University of Klagenfurt
Software Engineering Research Group
Technical Report Series

Analyzing the Relationships Between Android API Classes and their References on Stack Overflow

Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger

Report AAU-SERG-2017-002

AAU-SERG-2017-002

Published, produced and distributed by:

Software Engineering Research Group
Institute for Informatics Systems
Faculty of Technical Sciences
University of Klagenfurt
Universitaeatsstrasse 65-67
9020 Klagenfurt
Austria

Software Engineering Research Group Technical Reports:
http://serg.aau.at

For more information about the Software Engineering Research Group:
http://serg.aau.at/

Note:

Analyzing the Relationships between Android API Classes and their References on Stack Overflow

Stefanie Beyer*,[1] Christian Macho,[1] Massimiliano Di Penta,[2] and Martin Pinzger[1]

[1] *University of Klagenfurt, Austria*
[2] *University of Sannio, Italy*

Correspondence: *Stefanie Beyer, Email: stefanie.beyer@aau.at

Summary

Developers of mobile applications often face problems related to the usage of mobile-specific APIs. Those problems are frequently discussed on question and answer sites, such as Stack Overflow. Our conjecture is that several of these problems can be related to certain properties of the API classes, such as their code quality. To investigate this conjecture, we analyze various properties of the Android API classes, including code metrics, code smells, and code changes, and investigate their relationship to the amount and density of references of API classes on Stack Overflow. Our goal is to find out whether certain properties of API classes are related to more references in posts. For our study, we considered 945 Android API classes referenced in 726,734 Stack Overflow posts. Our results show that API classes with code smells or with high values for the source code metrics Response For a Class (RFC) and Weighted Methods Count (WMC) tend to be reference-prone and reference-dense. We validated our findings with a manual investigation of 100 Stack Overflow posts. Using our findings, API developers can detect reference-prone API classes, improve them, and consequently increase the usability and success of the API classes in future releases.

Keywords: Android API, Stack Overflow, Code Metrics, Code Changes

1 Introduction and Motivation

Mobile applications (apps) differ from desktop applications in terms of code size, complexity, and team size (34). They are smaller than ordinary desktop applications, but at the same time they may also be complex, due to the usage of several third-party libraries. Furthermore, mobile applications need to deal with restricted user interfaces, dynamic connectivity, and need to handle interruptions, such as incoming calls (13). Due to these properties, Franke et al. (13) stated that mobile apps suffer from a lack of quality. In many cases, as also shown in previous studies, such a lack of quality can be due to the quality of the used APIs (4). Developers of mobile applications often face questions concerning the usage of APIs, for instance how to use API classes of mobile SDKs or web APIs (43). Those

questions are often discussed on question and answer sites, such as Stack Overflow, where Android is among the most popular topics (3).

Several studies show that frequently used Android API classes are also frequently referenced on Stack Overflow (22, 36, 51). Other studies analyzed the extent to which Android API changes trigger discussions on Stack Overflow, finding that discussions are particularly triggered when the behavior of methods has changed (26). In addition to these existing studies, we argue that also properties of the source code and a detailed analysis of the changes is necessary to explain this relationship. Furthermore, recent studies use only a single property to explain the amount of references of an API class. We argue that the amount of references and posts can not only be attributed to a single property, such as its usage or its changes, analyzed in isolation.

In this paper, we analyze several properties of the Android API classes and their relationship to the number of references and posts on Stack Overflow (denoted as `refCount` and `postCount`), and to the ratio of `refCount` to the number of usages of the API classes in Android apps (denoted as `refDensity` and introduced to avoid our results being strongly influenced by widely used classes). We investigate whether certain properties of the Android API classes correlate with the amount of discussion about certain API classes. Concerning the properties of APIs, we consider code characteristics, such as source metrics, code smells, and process characteristics, *i.e.,* number of code changes, number of commits, and types of commits (based on their messages). The main goal of our study is to increase the awareness of API developers that the quality, changes, and usage of their API classes can impact the amount of discussions and questions on these classes on Stack Overflow. Furthermore, we argue that improving the code quality and knowing the effects of changes does help in improving the usage of the API which is essential for its success. Finding out whether there is a relationship between properties of API classes and the amount of references and posts, as well as the reference density of these classes in Stack Overflow posts is addressed by our first research question:

RQ1: To what extent are code metrics, code smells, and code changes correlated with the number of references and posts, as well as with the reference density of Android API classes on Stack Overflow?

We investigated this correlation with 945 Android API classes referenced 9,105,121 times in 726,734 questions and answers downloaded from Stack Overflow. Our results are in line with those of Parnin et al. (36) and Kavaler et al. (22), who analyzed the usage of API classes and its internal documentation, as well as the results of Linares-Vásquez et al. (26), who found that changes in the Android APIs lead to more discussions on Stack Overflow. In addition to the existing studies, our results show that code metrics, such as the Weighted Methods per Class (WMC), the Coupling Between Objects (CBO), and code smells correlate positively to the number of references and posts, and reference density of API classes.

Furthermore, we argue that the relationship between properties of API classes and their references or reference-density can not only be attributed to a single property, such as its code metrics or its changes, analyzed in isolation. Instead, the relationship might be impacted by multiple properties depending on each other. This leads to our second research question:

RQ2: To what extent does each property contribute to the classification of reference-prone, post-prone, and reference-dense Android API classes?

Using three machine learning algorithms and the Mean Decrease of Gini coefficient (MDG) (5, 27), we investigated the importance of each property for estimating whether an API class is reference-prone, post-prone, or reference-dense. Applied to the data from **RQ1**, we found that Weighted Method Count (WMC), Response for a Class (RFC), Lack of Cohesion Of Methods (LCOM), as well as the number of commits contribute most to the classification of reference-prone, post-prone, and reference-dense API classes.

To qualitatively validate our findings, we manually analyzed 100 randomly selected Android-related posts from Stack Overflow. The results corroborate our quantitative findings but also reveal a number of additional properties of API classes, such as the need for comprehensive documentation and tutorials of API classes. They can lead to more discussion on Stack Overflow and therefore should be considered by API developers.

In summary, we make the following contributions:

- A detailed investigation of the correlation between various properties of Android API classes and the amount of references, posts, and references per usage on Stack Overflow;
- A detailed investigation of the impact of various properties of API classes on the classification of reference-prone, post-prone, and reference-dense classes;
- A validation of our findings through a manual inspection of 100 Android-related Stack Overflow posts;
- A data collection and analysis methodology, as well as a data set that allows the replication of our experiments on the same data set as well as on other APIs and apps.[1]

The remainder of this paper is organized as follows. In Section 2, we describe our approach to collect the data from Stack Overflow, the Android repository, and the Android app store. Section 3 presents the results of the correlation analysis addressing RQ1. Section 4 presents the prediction models to determine which API class properties impact the amount of references and posts, and the reference-density most. We then present the answer to RQ2. In Section 5, we describe our manual evaluation and we discuss implications of our findings and threats to validity in Section 6. Related work is listed in Section 7 and we draw our conclusions and indicate future work in Section 8.

[1] https://serg.aau.at/bin/view/StefanieBeyer/TheAndroidAPIOnStackOverflow

2 Data Collection Approach

In this section, we first describe our approach to collect the posts of Stack Overflow related to Android API classes and their usage in apps. Then, we present the various properties and corresponding metrics that we measured for API classes.

2.1 Linking Android API Classes to Posts and Android Apps

In the first step, we downloaded the posts of the Stack Overflow data dump from September 2016. Each post on Stack Overflow is tagged with one to maximum five tags. We define posts as Android-related, if they have at least one tag that contains the word `android`. Querying the Stack Overflow data dump, we obtained 910,762 Android-related questions and 1,288,366 answers to these questions, in total 2,199,128 posts. In the remainder of the paper, we refer to questions and answers as posts. Next, we selected 1,462,933 posts that contain at least one code snippet enclosed by the tags `<code>` and `</code>`.

2.1.1 Android API References on Stack Overflow

To link the code snippets of posts to API classes, we used the JavaBaker tool from Subramanian et al. (47). JavaBaker generates an incomplete abstract syntax tree for code snippets with a minimum size of three lines, traverses them iteratively with a depth-first search approach, considers declarations, invocations and assignments, and provides an oracle to find the fully qualified names of referenced API classes and methods. The oracle is a neo4j[2] database that consists of classes, methods, and field signatures. To populate the oracle with this information, the modified Dependency Finder[3] from Subramanian et al. (47) was used to extract types and methods of `.jar` files. We filled the oracle with data of the `android.jar` of the API version 19 which is the most wide spread version covering 25.2% of the Android devices as of December 1st 2016.[4]

Using JavaBaker, we extracted the code snippets of the 1,462,933 Android-related posts of Stack Overflow and collected the links for 871,022 posts. The links for the remaining posts are missing since the contained code snippets are smaller than three lines. A first analysis of the links showed that we found links to 2,297 API classes of the `android.jar`. These classes comprise public classes, public abstract classes, public interfaces, public abstract generic classes, and public enum types. In the following, we refer to these types of classes as API classes. 1,065 API classes are contained by the android-packages having a fully qualified name starting with `android`. The remaining API classes are from packages modified by Android developers for their purpose, such as the `java`, `javax`, `apache`, or `xml` packages. Since they are not directly related to the Android API, we excluded them from our study. Furthermore, we

[2] https://neo4j.com/product/
[3] https://bitbucket.org/rtholmes/depfind_uw
[4] https://developer.android.com/about/dashboards/index.html

excluded the classes of the `android.test` package since the focus of our study is on the implementation of Android apps and not on testing them. The classes `android.R` and `android.Manifest` were excluded as well, since they are mainly used as resource classes.[5] As a result, we obtained 1,030 Android API classes that are referenced in posts on Stack Overflow. We do not consider API classes mentioned in the body or title of posts, since our main goal is to investigate the properties of the API classes that are referenced in code snippets. The extension of our study to take also into account the classes mentioned in the text of the body, title, or tags is left for future work.

2.1.2 Usage of Android API classes in Android Apps

To obtain the usage count of API classes, we first crawled the Google Play Store and collected the ids of the top 80 free apps of each app category. We then used the app ids to download the `.apk` files[6] from the Play Store. In total, we collected 3,141 app ids out of which 214 could not be downloaded because of security reasons or restrictions of the Play Store. For the remaining 2,927 apps, we used the approach by Linares-Vásquez (26) to convert the `.apk` files to `.jar`s, and then used the tool *dex2jar*[7] to disassemble the `.jar` files. We were able to convert 2,715 `.apk` files to valid `.jars`. Finally, we used *jclassinfo*[8] to extract the dependencies from the Android app to Android API classes. In total, 945 API classes out of 1,030 are used in our set of Android apps. These classes are referenced 9,105,121 times in 726,734 Android related posts and they represent the input set for our studies.

For each API class, we counted the number of all references to the API class in the posts denoted as `refCount`, the number of distinct posts denoted as `postCount`, the total number of usages in the set of Android apps denoted as `usageCount`, and the ratio of the `refCount` to the `usageCount` denoted as `refDensity`. The rationale of `refDensity` is to avoid having results being influenced too much by widely used APIs, which are likely to be referenced a lot because they are also used a lot. Unused APIs are clearly excluded when computing the `refDensity`. Table 1 shows the most frequently referenced API classes and their corresponding `refCount`, `postCount`, `usageCount`, and `refDensity`.

Table 1 confirms the findings of Subramanian et al. (46) who found that the API classes `android.content.Intent`, `android.view.View`, `android.widget.TextView`, `android.app.Activity`, and `android.os.Bundle` are most frequently referenced in Stack Overflow posts. In contrast to the most frequently referenced API classes showing a `refDensity` between 1.2264 and 9.4558, less frequently referenced API classes, such as `android.util.SparseLongArray`, `android.app.MediaRouteButton`, and `android.app.backup.BackupDataInputStream` show the highest values for `refDensity` with more than 3,000 references per usage.

[5] https://developer.android.com/reference/android/package-summary.html
[6] http://apkleecher.com/
[7] https://sourceforge.net/projects/dex2jar/
[8] http://jclassinfo.sourceforge.net/

Table 1: refCount, postCount, usageCount and refDensity of the top 10 most frequently referenced Android API classes

Android API class	*refCount*	*postCount*	*usageCount*	*refDensity*
android.view.View	726,370	234,419	165,219	4.3964
android.app.Activity	534,087	109,746	76,753	6.9585
android.content.Intent	533,397	147,925	129,504	4.1188
android.widget.TextView	439,518	104,516	86,610	5.0747
android.util.Log	438,206	115,329	118,662	3.6929
android.os.Bundle	356,132	197,032	97,837	3.6401
android.content.Context	263,936	122,577	215,211	1.2264
android.database.Cursor	217,975	38,626	23,052	9.4558
android.view.LayoutInflater	187,516	69,477	56,276	3.3321
android.widget.ImageView	176,301	54,137	55,113	3.1989

Furthermore, we calculated the Spearman's rank correlation coefficient ρ between `refCount` and `usageCount` obtaining $\rho = 0.641$, confirming the findings of Parnin et al. (36) who found a strong positive correlation of 0.797 for referenced API classes and their usage.

2.2 Properties of Android API Classes

In the next step, we computed various metrics of the 945 API classes that cover several properties, namely the source code, the change history, and the usage of API classes in Android apps. According to the Android open source project, the Android API classes and services are located in the project `platform.framework.base`.[9] We cloned the branch `android-4.4.4_r1` having API level 19 (Kitkat) and computed the metrics for those API classes.

2.2.1 Source Code Metrics

We used the tool Understand[10] to measure different volume metrics, such as lines of code and executable statements, complexity metrics, such as cyclomatic complexity and nesting, and also object oriented code metrics, such as depth of inheritance tree and number of methods. Since many code metrics showed a high correlation to each other, we decided to use the following set of metrics for our investigations: LOC (Lines Of Code), nesting, WMC (Weighted Methods per Class), DIT (Depth of Inheritance Tree), NOC (Number Of Children), CBO (Coupling Between Objects), LCOM (Lack of Cohesion Of Methods), and RFC (Response For a Class). We chose the Chidamber and Kemerer (6) metric suite since they have been used in many previous empirical studies, such as (10, 21, 42), and because they properly capture different characteristics (size, complexity, cohesion, coupling, and inheritance) of object-oriented code.

Table 2 shows descriptive statistics of the code metrics measured for the 945 API classes. Note, the minimum value 0 for all metrics stems from an annotation class.

[9] https://android.googlesource.com/platform/frameworks/base/
[10] https://scitools.com

Table 2: Descriptive statistics of API class code metrics

	Min.	*1st Qu.*	*Median*	*Mean*	*3rd Qu.*	*Max.*
LOC	0	30	108	242.3	267	8,264
Nesting	0	0	1	1.73	3	9
WMC	0	6	19	43.92	46	1,924
DIT	0	0	0	0.66	1	5
NOC	0	0	0	3.19	1	387
CBO	0	4	8	12.43	17	166
RFC	0	21	33	119.00	66	1,317
LCOM	0	30	75	60.23	90	100

2.2.2 Code Smells

We used the inCode tool to detect code smells (or design disharmonies) in the 945 API classes. inCode supports the detection of various code smells presented by Lanza and Marinescu (23), such as god class, data class, or feature envy. Looking at the results, we found that 144 of the 945 API classes suffer from at least one code smell. 85 classes suffer from the god class smell, 36 classes contain the smell data class, 23 classes are schizophrenic classes, and 5 classes are tradition breakers. 5 API classes contain 2 smells.

The severity of a code smell indicates the severity of a smell and ranges from 1 (low severity) to 10 (high severity). inCode outputs that 21 of the 144 API classes suffer from a code smell with highest severity, namely 10. Examples of these classes are the public classes `android.view.View`, `android.app.Activity` and the public classes `Textview`, `GridView`, and `ListView` from the `android.widget` package.

2.2.3 Code Changes

We applied JGit[11] to the cloned git repository `android-4.4.4_r1` to obtain all commits and revisions of the 945 API classes. For each API class, we first counted the number of commits denoted as commitCount. Next, we obtained all the revisions for each class and used the ChangeDistiller tool by Fluri et al. (12) to extract the source code changes between all revisions of an API class. ChangeDistiller parses two subsequent versions of a Java file into two ASTs and then compares them using a tree-differencing algorithm. The output is a list of tree edit operations (insert, delete, update, and move) to transform one version of the AST into the new version of the AST. The edit operations then are mapped to 48 different types of code changes in Java classes, such as the insertion of a method, the removal of statements in a method body, or the update of an if-condition. We refer the reader to Fluri et al. (11) that introduces the taxonomy of change types used by ChangeDistiller and used in our studies.

For each API class, we counted the frequency of each change type until the version `android-4.4.4_r1`. A first glimpse on the results showed that changes of statements in method bodies (count given in brackets) represented

[11] https://www.eclipse.org/jgit/

by the change types STATEMENT_DELETE (148,040), STATEMENT_INSERT (116,043), and STATEMENT_-UPDATE (51,016) occur the most frequent in Android API classes. This was expected and confirms the results of a previous study of Giger et al. (14) with other Java open source projects, where statement changes account for more than 70% of the changes. In addition, we found that the removal of methods (denoted as REMOVED_-FUNCTIONALITY) in the API classes occurred frequently, namely 33,143 times. Concerning the API classes, `android.webkit.WebView`, `android.view.View`, and `android.widget.TextView` exhibit more than 1,700 commits and more than 27,000 code changes.

2.2.4 Commit Categories

In addition to the frequency of commits and the code changes, we assume that different commit categories impact the frequency of discussions on Stack Overflow. For instance, the refactoring of API classes might lead to more discussion than a bug fix. We implemented the approach of Hattori et al. (19) to categorize commits based on their commit messages into the following four categories: Management, CorrectiveEngineering, ReEngineering, and ForwardEngineering. Hattori et al. specified sets of keywords each indicating a commit category. For each class, we counted the number of matched keywords per commit category. Note, a commit can have multiple categories. The number of matched keywords indicates how strong a commit falls into each category. For instance, the class `android.webkit.WebView` has the highest number of commit messages for all four categories.

3 API Classes and Stack Overflow Posts

In this section, we analyze the correlation between the various metrics of API classes and the number of references and posts, as well as the number of references per usage of those classes on Stack Overflow. With the results of the correlation analysis we aim to answer *RQ*1.

3.1 Correlation Analysis

For the correlation analysis, we input all metrics and calculated the Spearman's rank correlation coefficient ρ with the number of references (`refCount`), the number of posts (`postCount`), and the number of references per usage in Android apps (`refDensity`). We used the Spearman's rank correlation because, after performing a Shapiro-Wilk normality test, we concluded that the metric values significantly deviate from a normal distribution (p-values < 0.001 in all cases). According to Cohen (7), we define $0.1 < |\rho| < 0.3$ as small correlation, $0.3 \leq |\rho| < 0.5$ as moderate correlation, and $|\rho| \geq 0.5$ as strong correlation.

3.1.1 Posts and Code Metrics

Table 3 lists Spearman's ρ between the code metrics and `refCount`, `postCount`, and `refDensity` of API classes. The results show a strong correlation for `refCount` and `postCount` with the code metric WMC. Furthermore, they show a a moderate correlation for `refCount` and `postCount`, and a small correlation for `refDensity` with the code metrics CBO, RFC, LCOM, LOC, and Nesting. The metric NOC shows a small positive correlation with `refCount` and `postCount` but no correlation with `refDensity`. The metric DIT does not show any correlation with `refCount` and `refDensity` of API classes.

Table 3: Spearman's ρ between code metrics and refCount, postCount, and refDensity of API classes

	refCount	postCount	refDensity
LOC	0.4650	0.4358	0.2415
Nesting	0.3572	0.3341	0.2122
WMC	0.5286	0.5014	0.2656
DIT	0.0145	0.0298	0.0661
NOC	0.1951	0.2088	-0.0228
CBO	0.4827	0.4593	0.2679
RFC	0.4253	0.4130	0.2455
LCOM	0.4015	0.3751	0.2174

3.1.2 Posts and Code Smells

We investigated if API classes suffering from code smells are more referenced on Stack Overflow than API classes without code smells. Figure 1 shows the box-plots comparing the distributions of `refCount`, `postCount`, and `refDensity` of API classes with and without code smells. Since the Android API contains more classes without code smells than classes with code smells, we also computed the relative `refCount`, `postCount`, and `refDensity` per API class for both categories and show the ratios. Table 4 summarizes our findings.

Table 4: Number of Android API classes with and without code smells and their refCount, postCount, and refDensity

	with smell	without smell	ratio
classes	144.00	801.00	-
refCount	4,590,749.00	5,341,425.00	-
refCount/class	31,880.20	6,668.45	4.78
median(refCount)	2,758.00	456.00	6.05
postCount	1,273,669.00	2,085,474.00	-
postCount/class	8,844.92	2,603.59	3.40
median(postCount)	1,393.00	282.00	4.94
refDensity/class	43.04	66.68	0.65
median(refDensity)	4.79	2.47	1.94

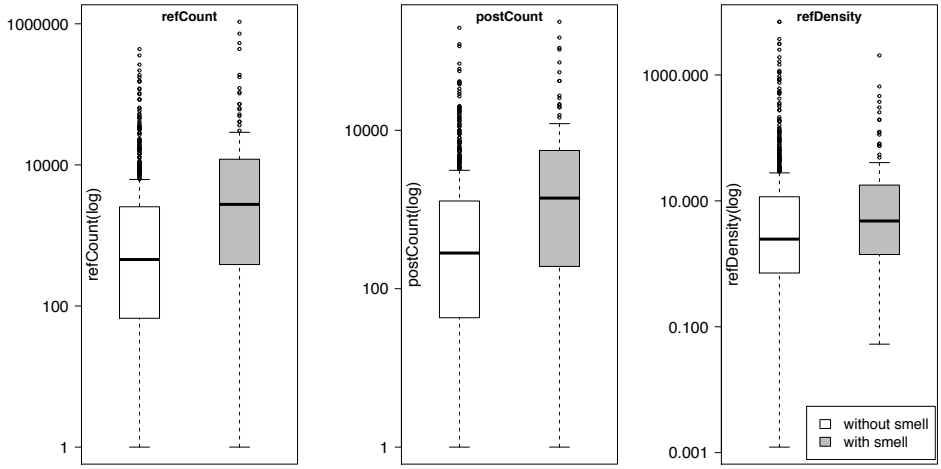


Figure 1: Distribution of `refCount`, `postCount`, and `refDensity` of API classes without (white boxplots) and with code smells (grey boxplots)

As mentioned before, inCode detected code smells in 144 out of the 945 API classes leaving 801 classes without a code smell. In sum, these 801 API classes were referenced more frequently than the 144 API classes containing a code smell. Comparing the `refCount` and `postCount` of API classes with a code smell and API classes without smell per class, we found that on average an API class with a code smell is referenced more frequently in Stack Overflow posts, namely 4.23 times more in terms of number of references and 3.40 times more in terms of number of posts. Furthermore, the median `refCount` of API classes is 6.05 times higher than of classes without smells. Similarly the median `postCount` of API classes with smells is 4.94 times higher than the median `postCount` of API classes without a smell. Regarding the `refDensity`, classes with a smell have a lower `refDensity` per class than classes without a smell, however, comparing the medians, API classes with a smell show 1.94 times higher `refDensity` than classes without a smell.

To show whether this difference is statistically significant, we applied a Mann-Whitney-U test and calculated Cliff's delta (d). According to Grissom et al. (16), d is negligible for $d<0.147$, small for $0.147\leq d<0.33$, medium for $0.33\leq d<0.47$, and large for $d\geq 0.47$. For both, `refCount` and `refDensity`, the test showed a statistically significant difference between API classes with and without code smells with a p-value $< 10^{-10}$ for `refCount` and $p < 0.002$ for `refDensity`. The value of Cliff's d is 0.3469 for `refCount`, 0.3424 for `postCount`, and 0.1629 for `refDensity` indicating that this difference is medium for `refCount` and `postCount`, and small for `refDensity`.

3.1.3 Posts and Code Changes

We calculated the Spearman's correlation between the number of commits (commitCount), the various change types extracted by ChangeDistiller, and `refCount`, `postCount`, and `refDensity` of API classes. The results are presented in Table 5.

Table 5: Spearman's ρ between commitCount, ChangeDistiller change types, and refCount, postCount, and refDensity of API classes (only change types with an $|\rho| > 0.3$ to refCount are shown)

	refCount	postCount	refDensity
commitCount	0.4709	0.4445	0.1691
ADDITIONAL_FUNCTIONALITY	0.3395	0.3157	0.2018
ALTERNATIVE_PART_DELETE	0.3549	0.3366	0.1094
COMMENT_DELETE	0.3263	0.3025	0.1211
COMMENT_INSERT	0.3248	0.3060	0.1597
CONDITION_EXPRESSION_CHANGE	0.3185	0.3013	0.1553
DOC_UPDATE	0.3457	0.3184	0.1850
REMOVED_FUNCTIONALITY	0.4587	0.4352	0.1684
REMOVED_OBJECT_STATE	0.3287	0.3094	0.0778
STATEMENT_DELETE	0.3978	0.3737	0.1503
STATEMENT_INSERT	0.3868	0.3632	0.1677
STATEMENT_PARENT_CHANGE	0.3902	0.3706	0.1417
STATEMENT_UPDATE	0.3895	0.3669	0.1658

The results show a moderate positive correlation between commitCount and `refCount`, as well as `postCount`, and a small correlation between commitCount and `refDensity`. Regarding the change types extracted by ChangeDistiller, we observe a moderate correlation between 12 out of 48 change types with `refCount` and `postCount`. The change type REMOVED_FUNCTIONALITY exhibits the strongest correlation with `refCount` (0.4622) and `postCount` (0.4352). This confirms the findings of Linares-Vásquez (26) that the deletion of public methods in the API correlates with more discussion on Stack Overflow. For `refDensity`, we observe a small correlation between 20 out of 48 change types, whereas the strongest correlations were observed for the change types ADDITIONAL_FUNCTIONALITY and DOC_UPDATE.

3.1.4 Posts and Commit Categories

Using Spearman's correlation, we investigated the correlation between the four commit categories and the `refCount`, `postCount`, and `refDensity` of API classes. Table 6 shows the results.

According to the results, all four commit categories show a moderate positive correlation to the `refCount` and `postCount`, and a small positive correlation to the `refDensity` of API classes. The highest correlation to `refCount` is observed for the commit category CorrectiveEngineering, and `refDensity` shows the highest correlation to the commit category Management.

Table 6: Spearman's ρ of the four commit categories and refCount, postCount, and refDensity

	refCount	postCount	refDensity
ForwardEngineering	0.4073	0.3847	0.1744
ReEngineering	0.4205	0.3950	0.1974
CorrectiveEngineering	0.4532	0.4274	0.1873
Management	0.4388	0.4133	0.2060

3.2 Summary and Answer to RQ1

Summarizing the findings, we can answer our first research question *RQ*1. We found the code metrics WMC, CBO, LOC, RFC, and LCOM, as well as Nesting are positively correlated with the `refCount`, `postCount`, and `refDensity` of API classes. Concerning code smells, we found that API classes with a code smell are significantly more referenced in posts on Stack Overflow and show a higher reference density than API classes without a code smell. We also found that API classes that change frequently are more often referenced in posts on Stack Overflow and show a higher reference density. This holds in particular for the change types that denote the addition and removal of an API methods, and updates of the Java documentation. This finding is further supported by the positive correlation of the four commit categories whereby CorrectiveEngineering showed the strongest correlation with `refCount` and `postCount`, and the commit category Management with `refDensity`. The results also show a difference in the strength of the correlation: while we found several moderate correlations to `refCount` and `postCount`, we only found small correlations to `refDensity`.

4 Reference-Prone and Reference-Dense API Classes

In this section, we investigate whether the relationship of API classes and their references or reference-density on Stack Overflow is impacted by multiple properties considering also correlations between them.

4.1 Experimental Setup

We used the machine learning algorithms Random Forests (RF) (5), Support Vector Machines (SVM) (8), and binary Logistic Regression (LR) (9) to compute models for classifying Android API classes into reference-prone (*ref-prone*) and not reference-prone. We performed the same experiment also for classifying post-prone (*p-prone*) and not post-prone API classes, and reference-dense (*ref-dense*) and not reference-dense API classes. We used the default-settings for the algorithms provided by the R (39) packages for RF (24), SVM (33), and LR (50). The API properties represent the independent variables while the label *ref-prone* (and respectively *p-prone* and *ref-dense*) represents the dependent variable. For the classification, we labeled an API class as *ref-prone* (and respectively *p-prone* and *ref-dense* using

the `postCount` and `refDensity`) as follows:

$$APIclass = \begin{cases} not\ ref\text{-}prone & : \texttt{refCount} \leq median \\ ref\text{-}prone & : \texttt{refCount} > median \end{cases}$$

Using the *median* as cut-off point, we assure that we have two balanced datasets for computing the models. In addition, we computed the models for a tertiary classification. Since these models showed similar results, we only report the results using the binary classification and provide the other results in the supplementary material.

Furthermore, we addressed the issue that highly correlated independent variables ($\rho \geq 0.8$) can influence the prediction models. We applied hierarchical clustering using the squared Spearman's correlation to all independent variables. Then, we cut the resulting tree at the height of $\rho^2 = 0.64$ to obtain clusters of highly correlated variables having an $\rho > 0.8$. The first cluster consists of the variables commitCount, REMOVED_FUNCTIONALITY, and the commit categories ForwardEngineering, ReEngineering, CorrectiveEngineering, and Management. The second cluster contains the code metrics LOC, WMC, and CBO. The third cluster consists of the change types STATEMENT_DELETE, STATEMENT_INSERT, and STATEMENT_UPDATE. From each cluster, we selected the one variable that correlates most to the target variable *ref-prone*, *p-prone*, or *ref-dense*, resulting in a set of 52 variables. For *ref-prone* and *p-prone*, we selected the variables commitCount, WMC, and STATEMENT_DELETE. For *ref-dense*, we selected the variables Management, CBO, and STATEMENT_INSERT.

Concerning the validity of our results, we applied ten times a 10-fold cross-validation. In each of the ten runs the data is randomly partitioned into 10 folds of equal size. Each fold is once used as test set and the remaining nine folds are used for training. This is repeated 10 times. To measure the performance of the prediction models, we computed the following metrics for the reference-proneness, post-proneness, and reference-denseness of API classes:

- *Accuracy (acc)* is the ratio of API classes correctly classified as *ref-prone* (*p-prone*, *ref-dense*) and not *ref-prone* (*p-prone*, *ref-dense*) with respect to all classified classes. Values range from 0 (low accuracy) to 1 (high accuracy).
- *Precision (prec)* is the ratio of correctly classified *ref-prone* (*p-prone*, *ref-dense*) classes with respect to all classes classified as *ref-prone* (*p-prone*, *ref-dense*). Values range from 0 (low precision) to 1 (high precision).
- *Recall (rec)* is the ratio of correctly classified *ref-prone* (*p-prone*, *ref-dense*) classes with respect to the classes that are actually observed as *ref-prone* (*p-prone*, *ref-dense*). Values range from 0 (low recall) to 1 (high recall).
- *F-measure (f)* is the harmonic mean of precision and recall. Values range from 0 (low performance) to 1 (high performance).

– *Area under ROC-Curve (AUC)* measures the ability to classify API classes correctly into *ref-prone* (*p-prone*, *ref-dense*) using various discrimination thresholds. An AUC value of 1 denotes the best performance, and 0.5 indicates that the performance equals a random classifier (i.e., guessing).

Note, our experiments aim to analyze the importance of each property of API classes instead of finding the best performing model to classify reference-prone, post-prone, and reference-dense API classes.

To determine and show the importance of each independent variable, we used the Mean Decrease in the Gini coefficient (MDG), a measure of how much each independent variable contributes to the homogeneity of the nodes and leaves in the Random Forest (5, 27). Finally, we perform the Scott-Knott test (45) to rank variables according to their importance, and to compute the effect-size of the difference between the variables' importance. Specifically, for this purpose we rely on the Scott-Knott Effect Size Difference (ESD) R package (48, 49). According to (49), values < 0.2 denote a difference that is considered to be negligible, < 0.5 small, < 0.8 medium, otherwise the difference is considered to be large.

4.2 Results for Reference-Prone API classes

Figure 2 shows the performance of 10x10 runs to classify API classes into *ref-prone* and not-*ref-prone* computed with the Random Forest (RF), Support Vector Machine (SVM), and binary Logistic Regression (LR) taken into account 52 API class metrics.

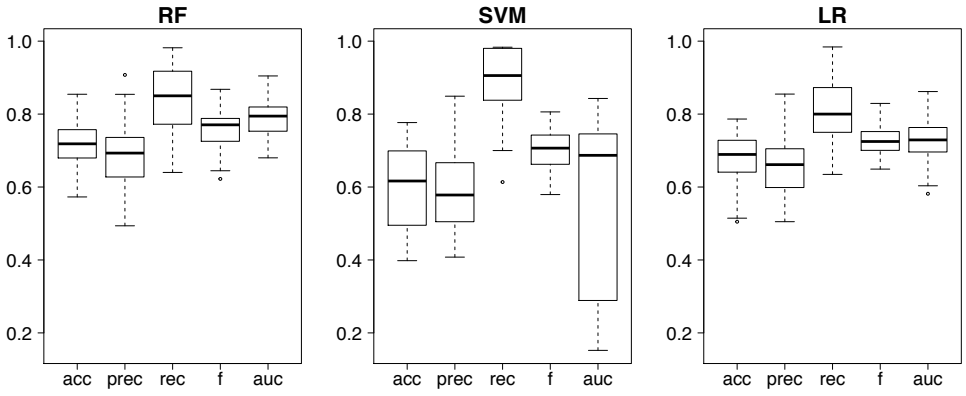


Figure 2: Accuracy, precision, recall, f-measure, and AUC of the *ref-prone* classification models obtained with RF, SVM, and LR

For the binary classification, the RF performs best in terms of accuracy, precision, f-measure, and AUC. The median values of RF models are: 0.72 (accuracy), 0.69 (precision), 0.85 (recall), 0.77 (f-measure), and 0.79 (AUC).

The SVM models outperform RF and LR only in the recall with a median value of 0.90. Furthermore, looking at the shape of the box-plots the plots for the RF models show a more stable performance across the 10x10 runs. Therefore, we selected the Random Forest algorithm for the analysis of the importance of the independent variables.

Figure 3 shows the box-plots of the MDG of the models to classify *ref-prone* API classes. The box-plots of the tertiary classification models are similar and are available in our supplementary material.

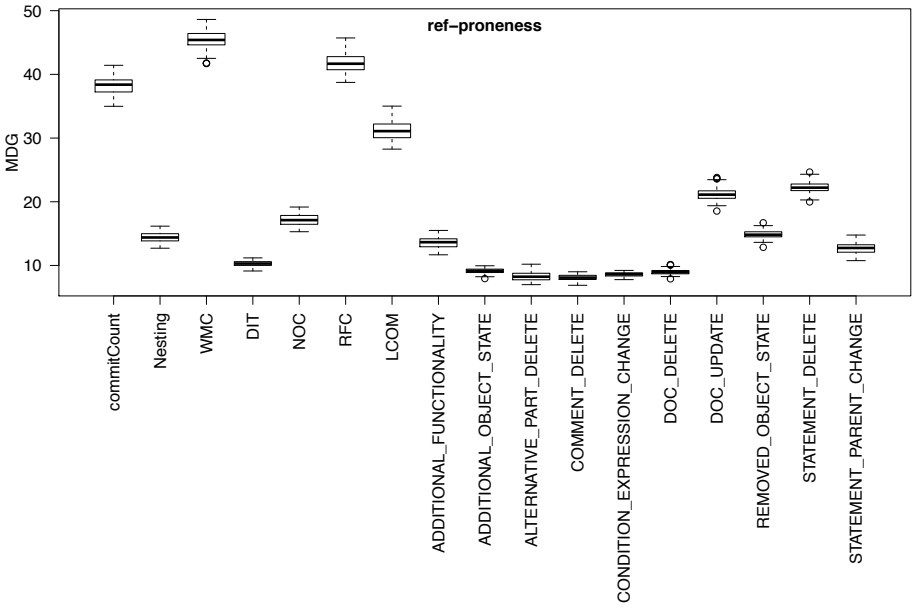


Figure 3: MDG of the independent variables to classify API classes into *ref-prone* and not *ref-prone* having an MDG ≥ 5

The box-plots show that the most predominant variables for classifying *ref-prone* API classes are the code metrics WMC and RFC, the commitCount, and LCOM with median MDGs of 45.40, 41.67, 38.39, and 31,14 respectively. Other relevant variables for the classification of *ref-prone* API classes with an MDG > 20 are STATEMENT_DELETE and DOC_UPDATE. These results are fully confirmed by the Scott-Knott ESD test, which for all variables shown in Figure 3 reports a large effect size difference, except for REMOVED_OBJECT_STATE and Nesting, for which it reports a medium effect size difference. Therefore, the variables are never clustered together. For the sake of space, Table 7 reports the results of the test for the top-10 ranked variables only.

Table 7: Classification of API classes into *ref-prone* and not *ref-prone*: results of the Scott-Knott ESD test (only the top-10 ranked variables are reported).

	RFC	commitCount	LCOM	STATEMENT_DELETE	DOC_UPDATE	NOC	REMOVED_OBJECT_STATE	Nesting	ADDITIONAL_FUNCTIONALITY
WMC	2.67	4.67	10.14	17.06	17.76	22.34	25.31	24.20	25.87
RFC		2.31	8.11	15.71	16.48	21.70	25.23	23.76	25.78
commitCount			5.21	11.98	12.77	17.13	19.95	19.09	20.63
LCOM				7.20	8.16	12.81	15.96	15.08	16.79
STATEMENT_DELETE					1.21	5.66	8.77	8.32	9.92
DOC_UPDATE						4.16	7.01	6.77	8.20
NOC							3.00	3.12	4.58
REMOVED_OBJECT_STATE								0.59	2.01
Nesting									1.14

4.3 Results for Post-Prone API classes

As a next step, we performed the previous experiment with *p-prone* API classes. The boxplots of the experiments are shown in Figure 4. Again, the Random Forest algorithm shows the best performance in terms of accuracy (0.73), precision (0.69), f-measure(0.76), and AUC (0.79). As before, the SVM outperformed RF only in terms of recall (0.94 vs. 0.84).

Figure 5 shows the boxplots of the MDG values of the independent variables for predicting *p-prone* API classes, showing that the variables WMC, RFC, commitCount, and LCOM with MDGs of 45.77, 41.52, 38.33, and 31.04 respectively, are the most important for the prediction of *p-prone* API classes. The results of the Scott-Knott ESD test (also in this case limited to the 10-most important variables) are shown in Table 8. The magnitude of the difference between all pairs of variables is large, except between STATEMENT_PARENT_CHANGE and REMOVED_OBJECT_STATE, for which the difference is negligible.

4.4 Results for Reference-Dense API Classes

As in the previous experiment, the Random Forest algorithm shows the best performance with a median accuracy of 0.76, precision of 0.62, recall of 0.92, f-measure of 0.74, and AUC of 0.74. As before, the SVM model outperformed

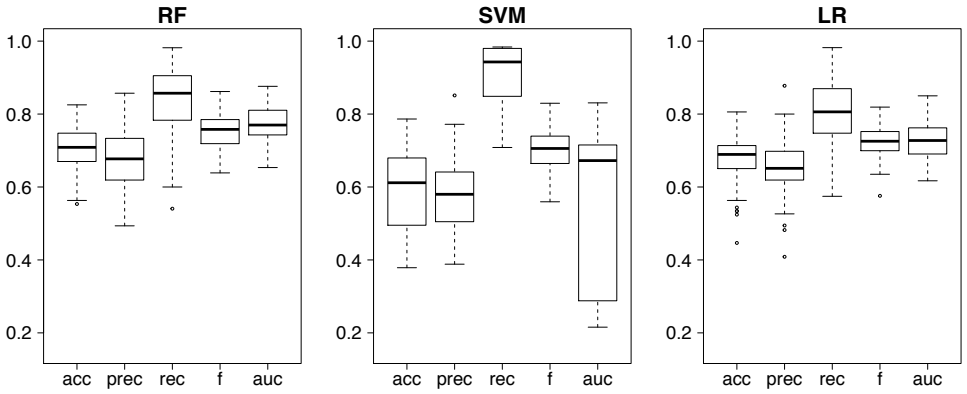


Figure 4: Accuracy, precision, recall, f-measure, and AUC of the *p-prone* classification models obtained with RF, SVM, and LR

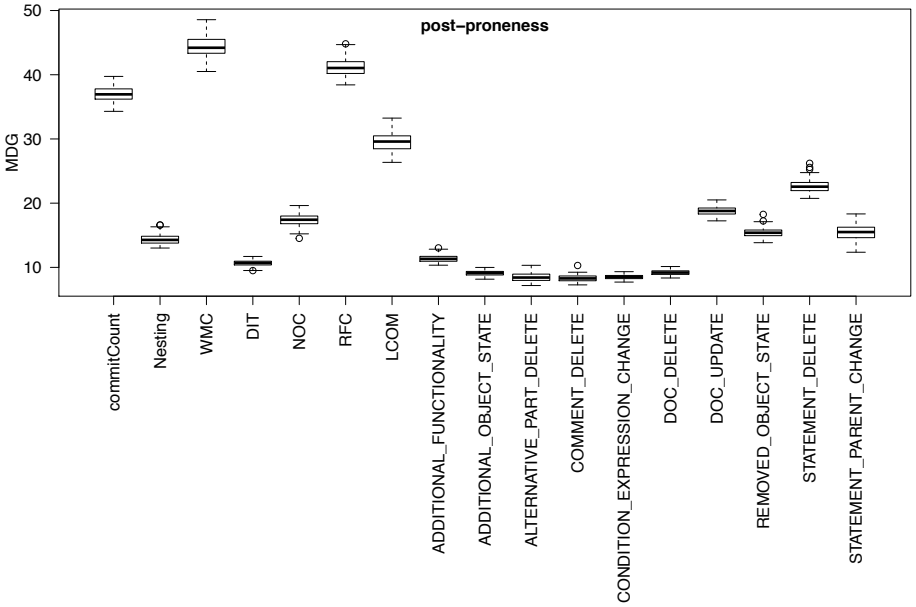


Figure 5: MDG of the independent variables to classify API classes into *p-prone* and not *p-prone* having an MDG ≥ 5

the RF only in terms of the recall with a median value of 0.96. Figure 6 shows the boxplots of the experiments. Analyzing the MDG values of the RF models, we obtained the box-plots depicted in Figure 7.

Table 8: Classification of API classes into *p-prone* and not *p-prone*: results of the Scott-Knott ESD test (only the top-10 ranked variables are reported).

	RFC	commitCount	LCOM	STATEMENT_DELETE	DOC_UPDATE	NOC	STATEMENT_PARENT_CHANGE	REMOVED_OBJECT_STATE	Nesting
WMC	2.16	5.29	10.05	16.18	21.06	20.78	21.09	23.86	24.14
RFC		3.17	8.34	14.78	20.10	19.74	20.05	23.16	23.43
commitCount			5.80	12.67	18.66	18.18	18.49	22.15	22.40
LCOM				5.52	9.77	10.21	11.06	12.88	13.44
STATEMENT_DELETE					4.34	5.30	6.56	8.17	8.95
DOC_UPDATE						1.76	3.54	4.99	6.07
NOC							1.76	2.35	3.43
STATEMENT_PARENT_CHANGE								0.12	1.19
REMOVED_OBJECT_STATE									1.42

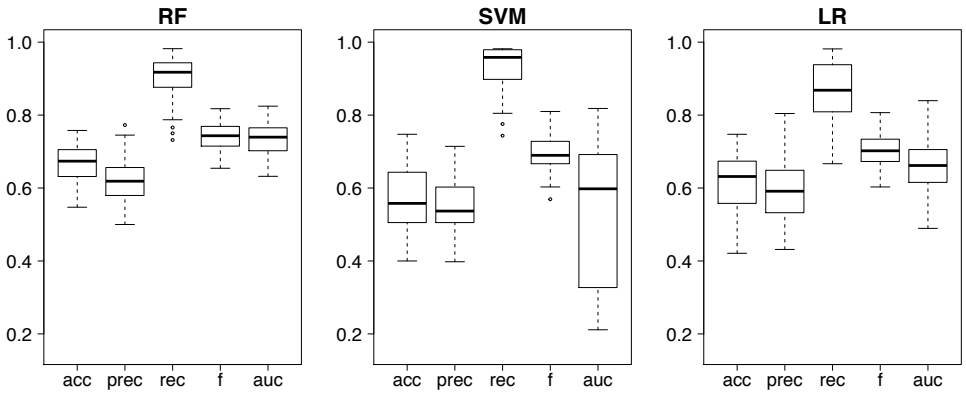


Figure 6: Accuracy, precision, recall, f-measure, and AUC of the *ref-dense* classification models obtained with RF, SVM, and LR

The most important independent variables for predicting *ref-dense* API classes are the code metrics RFC and LCOM with an MDG of 36.13 and 33.25 respectively. The box-plots also show that the commit category Management (31.67), CBO (26.93), and the change category DOC_UPDATE (19.43) are important for the classification of *ref-dense* API classes. Also in this case, the results are confirmed by the Scott-Knott ESD test, for which we

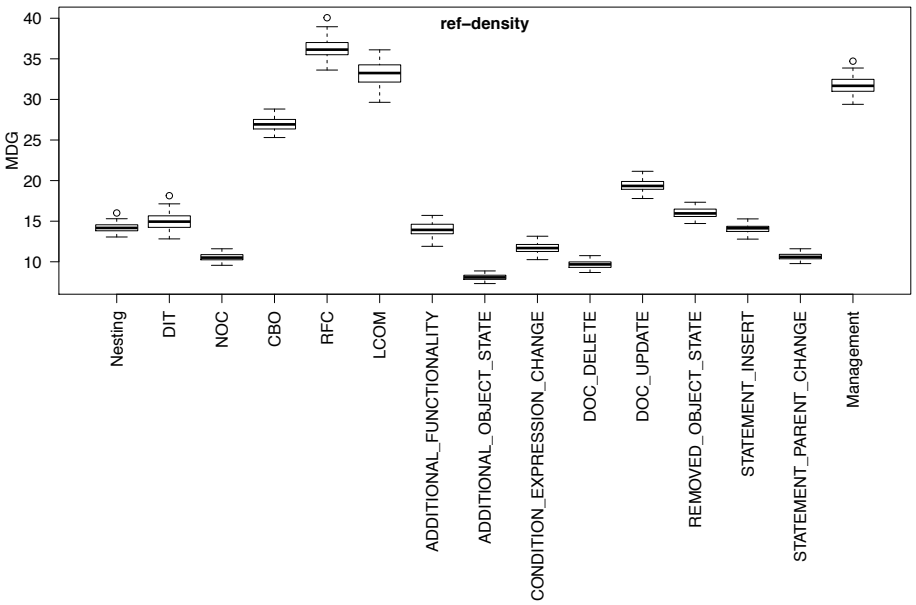


Figure 7: MDG of the independent variables to classify API classes into *ref-dense* and not *ref-dense* having an MDG ≥ 5

report distances for the top-10 variables in Table 9. The effect size is large, except for the pairs involving Nesting, STATEMENT_INSERT, and ADDITIONAL_FUNCTIONALITY, where it is always small.

4.5 Summary and Answer to RQ2

The results of our experiments show that the code metrics WMC, RFC, and LCOM, and the metric commitCount are important for classifying API classes into *ref-prone* and not *ref-prone*, as well as classifying API classes into *p-prone* and not *p-prone*. Regarding the reference-denseness, the WMC metric is replaced by the CBO metric, and commitCount is replaced by the commit category Management. For all three experiments, the change types showed to be less influential than the mentioned code and commit metrics.

5 Manual Evaluation

To validate the results of the quantitative studies, we performed a manual analysis with a random sample of Stack Overflow posts related to Android app development.

Table 9: Classification of API classes into *ref-dense* and not *ref-dense*: results of the Scott-Knott ESD test (only the top-10 ranked variables are reported).

	LCOM	Management	CBO	DOC_UPDATE	REMOVED_OBJECT_STATE	DIT	Nesting	STATEMENT_INSERT	ADDITIONAL_FUNCTIONALITY
RFC	2.88	4.47	9.66	18.45	22.86	22.12	25.13	26.34	24.15
LCOM		1.27	6.20	14.36	18.47	18.14	20.59	21.54	19.91
Management			5.48	14.83	19.66	18.97	22.18	23.54	21.14
CBO				9.16	14.04	13.81	16.62	17.79	15.84
DOC_UPDATE					5.30	5.97	8.18	9.11	7.85
REMOVED_OBJECT_STATE						1.45	2.93	3.48	3.08
DIT							1.00	1.28	1.32
Nesting								0.27	0.47
STATEMENT_INSERT									0.27

5.1 Experimental Setup

Concerning the sample set, we first randomly selected 50 question posts out of 178,409 for which we could extract a link to Android API classes and that had an accepted answer. Second, we randomly selected 50 posts out of 58,948 posts that are within the top 10% regarding their `score` or `favoriteCount` to make sure that we covered discussions favored by Android app developers.

We investigated the question and accepted answer of each post as follows. First, we checked whether the problem described in the post can be attributed to our properties of the API classes or other API properties currently not covered by our approach. Furthermore, we investigated the reason(s) of developers for discussing the problem on Stack Overflow considering the type of question and the main topic of each post.

5.2 Results

Our set of 100 posts references 251 Android API classes. Among the most frequently referenced API classes are `android.widget.TextView` (342), `android.view.View` (322), and `android.app.Activity` (302).

5.2.1 Posts, Code Metrics, and Code Smells

By manually analyzing the posts, we found that API classes that are frequently referenced are also among the classes with large values for several source code metrics supporting the findings of our quantitative analysis. For instance, the API classes `android.view.View` and `android.widget.TextView` are the two largest classes with an LOC of

8,624 and 5,752. They also exhibit the highest values for the WMC metrics, namely 1,924 and 1,402. Together with `android.app.Activity`, they are also among the classes with the highest CBO, namely 166, 157, and 105. We found similar results for code smells: the API classes referenced most frequently in posts are also the classes affected by high severity code smells. For instance, the API classes `android.view.View`, `android.widget.TextView`, and `android.app.Activity` suffer from code smells with severity 10 (according to the inCode tool).

5.2.2 Posts and Changes

The manual analysis showed several examples to support our finding that changes in API classes can lead to more references in posts on Stack Overflow. 3 posts discuss bugs in the Android API and restrictions of Android versions to access Micro SD cards and 2 posts discuss whether the stated problem is a bug in the Android API or the developers' code. Further 3 posts discuss how to implement features for newer or older versions of the API. In 2 of the 100 posts the problem relates to deprecated methods in the API classes `android.webkit.Webview` and `android.app.Service`. The post with the title *"Android - Highlight text in WebView above android version 4.0.3"*[12] is a good example of a post resulting from a change in the API class `WebView`. The question is presented in Figure 8 and its accepted answer in Figure 9, showing that this question arises due to a change in the API.

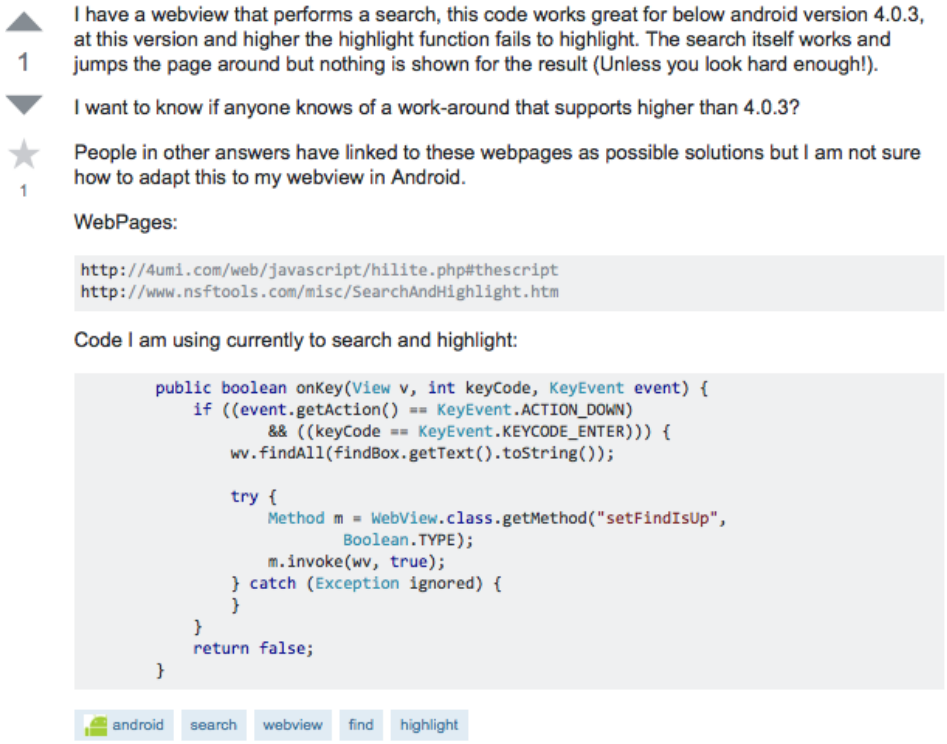
5.2.3 Reference Count, Post Count, and Reference-Density

In the set of manually analyzed posts, we found that the API classes `android.widget.TextView`, `android.app.Activity`, and `android.util.Log` are most frequently used and referenced. However, for several of these classes we also found evidence that high reference and usage count does not necessarily mean that the API class is the (main) subject of the discussion. For instance, the class `TextView` is the main subject in only 7 posts though it is referenced 342 times in 34 out of 100 posts. The class `Activity` is referenced 302 times in 45 out of 100 posts but discussed in only 2 posts. Moreover, the class `Log` is often referenced in posts and used in apps, however our sample set did not show a single post in which problems with logging are discussed. The `refDensity` of these API classes supports this finding, since `TextView` and `Activity` are more reference-dense than `Log`. This indicates, that regarding the identification of problems in API classes, the reference-density of an API class might be a better indicator for problematic classes than the reference count.

5.2.4 Other API Properties

In addition to our API specific properties, we found a number of other properties that led to discussions of API classes on Stack Overflow. This supports the findings of Rigby et al. (41) who stated that classes in code snippets tend to be less important to the purpose of the post. 17 posts discuss problems with handling exceptions. In 10 posts,

[12] http://stackoverflow.com/questions/11910041/android-highlight-text-in-webview-above-android-version-4-0-3



1

I have a webview that performs a search, this code works great for below android version 4.0.3, at this version and higher the highlight function fails to highlight. The search itself works and jumps the page around but nothing is shown for the result (Unless you look hard enough!).

I want to know if anyone knows of a work-around that supports higher than 4.0.3?

1

People in other answers have linked to these webpages as possible solutions but I am not sure how to adapt this to my webview in Android.

WebPages:

```
http://4umi.com/web/javascript/hilite.php#thescript
http://www.nsftools.com/misc/SearchAndHighlight.htm
```

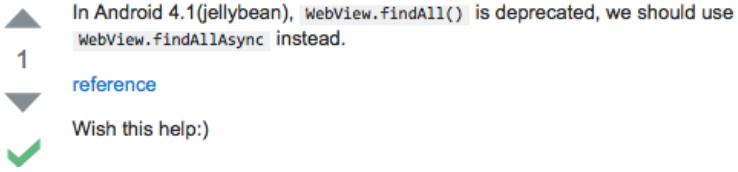
Code I am using currently to search and highlight:

```
        public boolean onKey(View v, int keyCode, KeyEvent event) {
            if ((event.getAction() == KeyEvent.ACTION_DOWN)
                    && ((keyCode == KeyEvent.KEYCODE_ENTER))) {
                wv.findAll(findBox.getText().toString());

                try {
                    Method m = WebView.class.getMethod("setFindIsUp",
                            Boolean.TYPE);
                    m.invoke(wv, true);
                } catch (Exception ignored) {
                }
            }
            return false;
        }
```

android search webview find highlight

Figure 8: Question about `ImageView` from Stack Overflow post 11910041



1

In Android 4.1(jellybean), `WebView.findAll()` is deprecated, we should use `WebView.findAllAsync` instead.

reference

Wish this help:)

Figure 9: Accepted answer for the question of post 11910041

the developers mention tutorials and documentation should cover parts of the Android API in more detail. We found 4 posts on hardware specific issues due to the media player and camera of the devices. Another 4 questions were posted by newbies in Android app development who faced problems with user interface components. Furthermore, several posts discuss the interaction of API classes, such as `Activity`, `AsyncTask`, and `Intents`. Finally, we found 7 posts that discuss issues with external APIs, such as Twitter or Libgdx, that we did not consider in our studies.

5.3 Summary of Results

Summarizing the results of the manual analysis, we found evidence that API classes affected by code smells and bugs, having large values for several source code metrics, and change frequently are also frequently referenced in posts on Stack Overflow. Furthermore, we found support that API classes with higher reference-density are more likely the topic of discussions in posts on Stack Overflow than frequently referenced API classes. While these findings support and validate the results of our quantitative studies in Section 3 and Section 4, we also found a number of other properties of API classes, such as exception handling, missing tutorials and documentation, that impact the amount of references and the reference density of API classes. For future work, we plan to consider these properties to obtain a more comprehensive view on the reasons for reference-prone, post-prone, and reference-dense API classes.

6 Discussion

In this section, we discuss the implications of our results, as well as threats to validity.

6.1 Implications of Results

In the following, we report and discuss implications of our findings for both researchers and practitioners.

6.1.1 For Researchers

The results of the two quantitative studies presented in Section 3 and Section 4 confirm the findings of Kavaler et al. (22) and Linares-Vásquez et al. (26) that the number of references of API classes on Stack Overflow correlates with the size and changes of these API classes. Additional, we found statistical evidence that also the code metrics WMC, RFC, CBO, and LCOM impact the amount of discussion whereby higher values of these metrics for an API class relate to more references, posts, and higher reference-density of that class.

The different results for the experiments with *ref-prone*, *p-prone*, and *ref-dense* API classes presented in Section 4 can be explained by different selections of variables for each run. To mitigate this bias, we repeated the experiments for classifying *ref-prone*, *p-prone*, and *ref-dense* API classes with RF for each combination of variables from each cluster, resulting in $6 \cdot 3 \cdot 3 = 54$ runs. In all three experiments we found that the variables from the first and second cluster show MDGs > 25 over all runs. Concerning the metrics from the first cluster, the 54 runs to classify *ref-prone* and *p-prone* API classes show the highest median MDG for WMC. For *ref-dense* the 54 runs show the highest median MDG for LOC. For the second cluster, the highest MDGs are obtained for commitCount for *ref-prone* and *p-prone*, and the commit category Management for *ref-dense* (details are provided in the supplementary material). Combining the results from the three experiments, we found that the larger and more complex API classes are, and

the more dependent and coupled to other classes they are, and the more frequently they change, the more frequently these classes are referenced in code snippets on Stack Overflow. Although we could not fully show causation between API class properties and the amount of references, these results should be taken into account in future research on this topic.

While the qualitative analysis in Section 5 validated several findings of our quantitative studies, it also showed that a high number of references of an API class in a post does not always imply that this class is the main subject of the discussion. Instead, we found evidence that the reference-density is a more precise measure for detecting problematic API classes. This calls for a refinement of our assumption that the amount of references and posts relates only to problems in API classes. Consequently, among the API properties investigated in our studies, researchers analyzing problems in API classes need to take into account the topics of discussions.

Furthermore, the qualitative analysis provided evidence of other properties leading to more references and posts on Stack Overflow. The most prominent properties are exception handling, missing tutorial and documentation, and the interaction of API classes that should be considered by future research on this topic.

6.1.2 For Android API Developers

Since all steps of our approach can be performed with existing tools, Android API developers can use our approach to find out the API classes that are frequently referenced on Stack Overflow (compared to their usage), plus potential reasons in terms of code metrics, code smells, and changes. Considering these recommendations when modifying and improving the source code, the discussion on Stack Overflow about certain API classes can decrease. We want to emphasize, that discussion about an API in general is not a bad thing, but discussion that is caused, for instance, by lacking quality of API code, could be diminished. This study is a first step towards a recommender system for Android API developers to suggest API classes that cause discussions and potential reasons for them. Further studies investigating the influence of the other properties that we found in our manual analysis are needed.

6.1.3 For API Developers

Developers of other Java frameworks, such as the Spring framework,[13] or Apache Commons,[14] that are discussed on Stack Overflow can also use our approach to identify reference-prone, post-prone, and reference-dense API classes. APIs developed in other programming languages than Java are currently not supported by our approach, since tools such as ChangeDistiller and JavaBaker need to be adapted first.

[13] https://spring.io
[14] http://commons.apache.org

6.2 Threats to validity

A threat to *construct validity* concerns the bias in our assumption that the main topic of a post is related to the number of references and posts of an API class. The manual evaluation of the posts revealed that frequently referenced API classes represent not always the main topics of the discussions. In future work, we will address this threat by also taking into account the topic. Another two threats concern the usage of the heuristics of Hattori et al. (19) for determining the four commit categories and the accuracy of ChangeDistiller to extract all code changes. The commit categories of Hattori et al. were evaluated with nine open source projects. The accuracy of ChangeDistiller was evaluated extensively with a benchmark presented in (12) and in a number of empirical studies, such as presented in (14, 28). Another threat to construct validity concerns the usage of JavaBaker (47) and *jclassinfo*. JavaBaker links code snippets to API elements with 98% precision, which we consider as sufficient. The investigation of posts without code snippets and code snippets with less then 3 lines of code is left for future work, as well as extending the linking between API elements and posts to referenced classes in the body and title of the posts. *jclassinfo* lists the references of API classes and does not distinguish between inheritance or invocation of used API classes. However, this thread is negligible since we only focused on the usage of the APIs and not on the type of the dependencies. Another threat to construct validity concerns the usage of only one API version, since problems may raise from a given API version. To diminish this thread, we assured that the API version we use is the most wide spread one, which was API version 19 in our case. We plan to extend the study to all available API versions in future studies.

Two threats to *internal validity* concern the selection of code metrics and code smells for our analysis. To mitigate the threat imposed by code metrics, we selected various types of code metrics, namely object-oriented metrics, volume metrics, and complexity metrics. Regarding code smells, we used inCode to detect code smells. inCode is capable of detecting a variety of code smells presented by Lanza and Marinescu (23). However, we might still have missed some code smells, therefore we plan to extend our study in future work considering also other smell detection tools, such as DECOR (35). Another threat to internal validity concerns the selection of the apps from the Android app store. We are aware that when excluding API classes from our investigations that are not used in our selection of Android apps, we might miss API classes that are used in apps. We addressed this threat by selecting the top 80 apps covering the various categories on the Android app store. In total, our set comprises 2,715 apps allowing us to draw our conclusions with a confidence level of 95% and a margin of error of 2%. Another threat to internal validity is related to the lack of causation, i.e., while we identified properties correlating with *ref-prone*, *p-prone*, and *ref-dense* APIs, we cannot claim that such properties actually caused discussions on Stack Overflow. We mitigated this threat through the qualitative analysis of Stack Overflow posts reported in Section 5. Another threat to internal validity concerns the possibility that decompiled source code of apps could include Android API libraries. We neglect this threat assuming that if the same API libraries are included in all classes the numbers would not change. In addition,

we believe that the overall results would not change, since all our experiments (also without usage count) showed the same results.

Threats to *external validity* mainly concern the generalizability of our results. To mitigate this threat, we performed the study with the Android API version 19 that is the most widely distributed version of the Android API. Still, more studies with other APIs need to be performed to obtain more generalizable results. In future work, we plan to extend our study to more versions of the Android API and also other APIs.

7 Related Work

In this section, we first present studies linking posts of Stack Overflow to API code. Next, we briefly summarize prior studies related to the Android API and app development and highlight the novelty of our research.

7.1 Links from Posts to Code

Linking informal documentation, such as posts from Stack Overflow to API classes and methods has been the focus of several studies. Rigby et al. (41) presented the tool ACE, extracting essential code elements using island parsing from posts of Stack Overflow with 90% precision for Android. Furthermore, Ponzanelli et al. (37, 38) presented the Eclipse plugins Seahawk and Prompter that link written source code to posts of Stack Overflow using Apache Lucene und SolR. In this study, we used JavaBaker from Subramanian et al. (47) to link code snippets of Stack Overflow to API classes using an iterative approach and an oracle. We chose JavaBaker since it shows a precision of 98% and a recall of 83% for Java code and handles declaration ambiguity of classes and methods.

7.2 Code Characteristics

A number of prior studies investigated the code characteristics of the Android API and Android apps. Similar to our findings, Kavaler et al. (22) found that class size and class documentation size have an impact on the mentions in questions on Stack Overflow. However, we also investigated other code metrics, such as complexity metrics and object oriented metrics. Mannan et al. (30) and Hecht et al. (20) investigated code smells in Android apps whereas Hecht et al. found that code smells influence the performance of Android apps. Reimann et al. (40) provided a catalog of code smells for Android app developers. Maji et al. (29) investigated the bug reports and bug fixes of the Android open source project. They found that despite the high cyclomatic complexity of Android API classes, the bug density is low. However, none of these studies investigated the relationships between API classes and the amount of references and posts on Stack Overflow.

7.3 Bug Reports and Changes

Bug reports and code changes of the Android API and apps have been investigated by several researchers. For instance, Han et al. (18) analyzed Android bug reports to investigate vendor specific issues using labeled-LDA. Furthermore, Martie et al. (31) investigated the XML-files of bug reports with LDA to infer trends in the discussions of the Android open source project. They found that the trend of discussions about runtime errors and the graphics library declined with the release of Android Gingerbread. The study of Asaduzzaman et al. (1) showed that bug introducing changes affect more files than fixing commits. Investigating the changes of the Android API, McDonnell et al. (32) showed that the Android APIs related to hardware, user interface, and web change most frequently. Furthermore, they showed that fast evolving APIs are more frequently used than slow ones.

Similar to our study, Linares-Vásquez et al. (26) examined if API changes trigger discussions on Stack Overflow. As confirmed by our results, they found that change-prone API methods and methods that change their behavior are more often discussed on Stack Overflow. They also found that the deletion of public methods leads to discussions of experienced developers on Stack Overflow. In another two studies, Bavota et al. (4) and Linares-Vásquez et al. (25) investigated the relationships between the success of apps, the APIs that are used, and the change and error-proneness of the used API classes. They found that successful apps use less error-prone APIs and API methods that are changed frequently are more often referenced in posts on Stack Overflow. These results are confirmed by Guerrouj et al. (17) who investigated how code churn of apps influence the success of apps and the discussions on Stack Overflow. In contrast to these studies, we take into account several code metrics, and moreover, code smells, changes, and the usage count of API classes and compute prediction models to investigate the impact of them on the amount of references and on the density of references.

7.4 Usage of Android API Classes

Several studies have investigated the usage of Android API classes. For instance, Wang et al. (51) investigated Stack Overflow posts that are tagged with the names of API classes and revealed that there are API usage obstacles, for instance for the classes of the package `android.widget`. They investigated 250 apps downloaded from F-Droid and extracted the usage of API classes with regular expressions.

Ruiz et al. (44) revealed that about 23% of the classes in Android applications inherit from Android API classes. Parnin et al. (36) found that the majority of Android API classes are referenced in posts on Stack Overflow. In particular, the classes of the packages `android.widget` and `android.view` are referenced frequently. Furthermore, they found classes that are not often discussed on Stack Overflow are not used frequently in apps. Additional, Azad et al. (2) investigated the usage of the Android API classes on Stack Overflow and predicted changes in apps based on the usage of the API.

Kavaler et al. (22) found a strong relationship between the usage of classes in apps of the Android market and references on Stack Overflow. We found similar results as Wang et al. (51) and Kavaler et al. (22). In contrast to these studies, we considered a significantly larger set of metrics and also more Android apps for our study. To the best of our knowledge, we are the first who investigated the relationships between API classes and their references on Stack Overflow considering various properties of API classes, namely code metrics, code smells, changes, commit categories, and usage.

8 Conclusions and Future Work

In this paper, we investigated various properties of API classes and their relationships to the number of references in (distinct) posts on Stack Overflow and to the number of references per usage in Android apps. As properties, we considered code metrics, code smells, code changes, and commit categories of Android API classes used in Android apps.

The results of our study showed that code metrics measuring the size, coupling and complexity of classes, as well as changes measured in terms of number of commits are related to the reference-proneness, post-proneness, and reference-denseness of API classes. A manual investigation with a random selection of 100 posts validated and confirmed these findings. But it also showed that the topics of the discussions and other properties of API classes, such as exception handling, missing tutorials, and documentation, should be considered as well for identifying problems regarding the usage of API classes.

The results of our studies have several implications on researchers, developers of the Android API and other APIs. The former should consider code metrics, most important WMC (weighted method count), RFC (response for a class), and LCOM (lack of cohesion in methods) in addition to API changes when investigating the reasons for frequently discussed API classes. Android API developers can use our approach and results to highlight reference-prone, post-prone, and reference-dense API classes, plus potential reasons in terms of code metrics, code smells, and changes. Furthermore, our approach is easily applicable to other APIs as well and can support API developers to find possible reasons for frequently discussed classes.

As the next step in our work, we plan to apply topic analysis to Android-related posts and perform surveys with Android API and app developers to get insights on the problems of API classes discussed on Stack Overflow. Furthermore, we plan to investigate more code smells and metrics, consider a finer level of granularity, such as method level, and extend our study to other versions of the Android API and APIs of other frameworks. Additionally, we plan to apply Granger's causality test (15) on our data to investigate the temporal causation between the change properties of API classes and the reference-proneness and reference-denseness of these API classes on Stack Overflow.

References

1. Asaduzzaman M, Bullock M C, Roy C K, Schneider K A. Bug introducing changes: A case study with Android 2012 (pp. 116–119)., Proceedings of the working conference on mining software repositories: ACM/IEEE.

2. Azad S, Rigby P C, Guerrouj L. Generating API call rules from version history and Stack Overflow posts. *ACM Transactions on Software Engineering and Methodology*. 2017;25(4):29.

3. Barua A, Thomas S., Hassan A E. What are developers talking about? an analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*. 2012;19:1–36.

4. Bavota G, Linares-Vásquez M, Bernal-Cárdenas C E, Di Penta M, Oliveto R, Poshyvanyk D. The impact of API change- and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering*. 2015;41(4):384–407.

5. Breiman L. Random forests. *Machine Learning*. 2001;45(1):5–32.

6. Chidamber S R, Kemerer C F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*. 1994;20(6):476–493.

7. Cohen J. *Statistical Power Analysis for the Behavioral Sciences*: Lawrence Erlbaum Associates; 1988.

8. Cortes C, Vapnik V. Support-vector networks. *Machine Learning*. 1995;20(3):273–297.

9. Cox D R. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*. 1958:215–242.

10. D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches 2010 (pp. 31–41)., Proceedings of the working conference on mining software repositories: IEEE.

11. Fluri B, Gall H C. Classifying change types for qualifying change couplings 2006 (pp. 35–45)., Proceedings of the international conference on program comprehension: IEEE.

12. Fluri B, Würsch M, Pinzger M, Gall H C. Change Distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*. 2007;33(11):725–743.

13. Franke D, Weise C. Providing a software quality framework for testing of mobile applications 2011 (pp. 431–434)., Proceedings of the international conference on software testing, verification and validation: IEEE.

14. Giger E, Pinzger M, Gall H C. Comparing fine-grained source code changes and code churn for bug prediction 2011 (pp. 83–92)., Proceedings of the working conference on mining software repositories: ACM.

15. Granger C W J. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*. 1969;37(3):424–438.

16. Grissom R J, Kim J J. Effect sizes for research. a broad practical approach. *Mahwah, NJ: Earlbaum*. 2005.

17. Guerrouj L, Azad S, Rigby P C. The influence of app churn on app success and stackoverflow discussions 2015 (pp. 321–330)., International conference on software analysis, evolution and reengineering: IEEE.

18. Han D, Zhang C, Fan X, Hindle A, Wong K, Stroulia E. Understanding android fragmentation with topic analysis of vendor-specific bugs 2012 (pp. 83–92)., Proceedings of the working conference on reverse engineering: IEEE.

19. Hattori L P, Lanza M. On the nature of commits 2008 (pp. 63–71)., Proceedings of the international conference on automated software engineering - workshops: IEEE/ACM.

20. Hecht G, Moha N, Rouvoy R. An empirical study of the performance impacts of android code smells 2016, Proceedings of the international conference on mobile software engineering and systems: ACM.

21. Jureczko M, Spinellis D. Using object-oriented design metrics to predict software defects. *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*. 2010:69–81.

22. Kavaler D, Posnett D, Gibler C, Chen H, Devanbu P, Filkov V. Using and asking: APIs used in the Android market and asked about in StackOverflow 2013 (pp. 405–418)., Social informatics: Springer.

23. Lanza M, Marinescu R. *Object-oriented metrics in practice: Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. 1st ed.: Springer Publishing Company, Incorporated; 2010.

24. Liaw A, Wiener M. Classification and regression by randomforest. *R News*. 2002;2(3):18–22.

25. Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, Di Penta M, Oliveto R, Poshyvanyk D. API change and fault proneness: a threat to the success of Android apps 2013 (pp. 477–487)., Proceedings of the joint meeting on foundations of software engineering: ACM.

26. Linares-Vásquez M, Bavota G, Di Penta M, Oliveto R, Poshyvanyk D. How do api changes trigger stack overflow discussions? a study on the android sdk 2014 (pp. 83–94)., Proceedings of the international conference on program comprehension: ACM.

27. Louppe G, Wehenkel L, Sutera A, Geurts P. Understanding variable importances in forests of randomized trees 2013 (pp. 431–439)., Proceedings of the international conference on neural information processing systems: Curran Associates Inc.

28. Macho C, McIntosh S, Pinzger M. Predicting build co-changes with source code change and commit categories 2016 (pp. 541–551)., Proceedings of the international conference on software analysis, evolution, and reengineering: IEEE.

29. Maji A K, Hao K, Sultana S, Bagchi S. Characterizing failures in mobile oses: A case study with android and symbian 2010 (pp. 249–258)., Proceedings of the international symposium on software reliability engineering: IEEE.

30. Mannan U A, Ahmed I, Almurshed R A M, Dig D, Jensen C. Understanding code smells in android applications 2016 (pp. 225–234)., Proceedings of the international workshop on mobile software engineering and systems: ACM.

31. Martie L, Palepu V K, Sajnani H, Lopes C. Trendy bugs: Topic trends in the android bug reports 2012 (pp. 120–123)., Proceedings of the working conference on mining software repositories: IEEE.

32. McDonnell T, Ray B, Kim M. An empirical study of API stability and adoption in the Android ecosystem 2013 (pp. 70–79)., Proceedings of the international conference on software maintenance: IEEE.

33. Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F. *e1071: Misc functions of the department of statistics (e1071), tu wien*; 2014. R package version 1.6-4.

34. Minelli R, Lanza M. Software analytics for mobile applications–insights & lessons learned 2013 (pp. 144–153)., Proceedings of the european conference on maintenance and reengineering: IEEE.

35. Moha N, Gueheneuc Y G, Duchien L, Le Meur A F. Decor: A method for the specification and detection of code and design smells. *IEEE*. 2010Jan;36(1):20–36.

36. Parnin C, Treude C, Grammel L. *Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow,*: Georgia Institute of Technology; 2012.

37. Ponzanelli L, Bacchelli Alberto, Lanza M. Seahawk: stack overflow in the ide 2013 (pp. 1295–1298)., Proceedings of the 2013 international conference on software engineering: IEEE.

38. Ponzanelli L, Bavota G, Di Penta M, Oliveto R, Lanza M. Mining stackoverflow to turn the ide into a self-confident programming prompter 2014 (pp. 102–111)., Proceedings of the working conference on mining software repositories: ACM.

39. R Core Team. *R: A language and environment for statistical computing*. Vienna, Austria, R Foundation for Statistical Computing; 2013.

40. Reimann J, Brylski M, Amann U. A tool-supported quality smell catalogue for android developers 2014, Proceedings of the conference modellierung in the workshop modellbasierte und modellgetriebene softwaremodernisierung.

41. Rigby P C, Robillard M P. Discovering essential code elements in informal documentation 2013 (pp. 832–841)., Proceedings of the 2013 international conference on software engineering: IEEE/ACM.

42. Romano D, Pinzger M. Using source code metrics to predict change-prone java interfaces 2011 (pp. 303–312)., Proceedings of the international conference on software maintenance: IEEE.

43. Rosen C, Shihab E. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*. 2015;21:1–32 (English).

44. Ruiz I J M, Nagappan M, Adams B, Hassan A E. Understanding reuse in the android market 2012 (pp. 113–122)., Proceedings of the international conference on program comprehension: IEEE.

45. Scott A. J., Knott M. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*. 1974;30(3):507–512.

46. Subramanian S, Holmes R. Making sense of online code snippets 2013 (pp. 85–88)., Proceedings of the international workshop on mining software repositories: IEEE.

47. Subramanian S, Inozemtseva L, Holmes R. Live API documentation 2014 (pp. 643–652)., Proceedings of the international conference on software engineering: ACM.

48. Tantithamthavorn Chakkrit. *ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test*; 2017

49. Tantithamthavorn Chakkrit, McIntosh Shane, Hassan Ahmed E., Matsumoto Kenichi. An empirical comparison of model validation techniques for defect prediction models. 2017;1.

50. Venables W N, Ripley B D. *Modern applied statistics with s*. Fourth. New York: Springer; 2002. ISBN 0-387-95457-0.

51. Wang W, Godfrey M W. Detecting API usage obstacles: A study of iOS and Android developer questions 2013 (pp. 61–64)., Proceedings of the working conference on mining software repositories: IEEE.

AAU-SERG-2017-002