repository is, it is rather oriented towards hosting standalone applications or self-contained libraries. This evidenced two particular consequences: First, that even though all projects declared as dependencies in the pom file that was used to form the corpus belonged to the `net.sourceforge` groupId, a large amount (37%) of transitive dependencies were not available in SourceForge. This subject is mentioned in depth in the discussion of research question 3. The second consequence is that since Maven software projects are generally divided into modules that have particular functions inside the project, by using the SourceForge categorization scheme (e.g ignoring these modules) all modules get classified under the same category although this is not necessarily correct and could lead to overly broad categories for projects.

We identified 41 projects out of the 106 projects from the repository present in SourceForge (39%) as being categorized under an overly broad category. For 38 out of these 41 projects, *Sally* was able to produce more specific categories than those given by SourceForge. In most of the cases where developers assigned appropriate and specific categories, *Sally* was able to generate terms closely related to them. *We conclude that Sally generates more descriptive categories for Maven projects than SourceForge.* However, for standalone applications that are categorized under a specialized category in a manual fashion, we can not say that *Sally* produces better categories because there are high-level concepts that can not be abstracted only by looking at information obtained from bytecode.

RQ₂: MVNRepository

MVNRepository assigns tags to projects by extracting information from their pom file¹². Unlike SourceForge, MVNRepository is focused solely on Maven projects, this allows us to do a better comparison with *Sally*. Manual examination showed that in most cases, *Sally* is able to generate tags that are at least as descriptive as those generated by MVNRepository.

¹²We do not include categories in the comparison because we do not have information about how they are assigned to projects

For 27 out of the 167 projects in the repository (16%), at least one of the categories generated by *Sally* exactly matched one or more tags from MVNRepository. Additionally, there are 66 projects ($\approx 40\%$) for which MVNRepository does not have any tag assigned but *Sally* does. Moreover, there is only 1 project for which MVNRepository has a tag and *Sally* does not: *javax.inject-1.jar* and the assigned tag is *javax*, which is filtered by *Sally* because we do not consider file names to be valid categories (e.g project *activation.jar* can not have a category named `activation`).

When present, tags assigned by MVNRepository were mostly considered appropriate for the projects they were assigned to. However, since these tags depend on the `description` tag from the pom file while the ones generated by *Sally* do not, *in the majority of cases Sally was able to produce more descriptive tags for Maven projects than MVNRepository.*

RQ₃: Availability of categories

In order to measure category availability, we counted the number of projects that were indexed on the sites but had no categories or tags assigned. Table 5-6 depicts these quantities. The number of projects that were not categorized by *Sally* is minimal compared to the number of projects without categories or tags assigned by the other approaches. This is a direct consequence of the fact that *Sally* does not rely in any source different to bytecode.

Sally		MVNRepository		SourceForge
Primary	Secondary	Categories	Tags	Categories
2	71	93	67	23
1.20%	42.51%	56.36%	40.6%	21.69%

Table 5-6.: Number of projects without categories per approach.

Since both SourceForge and MVNRepository need a certain set of conditions to be met in order to be able to categorize a project (i.e. manual categorization for SourceForge and a descriptive pom file for MVNRepository), we can conclude that *the availability of categories of Sally is superior than those of SourceForge and MVNRepository.*

5.1.3. Conclusion

We compared *Sally* to two popular online tools with different categorization schemes and found that both SourceForge and MVNRepository have weaknesses that our proposed approach does not. The success of the categorization made by SourceForge strongly depends on developers carefully choosing categories for their projects. On the other hand, the success of the categorization scheme applied by MVNRepository depends on developers adding a description of their projects on the pom files; neither of the requirements can be guaranteed to be fulfilled at all times. Results obtained from this experiment show that *Sally* can produce competitive results without the need for any special requirements from developers.

Comparing tags from MVNRepository and categories from SourceForge to the categories generated by *Sally*, we see that *Sally* can produce competitive and in various cases superior results in terms of how descriptive the generated categories are. Also, *Sally* is able to produce categories in conditions under which neither SourceForge nor MVNRepository can operate.

5.2. Experiment 2: MUDABlue

MUDABlue [14] is an unsupervised categorization approach based on Latent Semantic Analysis. We consider it a good baseline for comparison given the fact that both *Sally* and MUDABlue are unsupervised thus neither approach requires a set of predefined categories to work.

The MUDABlue method consists of 7 steps detailed below:

1. **Identifier extraction**

Only source code is considered for identifier extraction. Comments are left out because the authors consider that the presence of copyright notices and the varying quality of comments among software projects make them less useful for categorization.

2. **Identifier-by-software matrix creation**

Each software system is considered as a document and each identifier as a word to create an identifier-by-software matrix.

3. Identifier filtering

Identifiers that appear in only one software system or more than 50% of them are removed. According to the authors, identifiers that appear on only one system do not provide any useful input for LSA and identifiers that appear on more than half of the projects may be general terms whose removal should not affect the result of the categorization.

4. Latent Semantic Analysis

LSA[18] is applied on the filtered matrix. LSA is a statistical technique for inferring the contextual meanings of terms based on Singular Value Decomposition (SVD).

5. Identifier clustering

Cosine similarities between all pairs of identifiers are calculated using the matrix from the LSA result. Using these similarities, cluster analysis is applied in order to create identifier clusters. Each of these clusters is considered to be a category.

6. Software clustering

Software systems that contain one or more identifiers belonging to an identifier cluster are grouped together. These groups are referred to as software clusters.

7. Cluster labeling

To create labels that describe the software systems contained in each cluster, all identifier vectors comprised in the cluster are added together and the ten identifiers that have the highest values are concatenated to create its label.

Our implementation of MUDABlue is written as a python script; Gensim was used for LSI, corpus processing tasks and cosine similarity calculation. To the best of our knowledge, we have implemented MUDABlue as its authors describe it. In [14], specific details as the exact cluster creation algorithm, whether identifier clustering is done in a fuzzy or hard fashion and the exact parameters used for LSI are omitted. In our implementation, identifier clusters were created by grouping together identifiers for which the similarity score was greater than 0.8 and each one was allowed to belong to multiple clusters. Identifiers were obtained by using the ASM bytecode manipulation framework and stemmed using Apache Lucene. This is the same process used for the proposed categorization approach described in Chapter 4.

5.2.1. Preliminary Results

MUDABlue generated 744 categories for this corpus by following the procedure described in section 5.2, results are available at <http://bit.ly/mblueresults>. On average, each library was labeled with approximately 51% of the generated categories and each category was assigned to 51% of libraries. This indicates that categories were distributed over the whole repository rather than concentrating on a small set of libraries. Figure 5-3 shows how categories were distributed over libraries and vice versa. It is evident that a very high number of categories was assigned to each library, this could produce adverse results when searching for a library in the repository since a query could return too many matches.

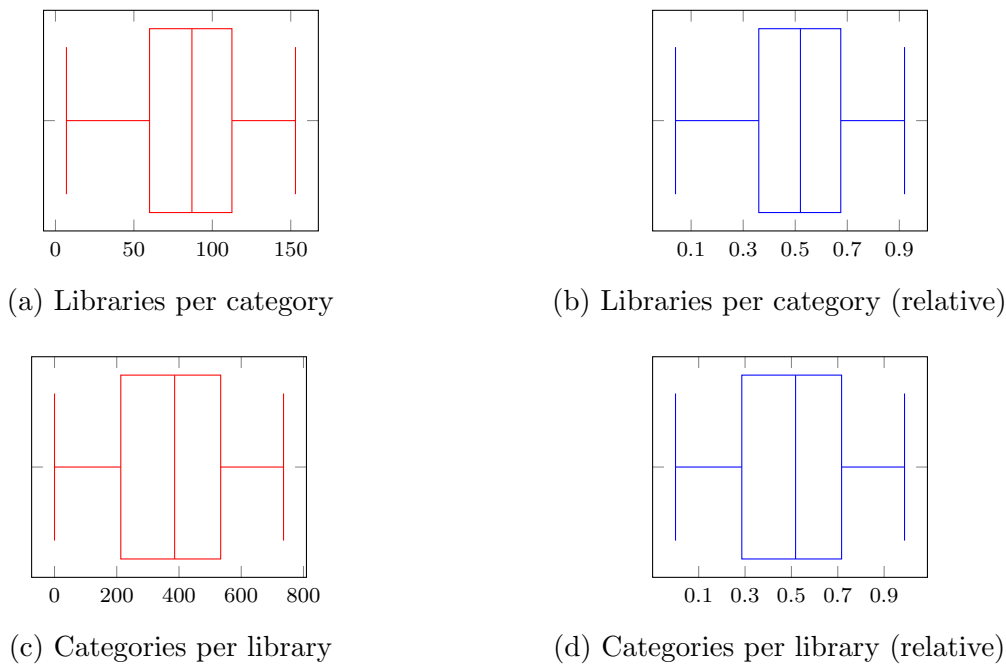


Figure 5-3.: Summary of the distribution of categories given by MUDABlue to the libraries in the corpus. Figures 5-3a and 5-3b show how many libraries belong to each cluster (are labeled with each category). Figures 5-3c and 5-3d show how many categories were assigned to the libraries in the corpus.

Regarding cluster titles, Table 5-7 shows an excerpt of the titles obtained for generated categories. The terms that comprise the titles do not seem to be closely related, which can make significantly different projects to be classified under the same category. To further explain this issue, Table 5-8 shows some of the libraries that were labeled with category 205 (*install, endian, circle, emacsript, keyup, keydown, mousedown*). The libraries shown in this table are related to domains as *web testing, graphics, XML file processing, DOM manipulation, Input/Output* and *Logging*. There does not seem to be a strong relation between them that would lead a manual categorization process to produce this same grouping.

Cluster id	Category name	Size
622	entries, external, grid, human	101
666	build, destroy, alt, apache	101
84	bind, image, clean, timer, indicator, addr, acc, producer, png, asymmetric	102
329	tag, evaluate, domain, day, arithmetic, regexp, azimuth, similar	102
584	modified, lang, mutex	102
604	build, common, authority, color, collapse, closure, banner	102
615	event, execute, evaluate, exclude, exec, exe	102
12	condition, marker, http, destroy, decoration, diff, mpeg	103
59	search, label, limiting, bracket, slot, busy, boot, drawable	103
384	change, manager	103

Table 5-7.: Excerpt of titles obtained for categories by MUDABlue

Cluster 205: (<i>install, endian, circle, emacsscript, keyup, keydown, mousedown</i>)	
Library	
findbugs-3.0.0.jar	batik-svg-dom-1.7.jar
htmlunit-2.15.jar	commons-io-2.4.jar
htmlunit-core-js-2.15.jar	xalan-2.7.1.jar
jackson-core-2.2.1.jar	commons-lang3-3.3.2.jar
logback-classic-0.9.28.jar	dom4j-1.6.1.jar

Table 5-8.: Excerpt of libraries belonging to cluster 205

These results lead us to think that MUDABlue is not a good approach for categorizing Maven repositories since the identifiers obtained from the libraries belonging to them may be significantly more related than those extracted from repositories of standalone software projects such as SourceForge, which produced good results in the past. Nevertheless, a user study was conducted in order to determine how *Sally* and MUDABlue compare on the task of categorizing repositories of Maven projects.

5.2.2. User study

We conducted an evaluative survey to assess the proposed categorization approach. The goal of the survey was to compare the categories generated by MUDABlue and the ones generated by *Sally* in terms of their *Expressiveness* and *Completeness* related to the purpose and application domain of selected projects. *Expressiveness* refers to how much information the set of terms provide about the purpose / application domain of the library. *Completeness* on the other hand, refers to the measure of how much of the purpose / application domain of the library is described by the terms. The context of the survey is a set of 50 randomly selected projects from the corpus used for Experiment 1 and the preliminary experiment with MUDABlue.

Research Questions

The relevant research questions are presented below:

- *RQ₄: How do the categories produced by Sally and MUDABlue compare in terms of their Completeness?*
- *RQ₅: How do the categories produced by Sally and MUDABlue compare in terms of their Expressiveness?*

We define E_{domain} and $E_{purpose}$ to represent the Expressiveness of the categories for describing the application domain and purpose of the projects respectively. Similarly, C_{domain} and $C_{purpose}$ represent the Completeness of the categories for describing the application domain and purpose of projects.

Both research questions directly aim at comparing the categories generated by the approaches. To answer them, we asked developers to rate the 10 terms that form category names for MUDABlue and 10 terms generated by *Sally* for each of the 50 projects using a Likert scale. The terms obtained from *Sally* correspond to 5 primary categories and 5 secondary categories. When no secondary categories were available, 10 primary categories were extracted. All terms were presented to developers without the relevance measure to avoid negative effects on the validity of the study.

Data Collection Process

To find out the perceptions of developers about the terms generated by both approaches, 5 different online surveys, each containing 10 projects for evaluation were created using Google Forms¹³. The survey included demographic questions intended for measuring the programming experience of participants taken from the work done by Feigenspan> et al. [8], the employed guidelines for the survey design were the following:

1. Participants should have enough information for them to understand the purpose and application domain of a project before rating its assigned categories. To ensure this, each question presented both a description of the project to evaluate as well as a link to its official website.
2. We should obtain information about the rationale behind the ratings given by developers. With this in mind, participants were asked to give a brief description of the reasons behind their ratings.
3. The survey should not take more than 45 to 50 minutes to be completed to reduce the possibility of quick answers and drop-out. We estimated that each question would take between 4 and 5 minutes to be answered so each survey type contained 10 questions.

Figure 5-4 shows one of the questions that were present on the surveys. It contains the description of the library *spring-context-2.5.6.jar* as well as a link to the official project site. Participants are asked to rate Expressiveness and Completeness of the presented sets of terms regarding their ability to describe the domain application and purpose of the library, after this, they are asked to give a brief description of the rationale behind the assigned rating.

Results

14 participants completed the survey, which ensured that the categories generated for each project were evaluated at least twice. Figures 5-5a and 5-5b depict the background information we gathered with the questions taken from [8]. The demographic questions that were included in the survey are the following:

1. DQ_1 : On a scale from 1 to 10, how do you estimate your programming experience?
2. DQ_2 : On a scale from 1 to 5, how do estimate your experience with the Java programming language?

¹³<http://forms.google.com>

The following is the description for library spring-context-2.5.6.jar: "Core support for dependency injection, transaction management, web applications, data access, messaging, testing and more."

The official website for spring-context-2.5.6.jar is <http://projects.spring.io/spring-framework/>.

Please rate using a scale from 0 to 5 the expressiveness and completeness of the following sets of words. Afterwards, please give the rationale behind your decision.

SET 1: bean jndi script weaver jmx bean autowiring annotation definition editor

SET 2: escape entries debug control component loader extensions external log level

Please rate the first set of words in the following aspects: *

bean jndi script weaver jmx bean autowiring annotation definition editor

	1	2	3	4	5
Expressiveness for application domain	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Expressiveness for library purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Completeness for application domain	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Completeness for library purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Please rate the second set of words in the following aspects: *

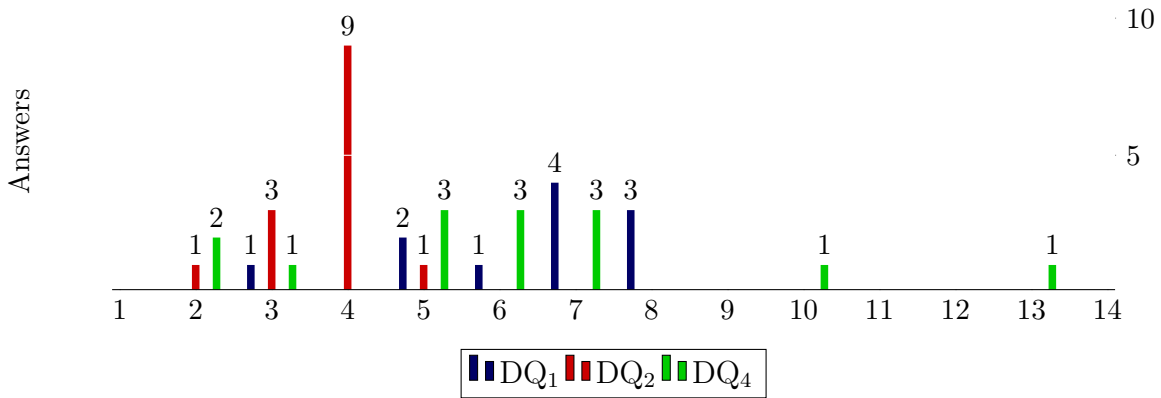
escape entries debug control component loader extensions external log level

	1	2	3	4	5
Expressiveness for application domain	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Expressiveness for library purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Completeness for application domain	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Completeness for library purpose	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

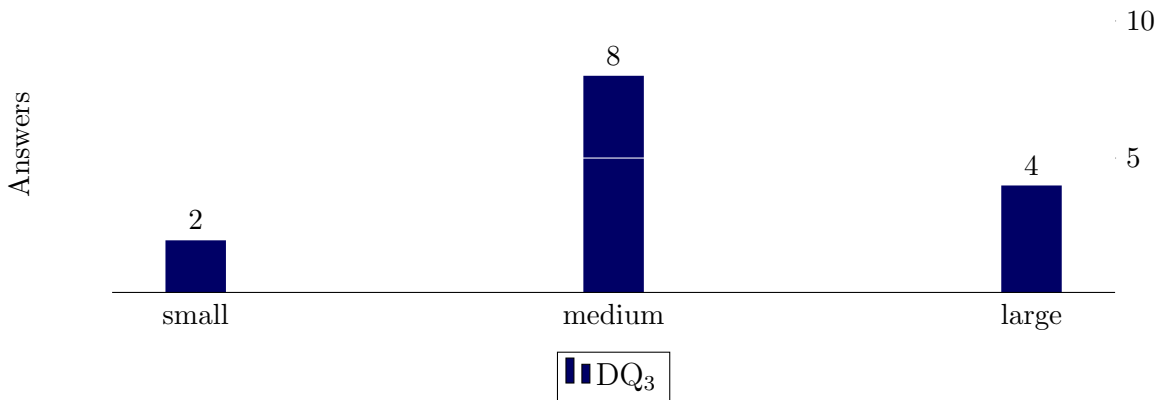
Please give a brief description of the rationale behind your decisions

Figure 5-4.: Sample question taken from the survey.

3. DQ_3 : What has been the typical size of the professional projects you have worked on? (Small for less than 900 lines of code. Medium for 901 to 4000 lines of code. Large for more than 4000)
4. DQ_4 : For how many years have you been programming?



(a) Summary of demographic questions 1,2 and 4.



(b) Summary of demographic question 3.

Figure 5-5.: Summary of reponses to demographic questions.

RQ₄: Completeness

Figure 5-6a depicts the ratings given by developers to the terms provided by both approaches with regard to their completeness. It can be seen that *Sally* substantially outperformed MUDABlue, e.g. *Sally* obtained a score greater or equal to 3 on 72% of the evaluations vs 26% for MUDABlue and the highest possible score on 15% of the evaluations vs 1.4%.

RQ₅: Expressiveness

Regarding *Expressiveness*, similar results were obtained and *Sally* is shown to be superior. For example, Figure 5-6b shows that *sally* obtained a score greater or equal to 3 on 76% of the evaluations vs 28% for MUDABlue and the highest possible score on 16.8% of the evaluations vs 1.1%.

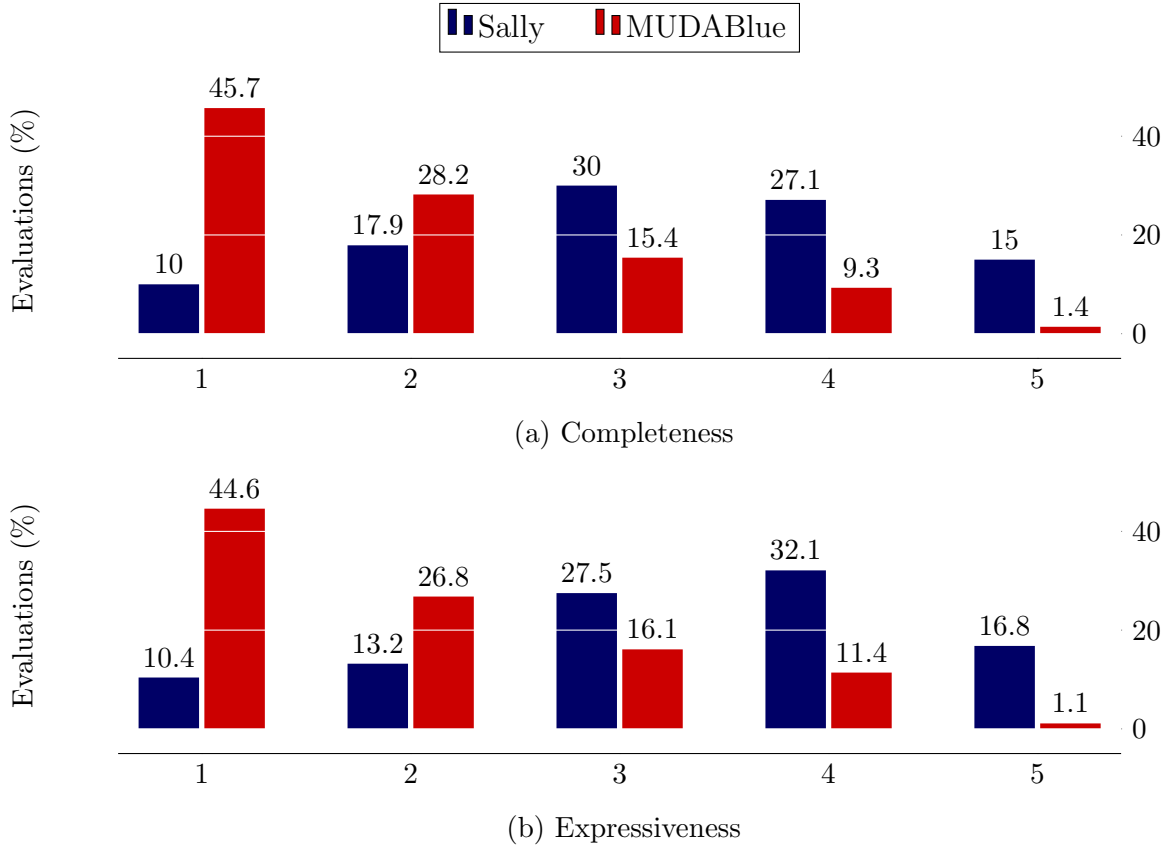


Figure 5-6.: Amount of evaluations per rating for both approaches

Both the results obtained in the preliminary experiment described in section 5.2.1 and the user study suggest that *Sally* is superior to MUDABlue for categorizing Maven projects. However, it is important to consider that MUDABlue was not developed with the intention of being used on these kind of repositories while *Sally* was specifically designed for this particular use case.

To validate that the results for each property (i.e., E_{domain} , $E_{purpose}$, C_{domain} , $C_{purpose}$) are statistically significant, when comparing the answers for Sally and MUDABlue, we used the Mann-Whitney test [6] with $\alpha = 0.05$. We also computed the Cliff's delta d effect size [10] to measure the magnitude of the difference. We followed the guidelines in [10] to interpret the effect size values: negligible for $|d| < 0.147$, small for $0.147 \leq |d| < 0.33$, medium for $0.33 \leq |d| < 0.474$ and large for $|d| \geq 0.474$. Because we are not assuming population normality and homogeneous variances, we used non-parametric methods (Mann-Whitney test, and Cliff's delta). In all the pairwise comparisons (e.g., $E_{domain}(Sally)$ vs $E_{domain}(MUDABlue)$) we found statistical significance (i.e., p-value < 0.05), and the magnitude of the difference is large according to the Cliff's delta coefficient.

Analysis of Open Questions

In general, participants perceived the categories assigned by MUDABlue to be either too general or to be completely unrelated to the project they were attempting to describe. In some cases, results obtained by *Sally* were said to contain words that were unrelated to the projects. The effect of these “less related” categories is mitigated by the use of the relevance measure described in Chapter 4, however, this measure was not presented to the participants in order to guarantee a valid comparison between both approaches.

6. Conclusions

This thesis presents an approach for performing automatic multi-label categorization of Java applications that use Maven as a project management tool. In order to do so, *Sally* only needs to have access to the bytecode of projects that are to be categorized, a requirement easily fulfilled by development environments that use Maven. *Sally* extracts identifiers from projects, filters them by using tags from StackOverflow as a knowledge base of software engineering and weights them using TF-IDF. Additionally, *Sally* extracts dependency information from projects to construct a dependency graph which is later used for another key aspect of the model; category propagation. Using all this information, the model is able to produce categories that are both curated using the knowledge of software developers and weighted according to their relevance for each of the projects, all of this without needing any special action from developers such as adding their own categories or describing their projects using natural language.

All the information obtained from the model is made available to its users via an intuitive web application that besides allowing them to search and browse through the repository, helps them to augment the context of the presented categories using two more features. A tag cloud in which the size of each tag visually represents its relative importance to the project at hand and a concept definition module, an extensible tool that allows *Sally* to obtain the definitions of presented tags using different sources of information.

Moreover, we compared *Sally* with popular online tools with different schemes for categorization of projects (namely SourceForge and MVNRepository) and to MUDABlue[14], a similar approach presented in the past which depends on source code and is also able to produce category names without the intervention of developers.

The comparison to available online alternatives shed light on the weaknesses of categorization schemes that need participation of developers in one way or another. For SourceForge, many projects can end up without any assigned category if developers choose not to add any or if they consider that the predefined set of categories offered by SourceForge does not contain one that properly describes their projects. With regard to this prior definition of the set of categories, it was discovered that on many cases it leads to overly broad categories being assigned to projects that can be described with more specific tags. To deal with this problems, SourceForge allows developers to add any number of free text tags to their projects, but this adds problems related to naming and noise to the category set. In the case of MVNRepository, for the approach to be able to generate tags it its necessary that development teams have the discipline to always add and properly populate the `description` tag of their pom files.

The comparison with MUDABlue showed that the characteristics of Maven repositories can influence negatively the performance of automatic categorization approaches that performed satisfactorily on different kinds of repositories. It is important to mention that MUDABlue was not conceived as a tool for categorizing Maven repositories and that as it was originally presented, it depends on identifiers extracted from source code and not bytecode. It is possible that these particular experiment settings affected the capacity of MUDABlue to produce useful categories for the categorization corpus.

To assess the quality of the categorization produced by *Sally* we conducted a survey with developers with a wide range of programming experience. By only comparing the performance to that of MUDABlue we cannot ensure that the categorization produced by *Sally* is perceived to be good by developers. However, we can look at some other aspects extracted from the survey:

- Over 45% of the scores given to *Sally* in both expressiveness and completeness were greater or equal to 4.
- In most cases, developers reported to be able to convey the purpose of the presented projects by looking at the categories generated by *Sally*.

Furthermore, some of the developers mentioned that they were able to understand the purpose of the libraries because they already knew the terms that were presented to them. This helps support the need for the *Concept Definition* module presented in Chapter 4.

To conclude, we have not only presented a competitive approach that provides meaningful and descriptive labels for projects, we have also shown that it is possible to do so without relying heavily on Machine Learning techniques.

A. Appendix: Experiments corpus

Table A-1 shows the set of projects that were used for the experiments described in Chapter 5.

	GroupId	ArtifactId	Version	Scope
1	antlr	antlr	2.7.7	compile
2	aopalliance	aopalliance	1.0	compile
3	bouncycastle	bcprov-jdk15	140	compile
4	ch.qos.logback	logback-classic	0.9.28	compile
5	ch.qos.logback	logback-core	0.9.28	compile
6	com.beust	jcommander	1.27	compile
7	com.fasterxml.jackson.core	jackson-annotations	2.3.0	runtime
8	com.fasterxml.jackson.core	jackson-core	2.2.1	runtime
9	com.fasterxml.jackson.core	jackson-databind	2.3.3	runtime
10	com.github.jsonld-java	jsonld-java	0.5.0	runtime
11	com.github.jsonld-java	jsonld-java-sesame	0.5.0	runtime
12	com.google.code.findbugs	annotations	3.0.0	compile
13	com.google.code.findbugs	bcel-findbugs	6.0	compile
14	com.google.code.findbugs	findbugs	3.0.0	compile
15	com.google.code.findbugs	jFormatString	3.0.0	compile
16	com.google.code.findbugs	jsr305	3.0.0	compile
17	com.google.guava	guava	17.0	compile
18	com.google.inject	guice	4.0-beta	compile
19	com.google.inject.extensions	guice-multibindings	4.0-beta	compile
20	com.sun	tools	0	system
21	commons-codec	commons-codec	1.9	compile
22	commons-collections	commons-collections	3.2.1	compile
23	commons-io	commons-io	2.4	compile
24	commons-lang	commons-lang	2.6	compile
25	commons-logging	commons-logging	1.1.3	compile
26	dom4j	dom4j	1.6.1	compile
27	javax.activation	activation	1.1	compile

28	javax.inject	javax.inject	1	compile
29	javax.mail	mail	1.4.5	compile
30	javax.servlet	servlet-api	2.5	compile
31	jaxen	jaxen	1.1-beta-8	compile
32	jdom	jdom	1.0	compile
33	jfree	jcommon	1.0.15	compile
34	jfree	jfreechart	1.0.12	compile
35	jivesoftware	smack	3.1.0	compile
36	jivesoftware	smackx	3.1.0	runtime
37	junit	junit	3.8.1	compile
38	log4j	log4j	1.2.17	compile
39	net.java.dev.javacc	javacc	5.0	compile
40	net.sf.trove4j	trove4j	3.0.3	compile
41	net.sourceforge.argparse4j	argparse4j	0.4.4	compile
42	net.sourceforge.barbecue	barbecue	1.5-beta1	compile
43	net.sourceforge.cobertura	cobertura	2.0.3	compile
44	net.sourceforge.collections	collections-generic	4.01	compile
45	net.sourceforge.cssparser	cssparser	0.9.14	compile
46	net.sourceforge.floggy	floggy-persistence-framework	1.4.0	compile
47	net.sourceforge.htmlcleaner	htmlcleaner	2.9	compile
48	net.sourceforge.htmlunit	htmlunit	2.15	compile
49	net.sourceforge.htmlunit	htmlunit-core-js	2.15	compile
50	net.sourceforge.jadex	jadex-applib-bdi	2.4	compile
51	net.sourceforge.jadex	jadex-bridge	2.4	compile
52	net.sourceforge.jadex	jadex-commons	2.4	compile
53	net.sourceforge.jadex	jadex-javaparser	2.4	compile
54	net.sourceforge.jadex	jadex-kernel-application	2.4	compile
55	net.sourceforge.jadex	jadex-kernel-base	2.4	compile
56	net.sourceforge.jadex	jadex-kernel-bdi	2.4	compile
57	net.sourceforge.jadex	jadex-kernel-bdibpmn	2.4	compile
58	net.sourceforge.jadex	jadex-kernel-bpmn	2.4	compile
59	net.sourceforge.jadex	jadex-kernel-component	2.4	compile
60	net.sourceforge.jadex	jadex-kernel-extension-envsupport	2.4	compile
61	net.sourceforge.jadex	jadex-kernel-micro	2.4	compile
62	net.sourceforge.jadex	jadex-model-bpmn	2.4	compile
63	net.sourceforge.jadex	jadex-platform	2.4	compile
64	net.sourceforge.jadex	jadex-platform-base	2.1	compile
65	net.sourceforge.jadex	jadex-rules	2.4	compile

66	net.sourceforge.jadex	jadex-rules-eca	2.4	compile
67	net.sourceforge.jadex	jadex-tools-base	2.4	compile
68	net.sourceforge.jadex	jadex-tools-base-swing	2.4	compile
69	net.sourceforge.jadex	jadex-xml	2.4	compile
70	net.sourceforge.javacsv	javacsv	2.0	compile
71	net.sourceforge.jchardet	jchardet	1.0	compile
72	net.sourceforge.jeuclid	jeuclid-core	3.1.9	compile
73	net.sourceforge.jexcelapi	jxl	2.6.12	compile
74	net.sourceforge.jtds	jtds	1.3.1	compile
75	net.sourceforge.jtransforms	jtransforms	2.4.0	compile
76	net.sourceforge.jwebunit	jwebunit-core	3.2	compile
77	net.sourceforge.jwebunit	jwebunit-htmlunit-plugin	3.2	compile
78	net.sourceforge.messadmin	MessAdmin-Core	4.1.1	compile
79	net.sourceforge.nekohtml	nekohtml	1.9.21	compile
80	net.sourceforge.owlapi	owlapi-api	4.0.0	compile
81	net.sourceforge.owlapi	owlapi-apibinding	4.0.0	compile
82	net.sourceforge.owlapi	owlapi-distribution	4.0.0	compile
83	net.sourceforge.owlapi	owlapi-impl	4.0.0	compile
84	net.sourceforge.owlapi	owlapi-parsers	4.0.0	compile
85	net.sourceforge.owlapi	owlapi-util	3.3	compile
86	net.sourceforge.plantuml	plantuml	8008	compile
87	net.sourceforge.pmd	pmd	5.1.3	compile
88	net.sourceforge.reb4j	net.sourceforge.reb4j	2.1.0	compile
89	net.sourceforge.saxon	saxon	9.1.0.8	compile
90	net.sourceforge.serp	serp	1.14.1	compile
91	net.sourceforge.stripes	stripes	1.5.8	compile
92	net.sourceforge.wurfl	wurfl	1.3.1.1	compile
93	org.antlr	antlr-runtime	3.1.3	compile
94	org.antlr	stringtemplate	3.2	compile
95	org.apache.ant	ant	1.8.3	compile
96	org.apache.ant	ant-launcher	1.8.3	compile
97	org.apache.commons	commons-lang3	3.3.2	compile
98	org.apache.directory.studio	org.apache.commons.io	2.4	compile
99	org.apache.httpcomponents	httpclient	4.3.3	compile
100	org.apache.httpcomponents	httpclient-cache	4.2.5	runtime
101	org.apache.httpcomponents	httpcore	4.3.2	compile
102	org.apache.httpcomponents	httpmime	4.3.3	compile
103	org.apache.xmlgraphics	batik-anim	1.7	compile

104	org.apache.xmlgraphics	batik-awt-util	1.7	compile
105	org.apache.xmlgraphics	batik-css	1.7	compile
106	org.apache.xmlgraphics	batik-dom	1.7	compile
107	org.apache.xmlgraphics	batik-ext	1.7	compile
108	org.apache.xmlgraphics	batik-parser	1.7	compile
109	org.apache.xmlgraphics	batik-svg-dom	1.7	compile
110	org.apache.xmlgraphics	batik-util	1.7	compile
111	org.apache.xmlgraphics	batik-xml	1.7	compile
112	org.apache.xmlgraphics	xmlgraphics-commons	1.3.1	compile
113	org.ccil.cowan.tagsoup	tagsoup	0.9.7	compile
114	org.eclipse.jetty	jetty-http	8.1.15	compile
115	org.eclipse.jetty	jetty-io	8.1.15	compile
116	org.eclipse.jetty	jetty-util	8.1.15	compile
117	org.eclipse.jetty	jetty-websocket	8.1.15	compile
118	org.functionaljava	functionaljava	3.1	compile
119	org.jdom	jdom2	2.0.5	compile
120	org.mortbay.jetty	jetty	6.1.14	compile
121	org.mortbay.jetty	jetty-util	6.1.14	compile
122	org.mortbay.jetty	servlet-api-2.5	6.1.14	compile
123	org.mozilla	rhino	1.7R3	compile
124	org.openengsb.wrapped	net.sourceforge.htmlunit-all	2.8.w1	compile
125	org.openrdf.sesame	sesame-model	2.7.12	compile
126	org.openrdf.sesame	sesame-rio-api	2.7.12	compile
127	org.openrdf.sesame	sesame-rio-binary	2.7.12	runtime
128	org.openrdf.sesame	sesame-rio-datatypes	2.7.12	runtime
129	org.openrdf.sesame	sesame-rio-languages	2.7.12	runtime
130	org.openrdf.sesame	sesame-rio-n3	2.7.12	runtime
131	org.openrdf.sesame	sesame-rio-nquads	2.7.12	runtime
132	org.openrdf.sesame	sesame-rio-ntriples	2.7.12	runtime
133	org.openrdf.sesame	sesame-rio-rdfjson	2.7.12	runtime
134	org.openrdf.sesame	sesame-rio-rdfxml	2.7.12	runtime
135	org.openrdf.sesame	sesame-rio-trig	2.7.12	runtime
136	org.openrdf.sesame	sesame-rio-trix	2.7.12	runtime
137	org.openrdf.sesame	sesame-rio-turtle	2.7.12	runtime
138	org.openrdf.sesame	sesame-util	2.7.12	compile
139	org.ow2.asm	asm	4.1	compile
140	org.ow2.asm	asm-analysis	4.1	compile
141	org.ow2.asm	asm-commons	4.1	compile

142	org.ow2.asm	asm-debug-all	5.0.2	compile
143	org.ow2.asm	asm-tree	4.1	compile
144	org.ow2.asm	asm-util	4.1	compile
145	org.semarglproject	semargl-core	0.6.1	runtime
146	org.semarglproject	semargl-rdf	0.6.1	runtime
147	org.semarglproject	semargl-rdfa	0.6.1	runtime
148	org.semarglproject	semargl-sesame	0.6.1	runtime
149	org.slf4j	jcl-over-slf4j	1.6.6	compile
150	org.slf4j	slf4j-api	1.7.7	compile
151	org.springframework	spring-beans	2.5.6	compile
152	org.springframework	spring-context	2.5.6	compile
153	org.springframework	spring-core	2.5.6	compile
154	org.springframework	spring-web	2.5.6	compile
155	org.w3c.css	sac	1.3	compile
156	oro	oro	2.0.8	compile
157	regex	regex	1.3	compile
158	xalan	serializer	2.7.1	compile
159	xalan	xalan	2.7.1	compile
160	xerces	xercesImpl	2.11.0	compile
161	xerces	xmlParserAPIs	2.6.2	compile
162	xml-apis	xml-apis	1.4.01	compile
163	xml-apis	xml-apis-ext	1.3.04	compile
164	xom	xom	1.0b3	compile

Table A-1.: Categorization corpus

Bibliography

- [1] William W Agresti et al. Software reuse: Developers' experiences and perceptions. *Journal of Software Engineering and Applications*, 4(01):48, 2011.
- [2] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.
- [3] M. Bruno, G. Canfora, M. Di Penta, and R. Scognamiglio. An approach to support web service classification and annotation. In *e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on*, pages 138–143, March 2005.
- [4] Julius Davies, DanielM. German, MichaelW. Godfrey, and Abram Hindle. Software bertillonage. *Empirical Software Engineering*, 18(6):1195–1237, 2013.
- [5] G.A. Di Lucca, M. Di Penta, and S. Gradara. An approach to classify software maintenance requests. In *ICSM'02*, pages 93–102, 2002.
- [6] Sheskin D.J. *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.
- [7] Letha H. Etzkorn and Carl G. Davis. Automatically identifying reusable oo legacy code. *Computer*, 30(10):66–71, October 1997.
- [8] J. Feigenspan, C. Kastner, J. Liebig, S. Apel, and S. Hanenberg. Measuring programming experience. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 73–82, June 2012.
- [9] M. Grechanik, Chen Fu, Qing Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. Exemplar: Executable examples archive. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 2, pages 259–262, May 2010.
- [10] R.J. Grissom and J.J. Kim. *Effect sizes for research: Univariate and multivariate applications*. Taylor and Francis, New York, NY, 2012.

-
- [11] J. Guo and Luqi. A survey of software reuse repositories. In *ECBS 2000*, pages 92–100, 2000.
- [12] Lars Heinemann, Florian Deissenboeck, Mario Gleirscher, Benjamin Hummel, and Maximilian Irlbeck. On the extent and nature of software reuse in open source java projects. In *Proceedings of the 12th International Conference on Top Productivity Through Software Reuse*, ICSR’11, pages 207–222, Berlin, Heidelberg, 2011. Springer-Verlag.
- [13] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue. Automatic categorization algorithm for evolvable software archive. In *Sixth International Workshop on Principles of Software Evolution*,, pages 195–200, 2003.
- [14] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue. Mudablue: an automatic categorization system for open source repositories. In *11th Asia-Pacific Software Engineering Conference*., pages 184–193, Nov 2004.
- [15] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys ’09, pages 61–68, New York, NY, USA, 2009. ACM.
- [16] A. Kuhn. Automatic labeling of software components and their evolution using log-likelihood ratio of word frequencies in source code. In *Mining Software Repositories, 2009. MSR ’09. 6th IEEE International Working Conference on*, pages 175–178, May 2009.
- [17] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. Semantic clustering: Identifying topics in source code. *Inf. Softw. Technol.*, 49(3):230–243, March 2007.
- [18] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.
- [19] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. On using machine learning to automatically classify software applications into domain categories. *EMSE*, 19(3):582–618, 2014.
- [20] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. Gplag: Detection of software plagiarism by program dependence graph analysis. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’06, pages 872–881, New York, NY, USA, 2006. ACM.
- [21] Yoëlle S. Maarek, Daniel M. Berry, and Gail E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Trans. Softw. Eng.*, 17(8):800–813, August 1991.

-
- [22] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [23] C. McMillan, M. Grechanik, D. Poshyvanyk, Qing Xie, and Chen Fu. Portfolio: a search engine for finding functions and their usages. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 1043–1045, May 2011.
- [24] C. McMillan, M. Linares-Vasquez, D. Poshyvanyk, and M. Grechanik. Categorizing software applications for maintenance. In *ICSM'11*, pages 343–352, Sept 2011.
- [25] Yana Momchilova Mileva, Valentin Dallmeier, Martin Burger, and Andreas Zeller. Mining trends of library usage. In *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, IWPSE-Evol '09*, pages 57–62, New York, NY, USA, 2009. ACM.
- [26] J. Ossher, H. Sajnani, and C. Lopes. Astra: Bottom-up construction of structured artifact repositories. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 41–50, Oct 2012.
- [27] Eduardo Ostertag, James Hendler, Rubén Prieto Díaz, and Christine Braun. Computing similarity in a reuse library system: An ai-based approach. *ACM Trans. Softw. Eng. Methodol.*, 1(3):205–228, July 1992.
- [28] S. Raemaekers, A. van Deursen, and J. Visser. The maven repository dataset of metrics, changes, and dependencies. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 221–224, May 2013.
- [29] S. Raemaekers, A. van Deursen, and J. Visser. Semantic versioning versus breaking changes: A study of the maven repository. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, pages 215–224, Sept 2014.
- [30] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.
- [31] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D. Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1, EMNLP '09*, pages 248–256, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
- [32] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *LREC'10*, pages 45–50, May 2010.

-
- [33] D. Romano, S. Raemaekers, and M. Pinzger. Refactoring fat interfaces using a genetic algorithm. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 351–360, Sept 2014.
- [34] H. Sajnani, V. Saini, J. Ossher, and C.V. Lopes. Is popularity a measure of quality? an analysis of maven components. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 231–240, Sept 2014.
- [35] C. Teyton, J.-R. Falleri, and X. Blanc. Mining library migration graphs. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 289–298, Oct 2012.
- [36] Kai Tian, M. Reville, and D. Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *MSR’09*, pages 163–166, 2009.
- [37] Yuan Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pages 44–53, Feb 2014.
- [38] Secil Ugurel, Robert Krovetz, and C. Lee Giles. What’s the code?: Automatic classification of source code archives. In *KDD’02*, pages 632–638, 2002.
- [39] Shaowei Wang, D. Lo, and Lingxiao Jiang. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In *ICSM’12*, pages 604–607, Sept 2012.
- [40] Tao Wang, Huaimin Wang, Gang Yin, C.X. Ling, Xiang Li, and Peng Zou. Mining software profile across multiple repositories for hierarchical categorization. In *ICSM’13*, pages 240–249, Sept 2013.
- [41] Tao Wang, Gang Yin, Xiang Li, and Huaimin Wang. Labeled topic detection of open source software from mining mass textual project profiles. In *Proceedings of the First International Workshop on Software Mining*, SoftwareMining ’12, pages 17–24, New York, NY, USA, 2012. ACM.
- [42] Xin Xia, Xiaozhen Zhou, D. Lo, and Xiaoqiong Zhao. An empirical study of bugs in software build systems. In *Quality Software (QSIC), 2013 13th International Conference on*, pages 200–203, July 2013.