UNIVERSIDAD NACIONAL DE COLOMBIA

Improving Android Applications Searching and Browsing by using Information Retrieval and Static Bytecode Analysis

Carlos Eduardo Bernal-Cárdenas

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
Bogotá D.C., Colombia
2014

Improving Android Applications Searching and Browsing by using Information Retrieval and Static Bytecode Analysis

Carlos Eduardo Bernal-Cárdenas

Thesis Work to Obtain the Degree of:
Magister in Systems and Computing Engineering

Advisor:
Ph.D.(c) Mario Linares-Vásquez

Co-Advisor:
Ph.D. Jairo Hernán Aponte Melo

Research Line:
Software Engineering
Research Group:
Colectivo de Investigación en Ingeniería de Software - ColSWE

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
Bogotá D.C., Colombia
2014

Dedication

"Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it."

— Steve Jobs

A mis padres por su apoyo incondicional en cada una de las decisiones que he tomado durante toda mi vida.

A Karen y Thomas quienes son mi fuente de inspiración diaria y me da fuerzas para seguir adelante.

Finalmente a todos aquellos que no están conmigo pero que están en mi memoria y que siempre me haran recordar, que la alegría es algo fundamental que siempre debemos transmitir a los que nos rodean, sin importar las circunstancias en las que nos encontremos.

Acknowledgements

To my advisor, without whom this would not be possible. Professor Mario has always been a person that motivates you to be better every day. I have never seen a person so dedicated and passioned for his work, moreover he inculcates the job well done and the passion for what you do. I do not have words to say, thank you my mentor!.

In addition, I want to thanks all members of SEMERU group from whom I have learned many things during my stay in 2013, specially professor Denys.

Abstract

A plethora of mobile applications have been developed to satisfy users needs. These applications help users to complete different activities like read books, access to bank accounts, listening to music, write notes, translate text, among others. All the applications are usually published on mobile markets, in which users can download the binary/byte-code that will be executed on the device. These markets provides information such as application description, rating, and related applications that is used when users perform a search. Nonetheless, most of the applications search engines only use textual information extracted from descriptions, applications names, software documentation, and source code. This thesis presents an approach that uses byte-code information such as sensors, permissions, and intents from Android APKs to augment the data that is used to perform the search. We surveyed 9 mobile developers to evaluate the effectiveness of our approach comparing it with other two search engines. As a result we obtained that there is no significant difference in the values of confidence level, precision, and normalized discounted cumulative gain compare to the other search engines. In addition we provided an in-depth analysis to validate and give reasoning about the obtained results.

Keywords: information retrieval, android, search engines, static analysis, bytecode

Resumen

Un gran número de aplicaciones móviles se han desarrollado para satisfacer las necesidades de los usuarios. Estas aplicaciones ayudan a los usuarios a completar diferentes actividades como leer libros, acceder a cuentas bancarias, escuchar música, escribir notas, traducir texto, entre otras. Todas las aplicaciones se publican por lo general en mercados de aplicaciones móviles, en los cuales los usuarios pueden descargar el código binario que se ejecutaráen el dispositivo. Estos mercados de aplicaciones proporcionan información como la descripción de la aplicación, clasificación y aplicaciones relacionadas que es usada cuando los usuarios realizan un búsqueda. Sin embargo, la mayoría de los motores de búsqueda de aplicaciones solo utilizan la información textual extraída de las descripciones, los nombres de las aplicaciones, la documentación del software y el código fuente. Esta tesis presenta un enfoque que utiliza la información de código binario tales como sensores, permisos e "intents"de archivos APK de Android para aumentar los datos que se utilizan para realizar la búsqueda. Se encuestó a 9 desarrolladores móviles para evaluar la efectividad de nuestro enfoque comparándolo con otros dos motores de búsqueda. Como resultado se obtuvo que no hay diferencia significativa en los valores de nivel de confianza, la precisión, y el normalizado de ganancia acumulada comparada con los otros motores de búsqueda. Además, se provee un análisis en profundidad para validar y dar el razonamiento sobre los resultados obtenidos.

Palabras clave: recuperación de información, android, motores de búsqueda, análisis estático, bytecode.

Contents

List of Tables

List of Figures

1 Introduction

A plethora of mobile applications (apps) have been developed to support several activities. Examples of apps that are widely used on mobile devices are calculators, maps, notes, drawings, remote access, books, translators, games, among others. Nowadays it is possible to download and use mobile apps anywhere because of the existence of mobile devices such as smartphones and tablets.

These apps are usually distributed through closed source repositories that are known as app stores or markets; app stores gather applications in one place in such a way that it is not necessary to surf the Internet to find useful/relevant applications. Moreover, closed source repositories are used to store apps in binary/byte-code files and provide the users with some services as searching, browsing, and rating.

Nowadays, multiple companies have created several app markets such as App[1], App World[2], Google Play [10], Market Place[3], OVI[4], Samsung Apps[5], among others. Users looking for apps can use those markets to download relevant apps, and app developers can in some cases get revenues as result of their sales.

Users search for apps in the markets by using a query, and the market retrieves the apps that are relevant to the query by matching it to textual attributes. In addition, browsing capabilities are supported by domain categories; it makes the browsing easier when the users are looking for apps belonging to a specific category. Most of the markets provide users with the same information (attributes) about the apps such as name, developer, price, description and ratings.

Although the markets provide similar information to the users, there are some facts and features that are specific for each market:

- Samsung market place (Samsung Apps) is for apps developed for Android OS. However,

[1]http://itunes.apple.com/us/genre/ios/id36?mt=8
[2]http://appworld.blackberry.com/webstore/
[3]http://www.windowsphone.com/en-us/store
[4]http://store.ovi.com/
[5]http://www.samsungapps.com/

most of Samsung devices used Bada[6] at the beginning.

- Nokia has been continuously working with Symbian OS and publishing apps in his own market called OVI. Nowadays, Nokia was bought by Microsoft, so most of new devices use Windows Phone. However there are Nokia devices that use Android, but these ones use the official Android market by default.
- Apple has its own OS for mobiles devices called iOS and its applications can be found at the App Store.
- BlackBerry (BB) has been making several changes since they launched his own OS. With the release of its new tablet called PlayBook, Android and iOS apps can be ported to BB.
- Google has been leading the development of Android OS; all applications developed for Android can be published at Google Play or unofficial market places like AppBrain[7].
- Microsoft released Windows Phone 8.1 on April of current year. Apps for this OS can be found at Market Place.

1.1 Motivation

According to the Google IO 2013[8] keynote, Android has got more than 900 millions of active devices, and according to AppBrain15, Google Play has more than 700.000 applications in his market place. The amount of apps is increasing daily, because developers find mobile markets as a great opportunity to deliver a solution to huge amount of potential users. Other fact that helps the growth of the amount of apps, in particular Android apps, is that Android markets do not validate the apps before publishing them. Therefore, there is no restriction about the type of apps that can be published in Android Markets.

The market of mobile devices is evolving and users are expecting more from market places to retrieve better results when using code search engines. However, current search engines for code and apps have some weaknesses, in particular for the case of mobile apps:

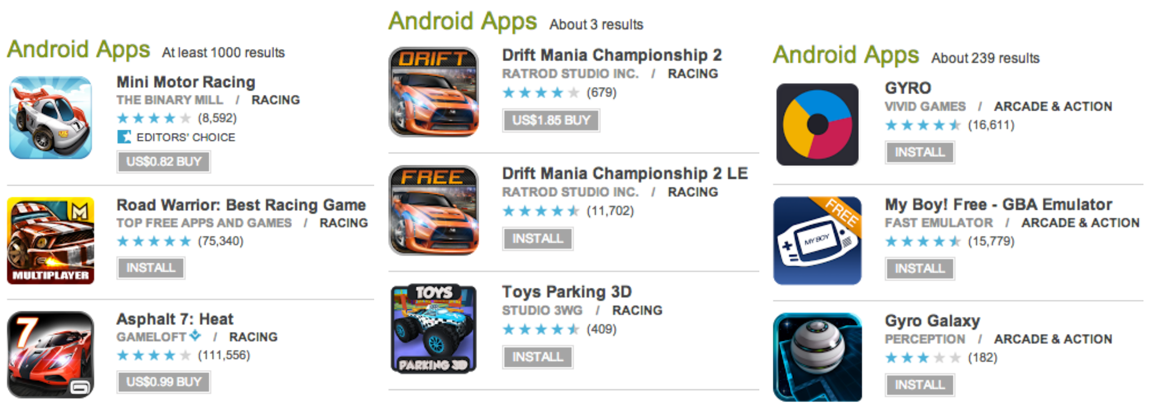
- Previous works have been focused only in code search engines for desktop applications but not in mobile apps.
- Code search engines use only textual information extracted from descriptions, applications names, software documentation, or source code.

[6]http://www.bada.com/
[7]http://www.appbrain.com/
[8]https://developers.google.com/events/io/

- Apps markets provide the users with byte-codes (e.g., APKs in the case of Android Apps, or Jars in the case of JME apps) and previous works have not used information extracted from byte-codes to support code searching and browsing.
- The performance (i.e., precision and recall) of application search engines, such as Google Play, is some cases is low, because the retrieval is based on the description provided by the developers; thus, relevant applications are not retrieved when the keywords in the query are not in the application description.

The following scenarios help to illustrate some of the issues when looking for mobile apps using Google play. Asphalt 7: Heat[9] is a racing game developed by Gameloft. Main characteristics of the game are:

- It appears in the Best Selling in Games[10] list.
- It belongs to the Racing category.
- It uses the gyroscope in the mobile device.



Android Apps At least 1000 results

Mini Motor Racing
THE BINARY MILL / RACING
★★★★☆ (8,592)
EDITORS' CHOICE
US$0.82 BUY

Road Warrior: Best Racing Game
TOP FREE APPS AND GAMES / RACING
★★★★★ (75,340)
INSTALL

Asphalt 7: Heat
GAMELOFT / RACING
★★★★☆ (111,556)
US$0.99 BUY

Android Apps About 3 results

Drift Mania Championship 2
RATROD STUDIO INC. / RACING
★★★★☆ (679)
US$1.85 BUY

Drift Mania Championship 2 LE
RATROD STUDIO INC. / RACING
★★★★★ (11,702)
INSTALL

Toys Parking 3D
STUDIO 3WG / RACING
★★★★★ (409)
INSTALL

Android Apps About 239 results

GYRO
VIVID GAMES / ARCADE & ACTION
★★★★★ (16,611)
INSTALL

My Boy! Free - GBA Emulator
FAST EMULATOR / ARCADE & ACTION
★★★★★ (15,779)
INSTALL

Gyro Galaxy
PERCEPTION / ARCADE & ACTION
★★★☆☆ (182)
INSTALL

(a) Results for query "racing car fastest" (b) Results for query "gyroscope racing car fastest" (c) Results for query "gyroscope game"

Figure **1-1**: Results for 3 different queries

When using the query "racing car fastest"in Google Play, it retrieved more tan 1.000 applications (Figure **1-1**a) with Asphalt 7: Heat in the third position because the query words appear frequently in the description. When using the query "gyroscope racing car fastest"(Figure **1-1**b), Google Play only three games (without Asphalt). The reason is that Asphalt uses the gyroscope but it is not mentioned in the description, and the Drift Mania Championship 2 description has the word gyroscope. Therefore, application/code search engines that only use textual information (i.e., app name and description) in some cases do

[9]https://play.google.com/store/apps/details?id=com.gameloft.android.ANMP.GloftA7HM
[10]https://play.google.com/store/apps/collection/topselling_paid_game

not retrieve all the relevant results.

For the second scenario we wanted to search for a game that uses the gyroscope. Therefore, we used the query "gyroscope game". In Figure 3 the first result is a game called GYRO[11] but it does not use the gyroscope and it does not appear in description; thus, probably stemming was used for the retrieval because it is widely used in Information Retrieval. We guess that after stemming, the root word of gyroscope is "gyro", consequently the app is relevant to the query because the word "gyro"appears several times in the GYRO app description. Other possibility could be that the Google Play search engine use textual comparison algorithms that rank "gyro"as a very similar word to gyroscope.

Both scenarios exemplify how results can be affected when using only textual information. Results from scenario 1 could be improved using features extracted from byte-code. For example, analyzing the `API calls` in am app, we can infer that the app provides features related to using the gyroscope sensor because the app calls the `SensorManager` class with the `Sensor.TYPE_GYROSCOPE` option. Moreover, `API calls` can be extracted from byte-codes by using several tools and byte-code libraries.

1.2 Goals

The main goal is to provide users with a search engine, which allows them to search/prototype apps using two levels of requirements: high-level requirements and system specifications. High-level requirements describe the features developers are looking for, and the system specs describe low-level programming elements such as `sensors`, `permissions` and `intents`. The searching process is simple, the user has to enter a query which represents a feature that the user want to see implemented in an app. The process uses information from two requirement levels described as follows:
High-level requirements formulation: When a user enters a high-level requirement as a query, the search will take into account apps' descriptions in order to retrieve the best results.

Low-level requirements formulation: When a user enters a low-level requirement as a query, the search will take into account `intents`, `sensors`, `permissions`, and `API calls` descriptions in order to retrieve the best results.

Specific Goals are the following:

- Implement a ranking model that uses textual and bytecode information extracted from APK files.

[11]https://play.google.com/store/apps/details?id=pl.submachine.gyro

- Implement an application retrieval model based on Information Retrieval techniques (i.e., Vector Space Model) and static bytecode analysis.
- Compare the proposed approach to state of the art code search engines (i.e., F-Droid and Google Play).

2 Related Work

2.1 Static Code Analysis

Bläsing *et al.* [3] used static and dynamic analysis for searching patterns in decompiled byte-code and trace system calls respectively; dynamic analysis was made with a tool called AASandbox that creates a log of all system calls; static analysis focused on finding patterns in byte-code of suspicious apps.

A similar approach to Bläsing *et al.* was used by Gibler *et al.* [9] to check whether a mobile application is violating android privacy settings. Dex2jar[6] was used to check if methods from byte-code required location, network or Internet permissions. This approach only took in consideration permissions detected in class files. Di Cerbo *et al.* [8] concentrates in permissions only declared in the manifest.xml file.

Mojica Ruiz *et al.* [22] analyzed 4,323 mobile apps downloaded from Google Play. To measure the amount of reused code in the apps; the authors analyzed the apps using a process inspired by the Software Bertillonage presented in [7]. To extract the information they used dex2jar[6] and APKTool[2] to extract the code from APKs from analyzing reuse. Among the results, the authors found that almost the 50% of all apps they downloaded from 5 different categories inherit from the same base class.

Desnos *et al.* [5] developed an algorithm to identify similarities and differences between methods in two applications. The process consists in extracting method signatures and furthermore detecting what the identical, similar, new and deleted methods are. Desnos *et al.* [5] tested their approach using two Skype versions (a version and a hot-fix version) to identify how a bug was fixed.

Shabtai *et al.* [27] proposed an approach based on Machine Learning (ML) techniques to analyze byte-code of apps in order to classify these ones. The authors used 2,285 apps and they extracted features from from `.dex`, and `.xml` files using a `dex` dissasembler. In the results using static code analysis along with ML classification techniques gave good results, the precision was 0.918 and the false positive rates were 0.172.

Linares-Vasques *et al.* [18] analyzed 24,379 free apps from Google Play to see the impact of third-party libraries on analyzing clone detections. They used tools such as dex2jar[6] and APKTool[2] to extract the code from APKs for analyzing clones. In addition the perform an evaluation about the impact of code obfuscation on class clone detection. As a result there is significant difference in the amount of class cloning between obfuscated and non-obfuscated classes.

In summary we have seen there are several works on extracting information from Android applications[3, 22, 5, 18]. This information can be used to extract features as shown by Shabtai[27] or detected components that are reused through different apps[22, 18]. We find useful the information from previous work to use the data extracted from APKs as first step for our approach on improving searching of Android apps.

2.2 Code Reuse Recommenders

Several approaches have worked on recommending code for reusing purposes. Heninger *et al.* [11] proposed a tool called CodeFinder that recommends pieces of code to be reused in a program. CodeFinder reformulates queries suggesting terms to narrow the search using code from software repositories, aiming in the expansion of the query scope.

Michail *et al.* [21] implemented a tool called CodeWeb that discovers library reuse patterns. CodeWeb is a tool based on browsing generalized association rules that takes into account inheritance relationships. Moreover, CodeBreaker [31] is a tool proposed by Ye *et al.* that uses the comments and methods' signatures in source code to retrieve methods in order to reuse them.

Holmes *et al.* [13] developed a tool called Strathcona. This tool extracts the structural context of the code under development and applies an structural matching to recommend structural examples of an specific framework.

Mandelin *et al.* [24] introduce a tool called Prospector that defines a query which describes the wanted code in terms of T_{in} and the output type $Tout$. The results of the query are code snippets that instantiates an object of T_{out} from an input type T_{in}. Sahavechaphan *et al.* [26] present a tool named XSnippet, which extends Prospector[24]. XSnippet provides developers with code fragments using a graph-based code algorithm. In addition this tool aims to support the range of queries and enable mining within and across method boundaries.

Thummalapenta *et al.* [30] proposed a tool called SpotWeb. This tool that mines code examples using an approach based on coldspots, and hotspots. SpotWeb mine code examples in the web and can help developers to understand how to reuse an specific framework.

MAPO [32] is a framework that helps developers to find useful code snippets. It combines frequent subsequence mining with clustering to mine and extract API usage patterns automatically from code snippets. As a result MAPO helps developers to write API client code effectively compared with two other code search engines.

Several code search engines have been implemented having into account textual information in metadata, source code, and software documentation, or structural information extracted from source code. Bajracharya *et al.*[4] created a search engine called Sourcerer, which uses information from source code. Sourcerer supports keyword-based and structure-based querying. It helps developers explore and reuse a large set of open source projects. This tool extracts projects from known repositories like Sourceforge [28] and storage them in a local copy. For the indexing phase, Sourcerer uses Apache Lucene [29].

Hsu *et al.* [12] developed a prototype named MACs. It provides an alternative method for retrieving related code snippets of API usages patterns. In addition it supply reuse patterns relevant to the current developer's task using a context-sensitive environment (i.e. code snippets, and API usages results).

McMillan *et al.* [19] created an code search engine called Portfolio, which combines Natural Language Processing techniques (NLP), with a variation of Page Rank and Spreading Activation Network (SAN). Portfolio helps developers to visualize relevant functions to a query and its usages; 49 professionals evaluated portfolio and the results show that there is strong statistical significance when comparing the accuracy of Portfolio to other code search engines (i.e. Google Code and Koders).
In summary there are different approaches for retrieving relevant code snippets; some of them try to find API usage patterns [32, 12, 21]. On the other hand, we have seen code search engines that uses code under development as the context to query the search engine a retrieve results[31, 24, 26].

2.3 Search Engines for Applications

McMillian *et al.*[20] present a tool called Exemplar, which matches keywords in a query with apps description and keywords in description of APIs documentation; after the users write a query, Exemplar retrieves relevant applications from a ranked engine that processes these applications using description and meta-data in API documents.

Little work has been done on code search engines for mobile apps. Panorama is one of the tools for mobile apps implemented by Jiang *et al.*[17] that proposes an Application Topic Model (ATM) in order to identify latent semantics in apps and then generate code snippets. Three metrics (centrality, formality and usefulness) were used to evaluate differences between application descriptions. Panorama combines the Term Frequency - Inverse Document Frequency (TF-IDF) [23] and the topic score evaluating 12 features. Topic score allows evaluating a relevance of an app according to a query. As a result, Panorama had better performance generating code snippets than some commercial applications markets.

In summary state of art code search engines have used textual and lexical information extracted from the source code or artifacts. However, previous works have been focused to desktop/web applications. Therefore, there is no evidence of code search engines that are based on binary files in mobile environments.

3 Approach

The main idea of our approach is to use all the information that can be extracted from and Android app, that can improve the effectiveness whether we search for an app. In Section 3.2 we explained how we chose the attributes. Then in Section 3.3 we explained the implementation of the bag of words we used in our approach.

3.1 APK Structure

The APK of an application contains all the information that allows an app to be executed. This contains data such as resources, layouts, Java code, images and libraries. Each APK contains a manifest, which is the descriptor of the app describing all components or `Activities`[1] that an app has. We will explain later what is an `Activity`. When a developers creates an application, this has to contain some folders; `res` which is the one that contains all the layouts, translations, themes, and images. `assets` which is the one that contains data that will be loaded in the app (e.g. databases in `JSON` format). Last but not the least `src` that contains all the Java classes of the app.

An `Activity` is a class that implements some methods to handle states in an screen (i.e. `onCreate`, `onStart`, `onResume`, `onPause`, `onStop`, `onDestroy`, `onRestart`). These methods are used in the lifecycle on every activity depending on the state of the screen. `Activity` is one of the most important class on Android, if we want to create a new screen in one app.

3.2 Choosing Relevant Attributes

In our approach we want to extend the search not only using descriptions but including relevant information from internal functionalities of an app. That is the why we choose `Intents`, `Sensors`, and `Permissions` as attributes we want to use to augment the search.

3.2.1 Intents

All screens in one app have to extend the `Activity` class in order to use it as screen. Moreover, whether we want to go to a different `Activity` we need to use the `intent` class. This

[1] http://developer.android.com/reference/android/app/Activity.html

class will create a new instance of the screen we need, which is the only way to do this. So, when we want to use for instance the camera we need to create an `intent` object and call the Activity that will launch the screen of the camera. All the `Intents` we used, have to be declared in the manifest.

We selected this attribute because it will provide information about internal interactions of the app with other Android components.

3.2.2 Sensors

Nowadays most of the apps contains different hardware configurations per device version and vendor[14]. One of these components are the `Sensors` that can be used by developers through the API provided by Android. `Sensors` are divided in three different categories:

- **Motion Sensors:** Sensors in this category measure rotational and acceleration forces in the three axes.
- **Environmental Sensors:** Sensors in this category measure ambient temperature, pressure, illumination, and humidity.
- **Position Sensors:** Sensors in this category measure physical position of the device.

We selected this attribute because it will provide information about internal hardware components used by the app.

3.2.3 Permissions

Android is based in a privilege-separated OS where each app run in a different Linux user ID, group ID, and identities in some cases. Android provides a fine-grained security level called permissions. These permissions allow the application interact and perform operations that could impact other applications, the OS or the user. Android apps that want to share, access resources and data must explicitly declare `Permissions` to use it.

We selected this attribute because it will provide information about resources and data share actions within the app and other apps.

3.2.4 API Calls

State of the art has shown that using documentation of API Calls[20] can improve the effectiveness of search for applications. So, we decided to include Android API descriptions in our approach. We extracted this information from the official web page of Android documentation[2]. We extracted the description of `packages`, `classes`, and `methods`. Moreover we

[2]http://developer.android.com/reference/packages.html

extended each method description adding the corresponding class and package description. Finally the corpus will be a file containing a list of methods with its description that will be used in the *bag of words* (See 3.3).

3.3 Bag of Words

For our approach we want to augment the description of apps, so we decided to use of *bag of words model*[23]. A document is defined as a set of words or terms. Each term appears multiple times on a document, so we can compute the *term frequency* that will tell as the importance of each term in a document. *Bag of words model* ignores the order of terms but the number of occurrences is a key here. In this model we need to take in consideration the fact that some terms can not contribute to the model, for instance in English language articles *the*, *for*, *a*, etc. called *stop words* are removed from the *bag of words*.

3.3.1 Term Frequency - Inverse Document Frequency

Term Frequency

This is the simplest weighting scheme where we map each term t the number of occurrences in a document d.

Document Frequency

This weight scheme of documents is defined as the number of documents in the corpus that contains the term t and is denoted as df_t

Inverse Document Frequency

This weight scheme assigns to the rare terms a high score, whereas the idf_t of a frequent term will be low. The idf_t is computed as shown in Equation (3-1).

$$idf_t = \log \frac{N}{df_t} \tag{3-1}$$

Tf-idf Weighting

For computing the weight of each term t in a document d. The total weight of t in d is computed as shown in Equation (3-2). It makes that the total weight be:

- highest when t occurs multiple times in few documents.
- lower when t occurs few times in d or t occurs in many documents.
- lowest when term occurs in all documents

Besides that we need to compute the score of each document for a given query this can be seen as the similarity between document1 $d1$ and document2 $d2$, which is the same if we see a document as a query.

$$tf\_idf_{t,d} = tf_{t,d} \times idf_t \tag{3-2}$$

We can compute the similarity between two documents using *cosine similarity* for the document d and query q represented as vectors $\overrightarrow{V}(d)$, and $\overrightarrow{V}(q)$ respectively. So, the score of d for a given query is computed as shown in Equation (3-3).

$$score(q, d) = \frac{\overrightarrow{V}(q) \cdot \overrightarrow{V}(d)}{|\overrightarrow{V}(q)||\overrightarrow{V}(d)|} \tag{3-3}$$

3.4 Implementation

We developed a web application composed by 2 main modules as shown in figure **3-1**. The first module is `Static Analyzer` that extracts meta data information from Google Play and APKs. The second is `Search Engine Core` that applies IR techniques to based on VSM to retrieve apps combining description information from `Sensors` , `Intents` , `Permissions` and `API Calls` .

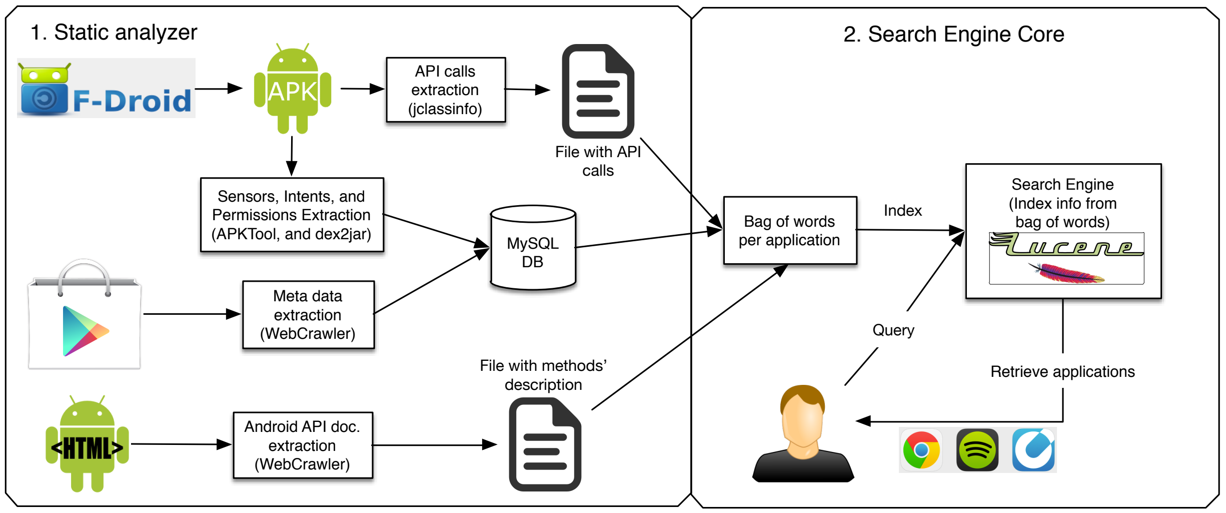

1. Static analyzer
F-Droid
APK
API calls extraction (jclassinfo)
File with API calls
Sensors, Intents, and Permissions Extraction (APKTool, and dex2jar)
MySQL DB
Meta data extraction (WebCrawler)
<HTML>
Android API doc. extraction (WebCrawler)
File with methods' description
2. Search Engine Core
Bag of words per application
Index
Search Engine (Index info from bag of words)
Lucene
Query
Retrieve applications

Figure **3-1**: Web Application Architecture

3.5 Static Analyzer

This module is implemented in java and extracts information from APKs, Google Play and Android API documentation.

3.5.1 APK Information Extraction

We collect information of `Sensors` , `Intents` and `Permissions` . There are different ways to extract information from sensors. the first one is using the manifest because all `Sensors` that are used in an APK should be declared there. However, it is not mandatory to follow that rule, so we focused on a different approach. We extracted this information from byte-code, most specifically smali code, which is an intermediate representation for the Dalvik Virtual Machine (DVM).

Moreover we used APKTool[2] to extract smali code and from this code look for the binary instantiation of a SensorManager class using grep linux command. To do this we need to look for the string

```
Landroid/hardware/SensorManager;->getDefaultSensor(I)Landroid/hardware/Sensor;
```

which is the smali translation for the following Java code

```
mSensorManager.getDefaultSensor(Sensor.<SENSOR_TYPE>);
```

and then we look in the next lines the value assigned to **const/4**. Finally we compared that value con the values defined in Android documentation[3].

By the other hand for `Permissions` and `Intents` we do not need to look for them in the byte-code. It is mandatory to define these attributes in the manifest. To extract `Permissions` we need to look for the XML tag **uses-permission** in order to extract all nodes that declare the usage of an specific `permission` in it. Besides that for `Intents` we use the tag **intent-filter** to extract all nodes that declare an `intent` in it.

Finally all the information is storing in a MySQL database (see 3.5.2).

3.5.2 Google Play Information Extraction

We extracted meta-data information that which is the one that an user can see in a web page of an specific Google Play app. This meta-data contains information about each app such as name, description, developer, category, rating, downloads, version, related apps and apps from the same developer. Besides all the information is stored in a MySQL database (see 3.5.2).

[3]http://developer.android.com/reference/android/hardware/Sensor.html

Database Structure

The database is designed to store all information showed in a single web page from an app. It is composed by 13 tables (see figure **3-2**) and the information stored in every table is described at following:

- APP_INFO_FULL: This table stores information such as name, description, downloads, package and more of the app.
- SENSOR: This table has information of all possible sensors that Android supports.
- APP_INFO_FULL_SENSOR: This table has information of sensors that every app can use.
- APP_INFO_SHORT: Every single app has related and other developer apps, this table store that information.
- VERSION: It contains versions of the app.
- REVIEW: This table store all reviews for every app.
- PERMISSION: It contains information from permissions supported by Android.
- CHANGE_LOG: It contains information that occurs; like new update of application, new range of downloads and so on.
- INTENT: It contains internal information extracted from Android PacKage (APK) that contains information from execution of other applications or internal components.
- RATING: This table contains information about rating of the app per range from 1 to 5.

3.5.3 API Documentation Information Extraction

We extracted description of packages, classes and methods. In addition we need to extract descriptions of `sensors`, `permissions`, and `intents`.

Sensors, Permissions and Intents Extraction

We created a mini-crawler to extract these descriptions, but in the case of `Intents` there are a lot of types that are defined outside `Intent.java` from Android SDK. Consequently we had to manually look for all the Android classes that contains the descriptions of different `intents'` types. the list of classes is presented in Table **3-1**. The output of this crawler are three files that for each line contains the name of the <*package,description*> in the case of package information, <*class,description*> in the case of classes description, and <*class,* *method signature, description>* in the case of methods information. Finally we merge these three files in one, that in each it contains *<return type#package.class.method#parameters type&&&package description+class description+method description>* in order to make the comparison with `API calls` easier.

API Calls Extraction

In order to extract `API calls` we use jclassinfo [15] to achieve it. First we used dex2jar [6] to generate a `.jar` file from the `APK`, then we extract all `API calls` traversing the `.class` files inside the `jar` and filter calls by packages that only belongs to the Android SDK (i.e. packages that start with *android.**).

Corpus Generation

Finally we generate five files per app; The first only contains `description` extracted from Google Play, the second one descriptions of all `sensors`, the third one descriptions of all `intents`, the fourth one descriptions of all `permissions`, and the last one contains descriptions of all `API Calls` per app. These files are used in the Section 4 for evaluating our approach.

3.6 Search Engine Core

This component uses information pre-processed by the static analyzer. This module uses the combination of all corpus generated aforementioned. We implemented this engine using a project from The Apache Software Foundation called Lucene[29] with an implementation of the Vector Space Model (VSM) in order to rank the list of apps relevant for a query.

Therefore we created a bag of words per application, that contains information such as `API calls`, `permissions`, `sensors`, and `intent` descriptions. Then using Lucene we index and rank the corpus for all applications using TF-IDF and cosine similarity. Later when a developer enters a query that is based on a feature which interested in, or a feature of a new application that the developer wants to implement. It retrieves a top-5 related apps for a given query as in Exemplar.

3.6.1 Lucene Configuration

In the configuration of Lunece we are using the `EnglishAnalizer` that by default includes the following:

- It uses and standard tokenizer.

Table **3-1**: Urls from where we extracted descriptions for packages, classes, methods, sensors, permissions and intents

Url (http://developer.android.com)	Type
.../reference/packages.html	Packages
.../reference/classes.html	Classes and Methods
.../guide/topics/sensors/sensors_overview.html	Sensors
.../reference/android/Manifest.permission.html	Permissions
.../reference/android/content/Intent.html	Intents
.../reference/android/provider/Telephony.Sms.Intents.html	Intents
.../reference/android/app/admin/DeviceAdminReceiver.html	Intents
.../reference/android/view/accessibility/AccessibilityNodeInfo.html	Intents
.../reference/android/accounts/AccountManager.html	Intents
.../reference/android/appwidget/AppWidgetManager.html	Intents
.../reference/android/nfc/NfcAdapter.html	Intents
.../reference/android/nfc/NfcAdapter.html	Intents
.../reference/android/bluetooth/BluetoothAdapter.html	Intents
.../reference/android/bluetooth/BluetoothHeadset.html	Intents
.../reference/android/bluetooth/BluetoothDevice.html	Intents
.../reference/android/provider/MediaStore.html	Intents
.../reference/android/hardware/usb/UsbManager.html	Intents
.../reference/android/net/ConnectivityManager.html	Intents
.../reference/android/net/wifi/WifiManager.html	Intents
.../reference/android/speech/tts/TextToSpeech.Engine.html	Intents
.../reference/android/text/style/SuggestionSpan.html	Intents

- It uses an English possessive filter.
- It uses a lower case filter.
- It uses a basic stop word list defined in `StopAnalyzer.java`.
- It uses the PorterStemmer [25].

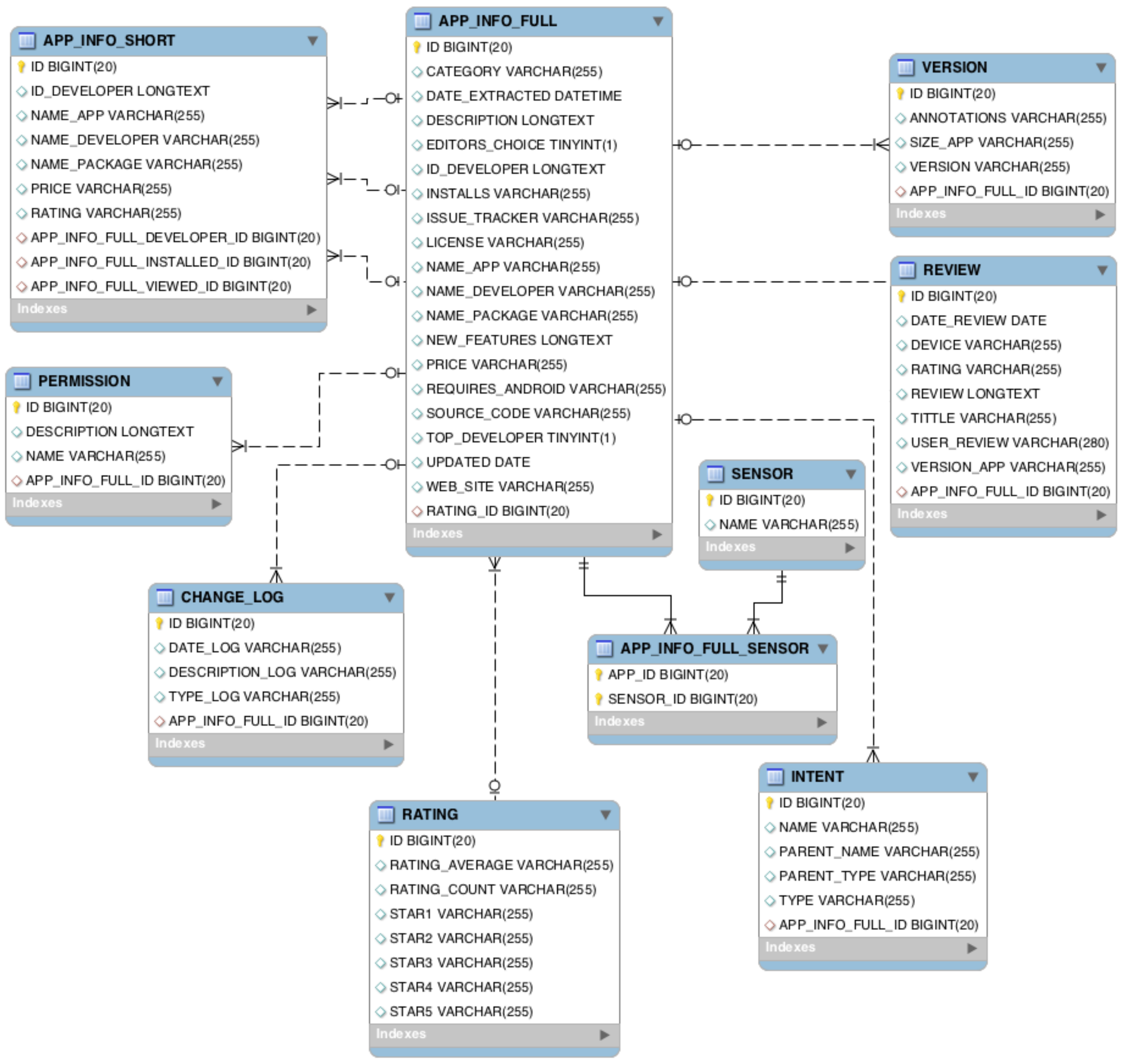


Figure **3-2**: Static database structure.

4 Evaluation

This section explains the design and the evaluation of a survey that aimed at answering research questions formulated in Section 4.1.1.

4.1 Case study: Evaluating Search Engines

The *goal* of this study is survey Android developers with the *purpose* of understanding the benefits of including Android specific attributes (i.e. `Sensors`, `Intents`, `Permissions`, and API Calls) on searching apps. The *context* consists on 1,209 open source apps downloaded from F-Droid market. The *quality focus* is on users' perception and the impact that searches can have using only apps descriptions.

4.1.1 Study Design

In the following, we present the design and planning of the study, by explaining the context selection, research questions, data collection and analysis method.

Context Selection

We downloaded 1,212 APKs from F-Droid but we discarded 3 of them because we could not get the jar file from these ones to extract the `API calls` due to errors in the extraction. So, we use 1,209 APKs in our corpus.

This study consist on evaluating the effectiveness of our approach comparing it with other two search engines. For doing this we need to generate some queries to evaluate the results retrieved by them. So, we followed these process to generate the tasks used in the case study. First, we went to the top two applications in 13 categories on Google Play and extracted manually the description of features. In addition we split in sentences the text because in most of the cases, we had one feature per line. Finally we randomly chose one of these sentences as query. This process was repeated for all the top-2 apps in each of the 13 categories. As a result we ended up with 26 queries (see Table **4-1**) extracted from Google Play descriptions in order to avoid biased queries. In addition we added 6 low level queries in Table **4-2** (i.e. **Q27, Q28, Q29, Q30, Q31, Q32**) to see the behavior of search engines using features close related with hardware components.

Table **4-1**: Queries extracted from 13 categories of Google Play

Query Id	Task
Q1	GENERATE PLOTS
Q2	SHARE TEXT
Q3	ACCEPT CREDIT CARD PAYMENTS
Q4	SHARE IMPORTANT PHOTOS, VIDEOS AND OTHER DOCUMENTS
Q5	SAVE YOUR FAVORITE COMICS
Q6	READING EXPERIENCE WITH HORIZONTAL-VERTICAL
Q7	SHARE YOUR PHONE'S INTERNET
Q8	SEARCH FOR PEOPLE AND GROUPS
Q9	MANAGING THE FLOW OF ASSIGNMENTS
Q10	ANSWER SHORT QUIZZES
Q11	SEND PHOTOS, MUSIC, AND VIDEOS ON YOUR ANDROID
Q12	RATE YOUR FAVORITE SHOWS AND MOVIES
Q13	TRACK YOUR PAYMENTS
Q14	PAY BILLS AND CREDIT CARDS
Q15	PERSONALIZED FITNESS PLAN
Q16	SHOW YOU ALL OF YOUR FITNESS DATA
Q17	DOWNLOAD PHOTOS IN ORIGINAL SIZE
Q18	STICKERS TO SLAP ALL OVER YOUR PHOTOS
Q19	SHARE SPECIFIC LOCATIONS
Q20	RECORD VIDEOS
Q21	CUSTOMIZABLE WIDGETS
Q22	AUTO CORRECTION
Q23	SHARE YOUR MEDIA WITH FRIENDS
Q24	RECORD A NEW AUDIO CLIP TO EDIT
Q25	CUSTOMIZABLE THEME-SKINS
Q26	ORGANIZE IMPORTANT CONTACTS

Research Questions

Our study aims at empirically answering the following research questions (RQs):

RQ$_1$: *What kind of information extracted from byte-codes could be used to improve the results of textual-based application code search engines?* This RQ is designed to evaluate recommendations of other search engines that are based only on apps' description versus our approach.

RQ$_2$: *What ranking models can be used to build an Android apps search engine when combining textual information and information extracted from byte-codes?* This RQ is designed to evaluate whether results can help the user to understand how the feature entered in the query works.

RQ$_3$: *Will we retrieve better results when combining textual information and information*

Table **4-2**: Queries created based on low-level requirements

Query Id	Task
Q27	ALARM ACCELEROMETER
Q28	CHANGE ORIENTATION
Q29	MEASURE TEMPERATURE
Q30	TAKE A PICTURE
Q31	WRITE INTO THE DISK
Q32	SEND SMS

extracted from byte-codes? This RQ is designed to evaluate which of the approaches retrieves better results.

4.1.2 Survey Design

This survey will help to answer $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$. For this study we will compare Exemplar [20], Google Play and our approach. We are comparing our approach with the state of the art. We implemented the approach of Exemplar using Lucene and the same corpus of our approach.

Generating Exemplar Results

To validate our approach we need to have a base line. We selected Exemplar which is one search engine from the state of the art. For implementing Exemplar[20] we use Lucene[29]. First, we extracted all the descriptions for all methods from Android SDK. Then we indexed the description of each method using Lucene. Moreover we need to define a fixed top-k number of `API Calls` to be used in the ranking model of Exemplar. One of the versions od Exemplar implements a combined ranking model using description of the system and descriptions from documentation extracted from the APIs used in each system.

Furthermore, to implement this model we set the top-k `API Calls` to a large number see Figure **4-1** for all queries, then we found that the maximum number of `API Calls` that a query retrieves was 14700. So, we computed the overlap with the `API Calls` for all the queries as in Exemplar [20]. In order to cover more than the 80% of the `API Calls` we need to set the top-k to 4000. In the case of exemplar they used 200, our guessing is that the difference is due to the number of `APIs` in the Android SDK.

Furthermore, we used Equation (4-1) to compute the score of every app based on its `API Calls` . Where p is the number of `APIs` retrieved for the query, $|A|^j$ is the total number of `API Calls` in one application j, n is the number of occurrences of that specific `API call` in the app and C is the score of the `API`.

$$S_{app}^{j} = \frac{\sum_{i=1}^{p} n_i^j \cdot C_i^j}{|A|^j} \tag{4-1}$$

Then we compute the total score (i.e. S_{total}) for all the apps see (4-2). We fixed the value of α to 0.5 to compute the final score for each app using Exemplar's approach. For the Study we use top-5 apps for Exemplar.

$$S_{total}^{j} = \alpha \cdot S_{app}^{j} + \alpha \cdot S_{descr.}^{j} \tag{4-2}$$

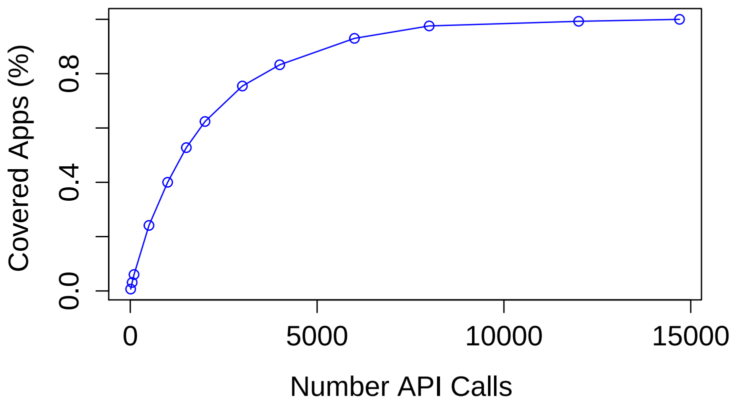


Figure **4-1**: Overlap for different values of top-k of the `API Calls`

Generating Google Results

We manually executed all the queries and extracted the top-5 apps that Google Play retrieved.

Generating Our Approach Results

We automatically extracted the top-5 applications of all queries using our approach, so it uses the information from Sensors, Intent, Permissions, API Calls, and app description.

Creating the Survey

Each question of the survey has a set of five mobile Android applications, that were the result of a search using an specific query (i.e. Q1, Q2, Q4, Q7, Q8, Q12, Q14, Q16, Q19, Q21,

Table **4-3**: Surveys' Distribution

Group1	Group2	Group3
Queries from 1-5 for ALL	Queries from 1-5 for EXE	Queries from 1-5 for GOO
Queries from 6-10 for GOO	Queries from 6-10 for ALL	Queries from 6-10 for EXE
Queries from 11-15 for EXE	Queries from 11-15 for GOO	Queries from 11-15 for ALL

Q24, Q27, Q29, Q30, and Q31). The task for the participants is to evaluate how relevant are the applications on containing/implementing each query. This survey uses a four-level Likert scale.

Table **4-4**: Search Engines to Evaluate

Id	Search Engine	Corpus description
EXE	Exemplar	Corpus created with API calls description and app descriptions
GOO	Google Play	Results from Google Play
ALL	Our approach	Corpus created with sensors, intents, permissions, API calls, and app descriptions

Furthermore, we defined hypotheses to evaluate the results in the survey in Section 4.1.3.

4.1.3 hypotheses

In this section we defined a *null hypotheses* H_{a-null} and an *alternative hypotheses* $H_{a-alte.}$ where a is one of the attributes we want to evaluate. Moreover we want to see if there is a significant difference between the group of our 3 search engines within confidence level, precision, and Normalized Discounted Cumulative Gain (NDCG).

Here we describes separately the hypotheses for each attribute we want to analyze from the results of the study. These hypotheses will aim to answer **RQ1** and **RQ2**.

Confidence Level

The confidence level is the value assigned by participants in the study and is based on a four-level Likert scale. The guidelines for assigning the confidence level are the following:

1. **Completely Irrelevant:** The is participant is confident that the application does not implements the feature described in the query

2. **Mostly Irrelevant:** there is a very small chance that the application implements the feature described in the query

3. **Mostly Relevant:** There is a chance that the application implements the feature described in the query
4. **Highly Relevant:** The participant is confident that the application implements the feature described in the query

We defined the confidence level hypotheses as the following:

- H_{0-null}: There is no significant difference in the value of confidence level per task between participants who use EXE, GOO, and ALL.
- $H_{0-alte.}$: There is significant difference in the value of confidence level per task between participants who use EXE, GOO, and ALL.

Once we tested the *null hypotheses* and if only if there is significant difference we will test the following hypotheses that compares all the 6 search engines with the confidence level value:

- H_{0_1}: There is no significant difference in the value of confidence level between EXE and GOO.
- H_{0_2}: There is no significant difference in the value of confidence level between EXE and ALL.
- H_{0_3}: There is no significant difference in the value of confidence level between GOO and ALL.

Precision

The precision is computed as shown in Equation (4-3), where retrieved will be always 5 because we are using top 5 results and relevant are the number of apps per query that received an score of 3 and 4 (i.e. Mostly relevant and Highly relevant).

$$Precision = \frac{\texttt{Relevant}}{\texttt{Retrieved}} \tag{4-3}$$

We defined the precision hypotheses as the following:

- H_{1-null}: There is no significant difference between the precision per task between participants who use EXE, GOO, and ALL.
- $H_{1-alte.}$: There is significant difference between the precision per task between participants who use EXE, GOO, and ALL.

Once we tested the *null hypotheses* and if only if there is significant difference we will test the following hypotheses that compares all the 6 search engines with the precision value:

- H_{1_1}: There is no significant difference in the value of precision between EXE and GOO.
- H_{1_2}: There is no significant difference in the value of precision between EXE and ALL.
- H_{1_3}: There is no significant difference in the value of precision between GOO and ALL.

Normalized Discounted Cumulative Gain

We used Normalized Discounted Cumulative Gain (NDCG) proposed by Järvelin *et al.*[16] which Al-Maskari *et al.* [1] showed that these metrics can be used to analyze the effectiveness of search engines. The NDCG is computed as shown in Equation (4-5) and it evaluates the effectiveness of one recommendation giving a higher ranking based on the position in the top-k than the irrelevant results.

$$DCG = S_1 + \sum_{i=2}^{5} \frac{S_i}{\log_2 i} \tag{4-4}$$

S_1 represents the score of the first result, S_i represents the score in ith position. NDCG is calculated as shown in Figure (4-5).

$$NDCG = \frac{DCG}{iDCG} \tag{4-5}$$

We defined the NDCG hypotheses as the following:

- H_{2-null}: There is no significant difference between the NDCG per task between participants who use EXE, GOO, and ALL.
- $H_{2-alte.}$: There is significant difference between the NDCG per task between participants who use EXE, GOO, and ALL.

Once we tested the *null hypotheses* and if only if there is significant difference we will test the following hypotheses that compares all the 6 search engines with the precision value:

- H_{2_1}: There is no significant difference in the value of NDCG between EXE and GOO.
- H_{2_2}: There is no significant difference in the value of NDCG between EXE and ALL.
- H_{2_3}: There is no significant difference in the value of NDCG between GOO and ALL.

5 Empirical Results

This section reports the results of the study aimed at answering the research questions formulated in Section 4.1.1.

5.1 Variables

The main independent variable is the search engine (i.e. EXE, GOO, ALL) that the participants used for finding Android apps. The dependent variables are the confidence level, precision, and NDCG. We report the result of these variables in this section.

5.2 Survey Results

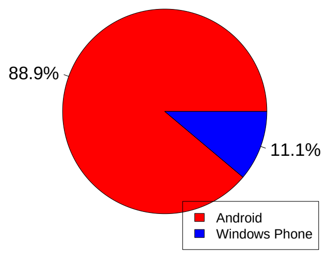
We had 9 participants in our study in which each participant analyzed results for 15 different queries. In each query the participants answer the level of confidence regarding to whether the app contains/implements the feature described in the query. According to the demographic questions 100% of the participants have used a mobile market, 100% of participants have used a mobile app, 11.1% of participants mostly develop for Windows Phone while the rest 88.9% mostly develop for Android see Figure **5-1**a.

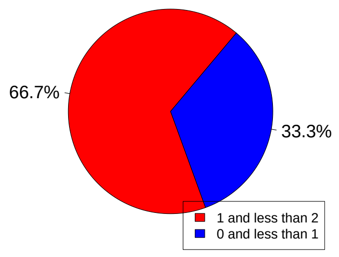
Regarding to the knowledge they have in mobile apps development, 66.7% of the participants have been a mobile developer between 1 and less than a year and the rest 33.3% have been a mobile developer between 0 and less than a year see Figure **5-1**b.

In the next section we are going to show the results for each of the three *null hypotheses.*

5.3 Testing the Null Hypotheses

We used Kruskal-Wallis Test to evaluate the 3 *null hypotheses* (i.e. H_{0-null}, H_{1-null}, H_{2-null}). As shown in Figure **5-2**, we visually present three metrics a confidence level, precision, and NDCG in the boxplot that compares the 3 search engines. In Figure **5-4** and Figure **5-5** is shown the distribution of the confidence level per query.





(a) Operating system of expertise (b) Years of mobile development

Figure **5-1**: Results of some demographic questions

5.3.1 Null Hypothesis - Confidence level

The results from the Kruskal-Wallis Test confirm that there is no statistical significant between the search engines ALL, GOO, EXE for the confidence level with a $p-value = 0.7035$ at $\alpha = 0.05$. Based on these results we accept the *null hypothesis* H_{0-null}

5.3.2 Null Hypothesis - Precision

The results from the Kruskal-Wallis Test confirm that there is no statistical significant between the search engines ALL, GOO, EXE for the confidence level with $p-value = 0.8429$ at $\alpha = 0.05$. Based on these results we accept the *null hypothesis* H_{1-null}

5.3.3 Null Hypothesis - NDCG

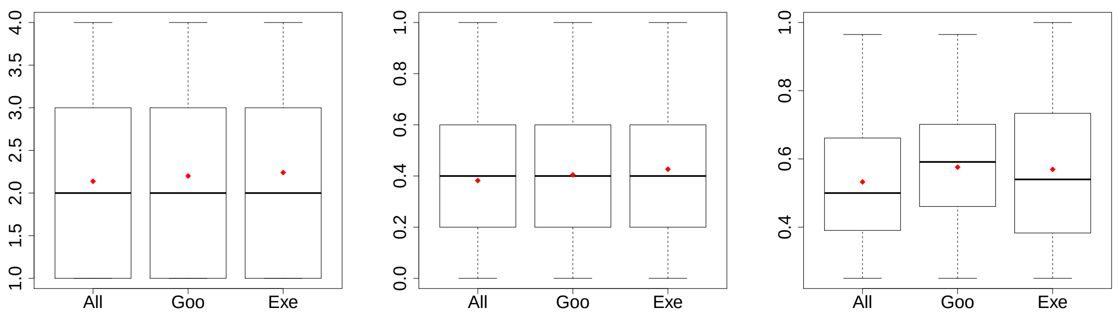
The results from the Kruskal-Wallis Test confirm that there is no statistical significant between the search engines ALL, GOO, EXE for the confidence level with $p-value = 0.5281$ at $\alpha = 0.05$. Based on these results we accept the *null hypothesis* H_{2-null}

5.3.4 Analysis per Query

In this section we provided an in-depth analysis through all the 15 queries used in the study. It will help to understand the results shown in Figure **5-2**. We want to give some possible explanations about the results obtained per search engine by query.

Q1: GENERATE PLOTS

- *ALL:* Apps from the results contains an screen that shows the utilization of plots.



(a) Confidence Level (b) Precision (c) NDCG

Figure **5-2**: Statistical summary of the results od the study for confidence level, precision, and NDCG

- *GOO:* Three of the applications generate plots for actresses and actors, the rest generates graphs.
- *EXE:* According to the screenshots one of the applications does not generate any plot. Additionally in the descriptions there is not insight that this app generates plots.

According to the observation we can say the following; the ambiguity of the query can refer to two different features that can affect the evaluation in the case that users were focused in only one meaning, or multiple minings.

Q2: SHARE TEXT

- *ALL:* According to the screenshots and the descriptions 3 out of 5 apps implement the feature of share text. However 1 application can share complete files and the last one is not related.
- *GOO:* According to the screenshots and the descriptions 2 of the applications implements the feature, other two share different things that a text and the last one is not related
- *EXE:* According to the screenshots and the descptions 3 out of 5 apps implements the feature, one of them don't and the last ones is not clear the purpose of the app. So it is confusing if this apps implements the feature, after installing the app is not clear how the app works.

According to the observation we can say the following; first for 3 of `ALL`'s apps implement the feature share text. Second, the functionality of some applications is confused or not well described.

Q4: SHARE IMPORTANT PHOTOS, VIDEOS AND OTHER DOCUMENTS

- *ALL:* According to the screenshots and descriptions 4 apps clearly are able to share files, images, videos, etc. The other app is no related with the query.

- *GOO:* According to the screenshots and the descriptions 4 of the 5 apps implement the feature.

- *EXE:* According to the screenshots and the descriptions 4 of the 5 apps implement the feature.

According to the observation we can say the following; first, for all the search engines 4 of the 5 apps implements the feature. Second, the answer from participants compare to one of the authors doesn't not agree, even when 4 of 5 apps contains the intents to share files for `ALL` approach.

Q7: SHARE YOUR PHONE'S INTERNET

- *ALL:* According to the descriptions only one app implements the feature, the rest are not related.

- *GOO:* According to screenshots and description 2 of the 5 apps implement the feature. Other two apps share files, and the last one contains the word share in the description but it invites users to share thought with developers.

- *EXE:* None of the results implement the query, however they share content or file types. *According to the observation we can say the following; first, for `EXE` apps from results does not implement the feature but they are related with share content or files. Second, for the other search engines only few of the results implements the feature. Finally, in our dataset there are not apps that share phone's Internet.*

Q8: SEARCH FOR PEOPLE AND GROUPS

- *ALL:* None of the results implement this feature.

- *GOO:* All of the results implement this feature

- *EXE:* None of the results implement this feature. *According to the observation we can say the following; first, in `GOO` we have much more apps than in our data set. Second, none of the apps in our dataset are related with this query. we performed a search in our entire corpus to check this*

Q12: RATE YOUR FAVORITE SHOWS AND MOVIES

- *ALL:* Only one app implements the feature.
- *GOO:* All of the apps implement the feature
- *EXE:* Onlye two apps implement the feature, there is one app that allows you to rate songs. The rest are not related *According to the observation we can say the following; first, in GOO we have much more apps than in our data set. Second, only few of the apps in our dataset are related with this query. we performed a search in our entire corpus to check this*

Q14: PAY BILLS AND CREDIT CARDS

- *ALL:* Base on descriptions only one app imlement the feature. There is other app that is related with money but in its description has a sentence with keywords related with the query, however it does not implement the query.
- *GOO:* Base on descriptions and screenshots apps from results implement the feature.
- *EXE:* Base on descriptions only one app imlement the feature. There is other app that is related with money but in its description has a sentence with keywords related with the query, however it does not implement the query. There is another app that is related with the query but does not implement its functionality. *According to the observation we can say the following; first, in GOO we have much more apps than in our data set. Second, only few of the apps in our dataset are related with this query. we performed a search in our entire corpus to check this*

Q16: SHOW YOU ALL OF YOUR FITNESS DATA

- *ALL:* None of the apps implement the feature.
- *GOO:* All of the apps are related to the query, how ever only one implements the query.
- *EXE:* None of the apps implement the feature.

According to the observation we can say the following; first, in GOO they have much more apps than in our data set. Second, some queries are domain specific. Finally, none of the apps in our dataset are related with this query. we performed a search in our entire corpus to check this

Q19: SHARE SPECIFIC LOCATIONS

- *ALL:* Apps from the results implement this feature base on the descriptions.
- *GOO:* Apps from the results implement this feature base on the descriptions.
- *EXE:* Only one application implements the feature. There rest of the apps contains the word share.

According to the observation we can say the following; first, in `GOO and ALL` we got good results. Second, `EXE` did not perform very well in this query, it retrieved apps that contain the word share.

Q21: CUSTOMIZABLE WIDGETS

In this query `ALL` obtained the best results, because in the case of `EXE` using API calls is not enough because of the type of Android application. The only way to know the type of application is extracting information from `Manifest.xml`

Q24: RECORD A NEW AUDIO CLIP TO EDIT

- *ALL:* Based on the screenshots and descriptions 4 of the five apps implement this feature.
- *GOO:* Based on the screenshots and descriptions only 3 of the 5 apps implement this feature. The rest of the apps contains the word record but in other contexts.
- *EXE:* Based on the screenshots and descriptions only 3 of the 5 apps implement this feature. The rest of the apps contains the word record but in other contexts.

According to the observation we can say the following; first, in overall most of the results are implements the feature. Second, some queries are domain specific and contains words that are ambiguous. In `GOO` and `EXE` we found that other apps contains the word record but in the context of record information in general

Q27: ALARM ACCELEROMETER

- *ALL:* Two of the apps are related with the query, however none of them implement the query.
- *GOO:* Based on the screenshots and descriptions 4 of the 5 apps implement the query. However all these 4 apps contain the words in the query on the descriptions.
- *EXE:* Based on descriptions and screenshots only 2 apps are realted with the query, however none of the apps implement the query.

According to the observation we can say the following; first, `GOO` *has a larger number of apps. Second, we look for apps that contains the word alarm in the description and inside the app use the accelerometer and only 3 from all the corpus achieve this criterion.*

Q29: MEASURE TEMPERATURE

This query is very general because some retrieved applications give the temperature of the phone and some others give the temperature of the place where the user is located.

Q30: TAKE A PICTURE

- *ALL:* Based on screenshots and description only 2 applications implement the query.
- *GOO:* Based on screenshots and descriptions only 3 applications implement the query.
- *EXE:* Based on the screenshots an descriptions 4 of the applications implement the query

According to the observation we can say the following; first, `EXE` *retrieved better results than the other search engines. Second, we look for apps that contains the* `intent android.permission.CAMERA` *in the app and we found that 56 apps in our corpus use this intent. However, it does not mean that the app take pictures, because the camera can be also used to record videos.*

Q31: WRITE INTO THE DISK

This query is very general because the apps that has the `permission android.permission.WRITE_EXTERNAL_STORAGE` *not necessarily used the permission. This is one of the easiest way to check if the app implement this query but it is still difficult to be sure about this.*

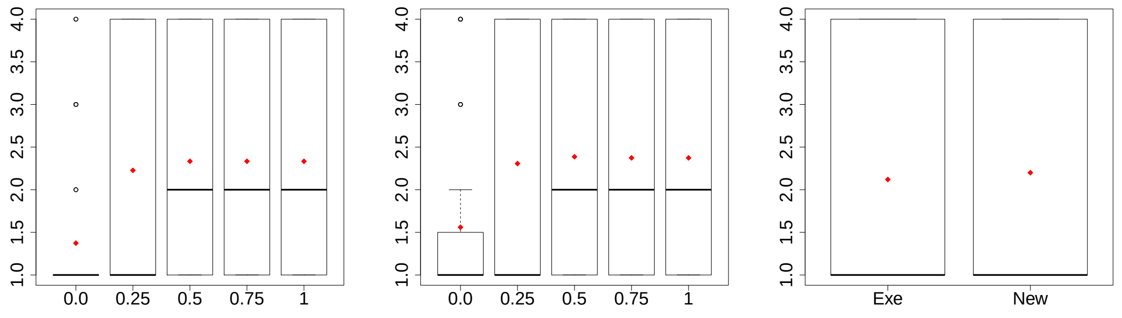
Additional Study

We decided to perform an additional study with two senior developers in order to see the impact of alpha value in the combined ranking model. We evaluated Exemplar and a new version of our approach. This approach includes the combined ranking model and the use of information of `Permissions`, `Sensors` and `Intents`.

For the implementation of our new approach (NEW) we treat each permission, sensor, and intent as an API call. Then, we rank all the API calls, permissions, sensors, and intents based on the query and next we compute the score per application. Finally, we calculate the total score combining description and the extended API call score using an alpha to give a weight to each score.

Alpha Values We evaluated results with five alpha values from 0.0 to 1 every 0.25 for the weight of description score. After an alpha value of 0.5 for description score there is not difference on the results. Moreover we run Mann-Whitney U test to evaluate the difference in the confidence level to a four-level Likert scale. We did not find a statistically significant difference at $p \leq 0.05$ between EXE and NEW. However, on average NEW is better. In Figure **5-3** we show the boxplots for different values of alpha for EXE and NEW.

During the evaluation of the results the senior developers agree in that ranking of apps are affected by the length of the query. Due to that all the words on the query must be contained into the bag of words we created per each application. In addition some queries did not retrieve any related results, as aforementioned in the observations.



(a) Confidence Level Exemplar (b) Confidence Level Our Exemplar extension (c) Confidence Level Summary of EXE and NEW

Figure **5-3**: Alpha values

In summary we presented an in-depth analysis for each query to find possibles causes of the results obtained in the survey. The main reasons are the following; first, some of the queries are domain specific and related to functionalities in which our dataset does not contain apps related. Second, some of the queries are ambiguous and short, which in our additional study has shown to have better results according to the two experts. Moreover we found couple of cases where some terms in the query can have multiple meanings affecting the perception of the user at evaluating results.

5.4 Research Questions

5.4.1 RQ1

What kind of information extracted from byte-codes could be used to improve the results of textual-based application code search engines? In the state of the art Exemplar[20] used

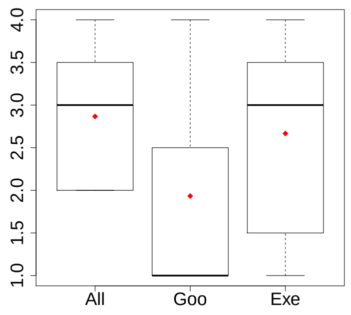
`API Calls` and shows that it can be used to improve the results on search application. In our approach we augmented the description of the application adding Intents, Sensors, Permissions, and API Calls descriptions to include valuable information that can be taken into account in the search. Based on the results this information can be used, however it did not improve or decreased the effectiveness of the results.

5.4.2 RQ2

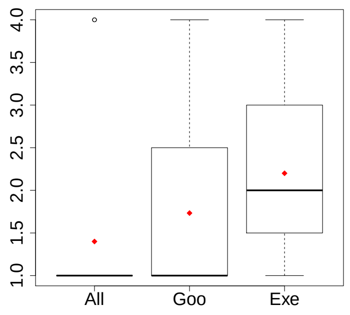
What ranking models can be used to build an Android apps search engine when combining textual information and information extracted from byte-codes? According to the state of the art there exists different models such as combined ranking-model-based, VSM based, and topic model based. Exemplar[20] is one of the approaches, which uses a combined ranking model for each app attribute (i.e. app description, `API call`) that ranks the apps and combines all the scores for each app. In our approach we proposed an augmented model that generates a document with all descriptions for the attributes we used (i.e. Intents, Sensors, Permissions, API Calls) and we used VSM to generate the ranking model of our search engine.

5.4.3 RQ3

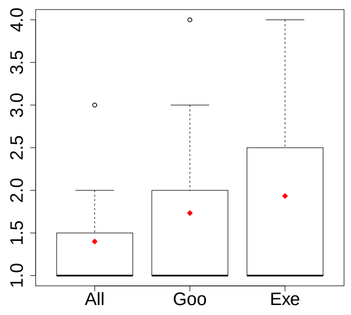
Will we retrieve better results when combining textual information and information extracted from byte-codes? In our approach we augmented the descriptions for each app and we added also descriptions for Intents, Sensors, and Permissions. However, according to the results of the study we did not find any significant difference between the results obtained in ALL, EXE, GOO for the variables confidence level, precision and NDCG. In addition we presented and in-depth analysis in Section 5.3.4 that gives some insights and possible explanations about the results. In the next Section 6 we present the conclusions of our work.



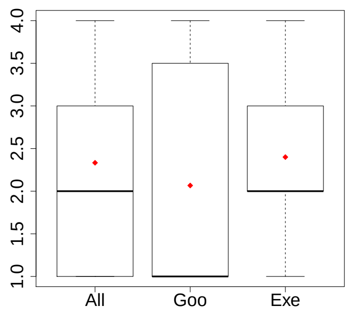
(a) Q1



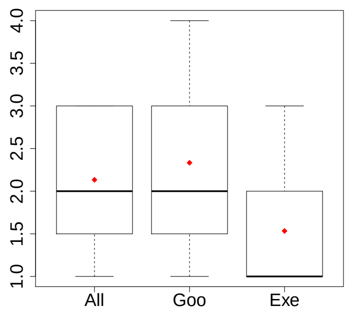
(b) Q2



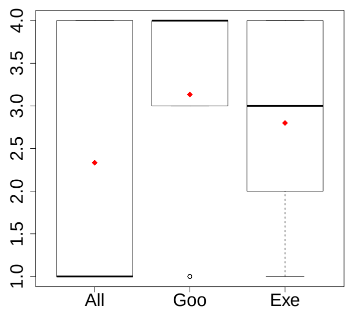
(c) Q4



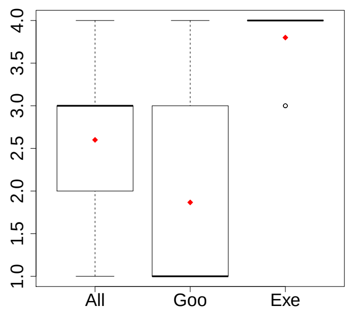
(d) Q7



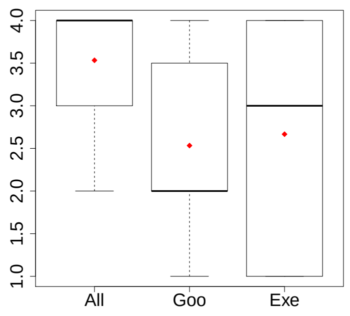
(e) Q8



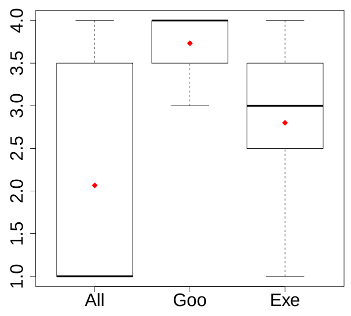
(f) Q12



(g) Q14

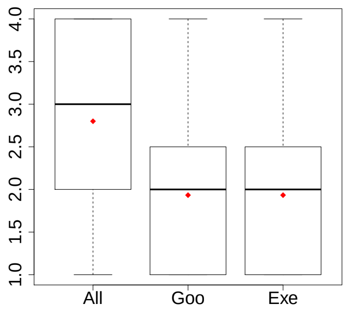


(h) Q16

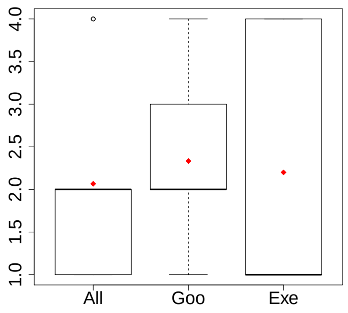


(i) Q19

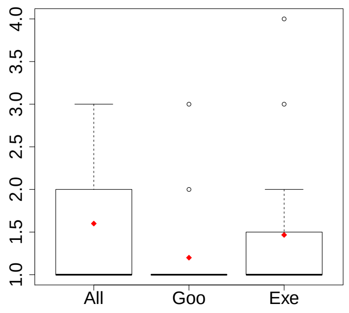
Figure **5-4**: Confidence level obtained from the first 9 queries. The red point represents the average



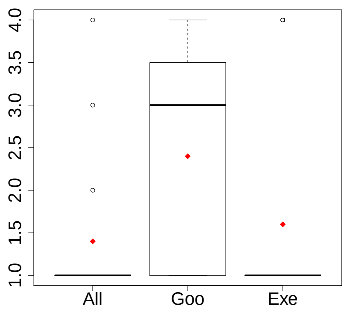
(a) Q21



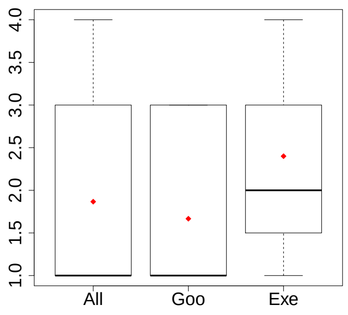
(b) Q24



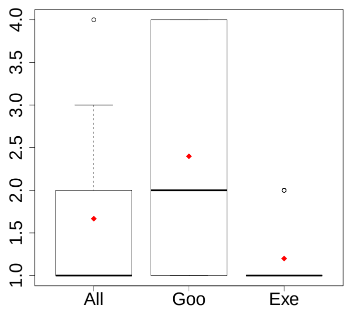
(c) Q27



(d) Q29



(e) Q30



(f) Q31

Figure **5-5**: Confidence level obtained from 15 queries

6 Conclusions and Future Work

This thesis presents an approach for searching Android applications based on descriptions of `Sensors`, `Intents`, `Permissions`, and `API Calls`. Our approach extract this information using decompilers to be able to know the specific internal information that each application implements.

We proposed an approach that augments the description of the apps with the descriptions of the attributes aforementioned. We conducted an evaluative survey in which 9 mobile developers evaluated the effectiveness of three different search engines. The dependent variables that we have are confidence level, precision, and NDCG that helps to evaluate the results of our approach. Finally as a result we obtained that our approach does not increase the effectiveness of the results compare to the others search engines.

Furthermore, we presented an in-depth analysis for each query to find possibles causes of the results obtained in the survey. The observations that we have are the following; first, some of the queries are domain specific, so in those cases our corpus of apps is too small to have a variety of apps that satisfies these kind of queries. Second, some of the queries are ambiguous because we found couple of cases where some terms in the query can have two different meanings, so the results of these queries can be affected because of the perception of the participants according to the query.

Moreover in the additional study we found out that the length of the query can affects the results, in the case of the ones that are short and general. Finally, in some cases we saw that in apps descriptions some of the terms of the query appears but those words are used in a different context, like promoting an app, giving suggestions or adding extra information not related with the application.

For future work we plan to include a dynamic analysis. So, we want to add traces collected from the execution of applications, and present to an user developer snippets about how one feature can be implemented using the source code of open source applications.

Bibliography

[1] Al-Maskari, Azzah ; Sanderson, Mark ; Clough, Paul: The relationship between IR effectiveness measures and user satisfaction. In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '07*. New York, New York, USA : ACM Press, Juli 2007. – ISBN 9781595935977, S. 773

[2] android-apktool: A tool for reverse engineering Android apk files. http://code.google.com/p/android-apktool/

[3] Bläsing, T ; Batyuk, Leonid ; Schmidt, Aubrey-derrick A.-D. ; Camtepe, Seyit A. ; Albayrak, Sahin ; Bl, Thomas ; Universit, Technische: An android application sandbox system for suspicious software detection. In: *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010*. Technische Universität Berlin, DAI-Labor, Germany, 2010. – ISBN 9781424493562, S. 55–62

[4] Bajracharya, Sushil ; Ossher, Joel: Sourcerer: An internet-scale software repository. *2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation* (2009), S. 1–4. ISBN 978–1–4244–3740–5

[5] Desnos, Anthony: Android: Static Analysis Using Similarity Distance. In: *2012 45th Hawaii International Conference on System Sciences*, IEEE, Januar 2012. – ISBN 1530–1605 VO –, S. 5394–5403

[6] dex2jar: Tools to work with android .dex and java .class files. http://code.google.com/p/android-apktool/

[7] Davies, J ; German, D M. ; Godfrey, M W. ; Hindle, A: Software Bertillonage Determining the Provenance of Software Development Artifacts. *Empirical Software Engineering* (2012), S. to appear

[8] Di Cerbo, Francesco ; Girardello, Andrea ; Michahelles, Florian ; Voronkova, Svetlana: Detection of malicious applications on android OS. In: *4th International Workshop on Computational Forensics, IWCF 2010, November 11, 2010 - November 12, 2010* Bd. 6540 LNCS. Bd. 6540 LNCS. Center for Applied Software Engineering, Free University of Bolzano-Bozen, Bolzano-Bozen, Italy : Springer Verlag, 2011. – ISBN 03029743, S. 138–149

[9] Gibler, C ; Crussell, J ; Erickson, J ; Chen, H. AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale. 2012. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)

[10] Google. Google Play. http://play.google.com

[11] Henninger, Scott: Information access tools for software reuse. *Journal of Systems and Software* 30 (1995), September, Nr. 3, S. 231–247. – ISSN 0164–1212

[12] Hsu, Sheng-Kuei ; Lin, Shi-Jen: MACs: Mining API code snippets for code reuse. *Expert Systems with Applications* 38 (2011), Juni, Nr. 6, S. 7291–7301. – ISSN 09574174

[13] Holmes, R. ; Murphy, G.C. C.: Using structural context to recommend source code examples. In: *27th IEEE/ACM International Conference on Software Engineering (ICSE'05)*. St. Louis, MO, USA : IEEe, 2005. – ISBN 1–59593–963–2, S. 117–125

[14] Han, Dan ; Zhang, Chenlei ; Fan, Xiaochao ; Hindle, Abram ; Wong, Kenny ; Stroulia, Eleni: Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs. In: *19th Working Conference on Reverse Engineering (WCRE'12)*, IEEE, 2012. – ISBN 978–0–7695–4891–3, S. 83–92

[15] jclassinfo: Extracts information from .class files. http://jclassinfo.sourceforge.net/

[16] Järvelin, Kalervo ; Kekäläinen, Jaana: Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20 (2002), Nr. 4, S. 422–446. – ISSN 10468188

[17] Jiang, Di ; Vosecky, Jan ; Leung, K.W.-T. Kenneth Wai-ting ; Ng, Wilfred: Panorama : A Semantic-Aware Application Search Framework. In: *ACM International Conference Proceeding Series*, 2013. – ISBN 9781450315975, S. 371–382

[18] Linares-Vásquez, Mario ; Holtzhauer, Andrew ; Bernal-Cárdenas, Carlos ; Poshyvanyk, Denys: Revisiting Android Reuse Studies in the Context of Code Obfuscation and Library Usages. In: *11th IEEE Working Conference on Mining Software Repositories (MSR'14)*. Hyderabad, India : ACM Press, 2014. – ISBN 9781450328630, S. to appear 10 pages

[19] McMillan, C ; Grechanik, M ; Poshyvanyk, D ; Xie, Q ; Fu, C: Portfolio: Finding Relevant Functions And Their Usages. In: *33rd IEEE/ACM International Conference on Software Engineering (ICSE'11)*. Honolulu, Hawaii, USA, 2011. – ISBN 9781450304450, S. 111–120

[20] McMillan, Collin ; Grechanik, Mark ; Poshyvanyk, Denys ; Fu, Chen ; Xie, Qing: Exemplar: A Source Code Search Engine for Finding Highly Relevant Applications. *IEEE Transactions on Software Engineering (TSE)* 38 (2012), September, Nr. 5, S. 1069–1087. – ISSN 0098–5589

[21] Michail, A: Data Mining Library Reuse Patterns Using Generalized Association Rules. In: *22nd International Conference on Software Engineering (ICSE'00)*. Limerick, Ireland : IEEE Computer Society: Los Alamitos CA, 2000, S. 167–176

[22] Mojica Ruiz, Israel J. ; Nagappan, Meiyappan ; Adams, Bram ; Hassan, Ahmed E.: Understanding reuse in the Android Market. In: *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, IEEE, Juni 2012. – ISBN 978–1–4673–1216–5, S. 113–122

[23] Manning, Christopher D. ; Raghavan, Prabhakar ; Schütze, Hinrich: *Introduction to Information Retrieval*. Bd. 1. 2008 496 Seiten. – ISBN 0521865719

[24] Mandelin, D ; Xu, L ; Bodík, R ; Kimelman, D: Jungloid mining: helping to navigate the API jungle. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'05)*, 2005, S. 48–61

[25] Porter Stemmer. http://tartarus.org/˜martin/PorterStemmer

[26] Sahavechaphan, N ; Claypool, K: XSnippet: mining for sample code. In: *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'06)*, 2006, S. 413–430

[27] Shabtai, A ; Fledel, Y ; Elovici, Y: Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. In: *2010 International Conference on Computational Intelligence and Security (CIS)*, IEEE, 2010, S. 329–333

[28] Sourceforge: Open source repository. http://sourceforge.net/

[29] The Apache Software Foundation. Lucene: Java search engine library. 2008. http://lucene.apache.org

[30] Thummalapenta, S ; Xie, T: SpotWeb: Detecting Framework Hotspots and Coldspots via Mining Open Source Code on the Web. In: *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. L'Aquila, Italy, 2008

[31] Ye, Y ; Fischer, Gerhard: Supporting Reuse by Delivering Task-Relevant and Personalized Information. In: *IEEE/ACM International Conference on Software Engineering (ICSE'02)*. Orlando, FL, 2002, S. 513–523

[32] Zhong, H ; Xie, T ; Zhang, L ; Pei, J ; Mei, H: MAPO: Mining and Recommending API Usage Patterns. In: *23rd European Conference on Object-Oriented Programming (ECOOP'09)*. Genova, Italy, 2009, S. 318–343