Mining families of android applications for extractive SPL adoption

Li Li, Jabier Martinez, Tewfik Ziadi, Tegawendé Bissyandé, Jacques Klein, Yves Le Traon

Mining Families of Android Applications for Extractive SPL Adoption

Li Li
SnT, University of Luxembourg
Luxembourg
li.li@uni.lu

Jabier Martinez
SnT, University of Luxembourg
Luxembourg
& LIP6, Sorbonne Universités,
UPMC Univ Paris 06, CNRS
jabier.martinez@uni.lu

Tewfik Ziadi
LIP6, Sorbonne Universités,
UPMC Univ Paris 06, CNRS
Paris, France
tewfik.ziadi@lip6.fr

Tegawendé F. Bissyandé
SnT, University of Luxembourg
Luxembourg
tegawende.bissyande@uni.lu

Jacques Klein
SnT, University of Luxembourg
Luxembourg
jacques.klein@uni.lu

Yves Le Traon
SnT, University of Luxembourg
Luxembourg
yves.letraon@uni.lu

ABSTRACT

The myriads of smart phones around the globe gave rise to a vast proliferation of mobile applications. These applications target an increasing number of user profiles and tasks. In this context, Android is a leading technology for their development and on-line markets are the main means for their distribution. In this paper we motivate, from two perspectives, the mining of these markets with the objective to identify families of apps variants in the wild. The first perspective is related to research activities where building realistic case studies for evaluating extractive SPL adoption techniques are needed. The second is related to a large-scale, world-wide and time-aware study of reuse practice in an industry which is now flourishing among all others within the software engineering community. This study is relevant to assess potential for SPLE practices adoption. We present initial implementations of the mining process and we discuss analyses of variant families.

Keywords

Software Product Line Engineering; Reverse Engineering; Mining Software Repositories; Android; AppVariants

1. INTRODUCTION

Software repositories contain a wealth of artefacts and information that can be mined to study development processes and experimentally assess research approaches. In recent years, GitHub has offered such opportunities to the Software Engineering research community with its millions of projects. Unfortunately, there are a lot of perils in using GitHub data [5]. Among these, it is noteworthy that a large proportion of GitHub projects are toy projects, while other repositories are frequently used for sharing code between a limited number of people. Findings on such software data are thus often not generalisable of software development. Recently, on the other hand, with the wide adoption of smart mobile devices, application (app) markets are growing with millions of software artefacts, written by isolated developers as well as by companies to target large user bases.

SPLE research community can leverage the content of app markets. It is a common belief that case studies for experimenting with SPL adoption approaches are needed. One of the adoption models is the *extractive* approach where the organization capitalizes on existing custom software systems by extracting the common and varying source code into a single production line [8]. In this direction, and in order to cover the whole transition process, several techniques have been proposed, such as feature identification, feature location, mining feature constraints or extraction of reusable assets [14]. However, SPL migration remains a challenging endeavour at technical level and, to undermine this situation, researchers find great difficulty in obtaining software variants which are realistic and not proprietary. ArgoUML SPL derived variants [3] is a widely used case study, however, more case studies are essential in order to empirically evaluate the proposed techniques. While ArgoUML SPL variants among others are created "in the lab", we present an exploratory study to identify variant families "in the wild". In the past, this kind of exploration already culminated in realistic benchmarks (e.g. Linux kernel variants [19] or Eclipse releases [15]).

Games, weather, social networking services (SNS), navigation, music or news are among the most popular apps categories and each app target specific user needs. In our definition, it is not mandatory that a family belongs to the same category. We consider a family as a set of variants from which feature-oriented systematic reuse may be performed. A feature is a characteristic, quality or user-visible aspect of a software system or systems [6]. Some approaches have been proposed to try to identify features information from a single product (e.g. [7]). In some SPL adoption scenarios, it is possible that the SPL wants to be created from a single product by separating its features. In our case we focus on the scenario of several variants created by reusing domain assets to fit different customer needs. Therefore, we propose

mining variants, and not single products, directly from public Android apps markets. It is important to clarify that we are more interested in variability in space (variants) than in variability in time (versions). However, as we will see later, we keep the information of the versions.

The benefits of systematic and planned reuse have already been evidenced in "traditional" SPL domains such as embedded systems[1]. By targeting Android applications we aim to explore a software engineering industry where SPL practices can carry a lot of potential. The market for Android apps is experiencing a steady increase in demand since the mass adoption of smart-phones[2]. In our opinion, this fact guarantees the interest in this topic in the years to come. We consider that a large-scale, world-wide and time-aware study of reuse practice in this domain can be leveraged to propose automatic assessment of the potential of Android apps families to adopt an SPL approach.

The remainder of the paper is structured as follows: Section 2 presents a motivating example. Section 3 summarizes our vision and contribution. Section 4 presents our solution for the family variants mining process. Section 5 presents and discusses the analysis of selected families. Section 6 presents related work and Section 7 concludes this work.

2. MOTIVATING EXAMPLE

8684 is a Chinese company specialized in travel and transportation apps[3] which reached more than 5 million users. The 8684 family of variants that we manually identified motivated our interest in trying to automatically identify other families in app markets. In the 8684 portfolio we have found several apps including `CityBus`, `Metro`, `Train` etc. Given the specialized nature of this company, their apps have many aspects in common such as time schedule or location management. We aim to explore the reuse that has been conducted while implementing the different apps and check if some of this reuse corresponds to distinguishable features. Our hypothesis is that, apart from the different graphical elements, there may be shared implementations of the business logic.

We analysed 6 variants with But4Reuse (Bottom-Up Technologies for Reuse) [14] which is a framework for extractive SPL adoption. In the packages that we identified as related to the company source code (`cn.tianqu` and `cn.chinabus`), 270 java files are shared (identical) by at least two variants, and 470 java files in average are specific to each variant. We ignore if the reuse was performed using copy-paste-modify or more advanced reuse techniques were being applied. Figure 1 illustrates the blocks of java files automatically identified among the variants in these two packages. This visualisation is known as the pruned concept hierarchy [17]. We included, for each Block, relevant words that appeared in the names of the files. `Block 0`, at the top, is source code shared by the 6 apps except `TrainTicket` and consists of 10 files with 333 lines of code (LoC) related to error handling. The blocks that appear with the name of the variants at the bottom (e.g. `Block 10` for `LongWayBus`) are the ones that are specific to each variant. `Block 8` to `11` are 20KLoC in average and `Block 12` and `13` are 3KLoC and 5KLoC respectively. From `Block 1` to `7` the average is 1733 LoC.

From a research perspective, one objective is to apply and validate techniques for feature identification and location towards the extraction of an SPL that will be able to, at least, derive the same 6 apps. Nevertheless, in this paper, our objective is to automatically identify a large number of other relevant families within app markets.

[1]http://splc.net/fame.html
[2]http://www.cnet.com/news/android-shipments-exceed-1-billion-for-first-time-in-2014
[3]http://www.8684.cn

3. VISION

We consider that empirical analyses of reuse practices in the domain of Android apps could be an interesting case study for SPL research and will provide insights on how Android app development, in its current state, can benefit from SPLE. In other domains, approaches for a collaborative assessment of the reuse potential has been proposed [16] and industrial experiences has been reported [4]. The vision is to automatize this assessment minimizing human involvement in the domain of Android apps. The following two points summarizes our vision regarding this initial work:

- **Towards building realistic case studies for extractive SPL adoption related techniques**
- **Large-scale, world-wide and time-aware study of reuse practices for automatic assessment of extractive SPL adoption in families of apps**

We present our initial prototypes and preliminary results:

- **An app families clustering method:** A method to mine datasets of apps in order to identify families of apps.

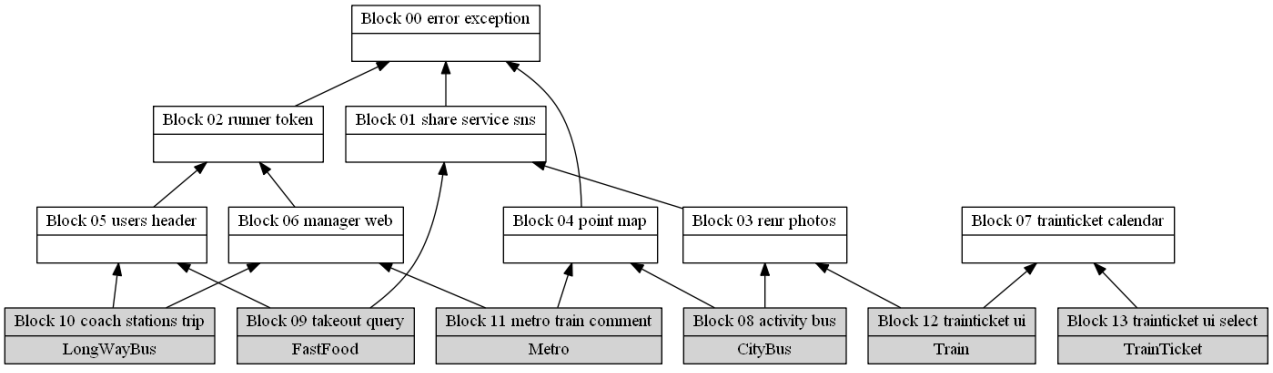

Figure 1: Concept lattice with the variants and blocks in the packages cn.tianqu and cn.chinabus of the 8684 family.

- **Preliminary analyses of selected app families:** We perform feature identification to discuss the characteristics of some mined app families.

4. MINING FAMILIES OF ANDROID APPS

Thanks to manual observation, we found three essential and simple characteristics which could be leveraged to mine app families, including 1) the unique package name of apps, 2) the certificate signed by app developers and 3) code similarity among apps. Figure 2 illustrates the working process of our approach, which takes as input a set of apps (e.g. from a market) and output the families of apps. We provide details about these three steps.

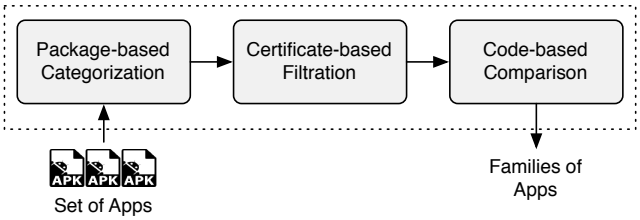

Figure 2: Steps of our families mining approach.

Package-based Categorization. In Android, each app is uniquely specified through a full Java-language-style package name. This package can be recognized because it is declared in the app meta-data. As officially recommended by Google[4], to avoid conflicts with other apps, developers should use internet domain ownership as the basis for their package names (in reverse). As an example, apps published by Adobe should start with *com.adobe*. Thus, if two apps start with a same company domain in their package names, these two apps are more likely developed by the same provider.

Certificate-based Filtering. Unfortunately, the package naming style is not respected in every case. Concretely, it is a common practice for malicious apps to use the same package names as other legitimate apps [13]. This is possible with a technique called repackaging where apps are disassembled and assembled again. Therefore, the validity of the package name characteristic is threatened. We complement this characteristic with another one related to the certificate of apps. In order to make an app publicly available in a market, developers have to sign their apps through their unique certificate. Thus, if two apps are signed by a same certificate, we have reason to believe that these two apps are developed by the same provider.

Code-based Comparison. In the last step, we perform pairwise comparison on apps that are located in a same family thanks to the aforementioned two steps. Based on the comparison results, we attempt to filter out such apps that are not sharing code with others and consequently can be considered as outliers. Given a threshold t, an app family F, and an app $a_i \in F$, for every $a_j \in F$, where $j \neq i$, the similarity distance $a_{ij} < t$, we consider that a_i is an outlier of family F, and thus we drop it from F in this step. For the source code similarity we computed it based on the formula $s = identicalMethods/totalMethods$, which has already been used in other studies [11]. Because of the unscalable characteristic of this step, we only use it to clean the families from potential outliers.

[4]http://developer.android.com/guide/topics/manifest/manifest-element.html

As summary, we cluster apps into a same family if they belong to the portfolio of the same app provider (step 1 and 2) and as long as they are partially similar in terms of source code similarity (step 3).

4.1 Implementation and Preliminary Results

We present a prototype tool called *AppVariants*, which is dedicated to validate the pertinence of the aforementioned three characteristics.

Prototype Implementation. *AppVariants* adopts a tree model to store the meta information of mined apps. Figure 3 shows an illustrative example for `com.baidu.*` apps on how their meta-data are stored. Each non-leaf node of the tree is represented by a package segment (e.g., `baidu`) while leaf node is represented through the remaining package segments (e.g., `BaiduMap`). Furthermore, each leaf node has affiliated a ranked list of meta-data of apps, including their certificates and assemble times. As shown in Figure 3, the time line of the vertical axis shows that the affiliated list is ranked through times and the apps it contains are actually different versions of the app indicated by the leaf node. Given a time point, the tree model also gives a way to identify family variants. For example, as shown in the dashed rectangle, we are able to collect a set of variants for `com.baidu`, given the latest time point.

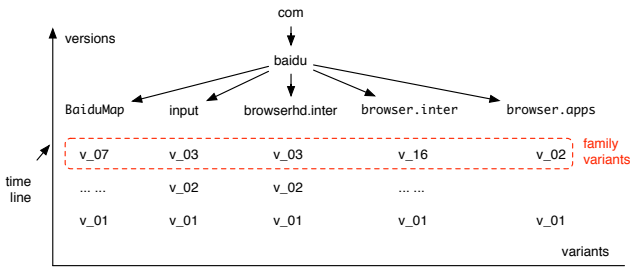

Figure 3: A simplified example showing how meta-data of app variants of Baidu are stored.

Experimental Setup. For the purpose of providing real family variants for SPL analysis, we need to apply our approach on a market scale. Thus, for this mining process, we use the AndroZoo [2] dataset to select around 1.5 million Google Play apps. This data set has already been used in other analyses [9, 10].

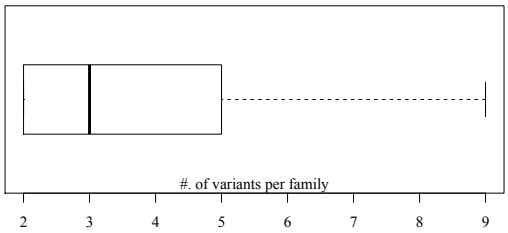

Figure 4: Boxplot on the number of variants per each family.

Preliminary Mining Results. Among the 1.5 million Android apps, based on package-based categorization and certificate-based filtration (steps 1 and 2), we are able to collect 75,963 families of apps. The number of variants in each family varies from 2 to 12,702 apps. Figure 4 plots the

distribution of the number of variants in our collected families before the third step using a boxplot. The median number of variants is three, meaning that half of the collected families have at least three variants. Actually, the boxplot only shows 88% families of our mined dataset, because we have disabled outliers in order to better present the results. The remaining 12% families have over 10 variants in each. Furthermore, 760 families in our mined dataset (i.e., 1%) have over 100 variants. Table 1 shows the top 10 families regarding the number of apps in the app provider portfolio. As an example, *com.andromo*, the top ranked package prefix, has 12,702 variants.

Table 1: Top 10 families in number of variants

Package Prefix	Variants	Package Prefix	Variants
com.andromo	12,702	com.jb	2,203
com.conduit	11,766	com.appsbar	2,143
sk.jfox	9,055	com.skyd	1,922
com.reverbnation	3,932	com.appmakr	1,787
air.com	2,732	com.gau	1,764

Thanks to our manual observation, we have found that, based only on the first two steps of our approach, we are already able to collect a big data set of families. However, some families that we collect do not share any common source code from one another. The family `com.ethereal` contains four variants, where the similarity of all of them are less than 5%. To this end, we leverage the code-based comparison (step 3) in order to mitigate those potential false positives. As our pairwise code comparison is not scalable to families with high number of variants, in this work, we randomly select 100 families of no more than 9 variants to evaluate the soundness of the code-based comparison step. Figure 5 illustrates the distribution of similarities between pairs of variants. The median value is around 77%, showing that variants of most families are actually sharing a lot of common code. This fact make them good samples for exploring and assessing SPL adoption.

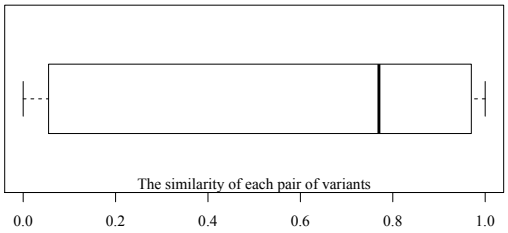

Figure 5: Distribution of the similarity of the variant pairs.

5. ANALYSIS OF SELECTED FAMILIES

In order to perform more in-depth analyses, we analysed the app families at different levels: files, source code and meta-data (e.g. user permissions defined in the Android manifest file) using Bottom-Up Technologies for Reuse [14]. In this Section we have categorized different cases that we have found from which we present a representative example.

Libraries Reuse. The third step (source code similarity) of the mining approach, presented in Section 4, does not restrict the similarity analysis to the main package of the apps, but to the whole source code which includes the libraries. In general, around 41% of an app source code is related to libraries [11]. As an example, we analysed a mined family with 4 apps: `MyShopping`, `BattleSpace`, `StrongBox` and `FrancsEuros`. The reuse only concerns the code of the used library `fr.pcsoft`. And very slight reuse between the apps in the provider package which a manual examination showed that it belongs to technicalities of the used development environment (i.e. WinDev) but not to actual relevant code. In apps clone detection it is a known limitation distinguishing actual source code from libraries and it is also a limitation of our approach. The main technique to avoid these false positives is to ignore common libraries during the similarity calculation step through whitelists with known and frequent libraries [11].

App Generators and Distributors. In Table 1 we presented app provider portfolios with elevated number of apps. We analysed variants from the `com.andromo` family to investigate the reason behind this fact. They all share a common block which is mainly related to screen layout management and many other blocks are shared between some of the variants. By looking at their website, Andromo is a company that offers a framework that aims to hide technical details of apps development and they provide services for easing apps distribution. That is the reason why they all have the certificate from `com.andromo` even if actually they are different developers from their community. Concretely, the main package of an app looks like this `com.andromo.dev282094.app267841` where dev is the id of the user and app is the id of this app. Preliminary analysis of the variants confirmed that the Andromo framework is feature-oriented providing initial components (e.g. photo gallery or radio) for user customization. Despite that the user customizations difficult the traditional automatic analysis of the variants' source code, more in-depth analyses of frameworks like this one, could aim to reverse engineer their variability and architecture. Given that the business value of these frameworks is their architecture, protection from extractive approaches like ours, which has been coined as *variability-aware security* [1], is an important research direction that can highly benefit from this work.

Content-driven Variability. We analysed the 6 mined variants from `tudou`. Tudou is a famous Chinese on-line video services company that broadcast users content and tv series. One outlier consisted in one heavily obfuscated app that had shared libraries with the others. We also found three that were very similar, almost complete app clones. A manual examination showed that they reused the same source code to distribute apps targeting different tv series. In these cases we have variability at the level of the app displayed content but the app architecture remains the same.

Device-driven Variability. We analysed the 6 mined variants from `baidu`, a famous Chinese web services provider. Baidu also provides an app software development kit which they extensively use in these 6 variants. In two apps, `Baidu Maps` and `input`, they share the voice recognition feature. In another two variants, `browser.inter` and `browserhd.inter`, we found that many code has been reused. They are both internet browser apps where one is specialized for mobile phones and the other for tablets. Therefore the variability exists because of the targeted device.

We cannot generalize the findings of these examples and we do not intend this categorization to be exhaustive. An automatic approach needs to be put in place to explore the mined apps family variants to determine the semantic and syntactic cohesion of the family members. The conclusion of such automatic analysis can spot interesting families.

6. RELATED WORK

As discussed in the introduction, Github is trending in the mining software repositories community. However, its expansion as public repository has degraded or complicated its practical exploitation in research [5], mainly because the results may not be generalizable to software engineering practices in industry. Without detracting from the validity of research using Github, we consider that Android apps at the app markets have more guarantees of being non merely personal projects.

The Diversify project has conducted several large-scale studies regarding software libraries usages in different contexts such as JavaScript websites, WordPress or Maven[5]. Their objective is to analyse the diversity as an enabler to adapt a software product during its evolution. In our work, library usage is also important but it is more important the actual code of the app. In previous work we analysed the usage of Android libraries [11] and others have analysed reuse in Android markets [18]. Our visions aims to bring the Android research community to topics specific to SPLE. The research field on malware detection is creating advanced and scalable methods for mining app markets [12]. Their objective is to heuristically define if a legitimate app has a malware counterpart signed with another certificate. Clone detection techniques and the analysis of other app characteristics, such as user permissions, are the basis for their analysis. Our objective is to find variants from the same provider to analyse its commonality and variability. However, we can see some synergies between the techniques used in extractive SPL adoption and the malware detection techniques.

7. CONCLUSIONS

Apps development is a flourishing industry which products are publicly available in apps market places. We proposed to use these apps datasets to identify apps families with two visions 1) provide realistic case studies for SPLE research in extractive SPL adoption and 2) study reuse practices and assess if app families are suitable for SPL adoption. We presented our prototype implementation of the mining process and the analysis of selected families.

As further work, we plan to improve the mining process and conduct more in-depth analysis to approach to these visions. Regarding the automatic mining algorithm, we aim to improve the source code similarity comparison with approaches that could escape cases of obfuscation, and also include semantic analysis of app descriptions and user comments in the market website. In relation to the analysis of families, we aim to report end to end extractive SPL adoption of selected mined families.

8. ACKNOWLEDGMENTS

This work was supported by the Fonds National de la Recherche (FNR), Luxembourg, under the AFR 7898764 and the project AndroMap C13/IS/5921289.

9. REFERENCES

[1] Mathieu Acher, Guillaume Bécan, Benoît Combemale, Benoit Baudry, and Jean-Marc Jézéquel. Product lines can jeopardize their trade secrets. In *FSE*, 2015.

[2] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *MSR*, 2016.

[3] Marcus Vinicius Couto, Marco Tulio Valente, and Eduardo Figueiredo. Extracting software product lines: A case study using conditional compilation. In *CSMR*, 2011.

[4] Dominik Domis, Stephan Sehestedt, Thomas Gamer, Markus Aleksy, and Heiko Koziolek. Customizing domain analysis for assessing the reuse potential of industrial software systems: experience report. In *SPLC*, 2014.

[5] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *MSR*, 2014.

[6] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[7] Christian Kästner, Alexander Dreiling, and Klaus Ostermann. Variability mining: Consistent semi-automatic detection of product-line features. *IEEE Trans. Software Eng.*, 40(1):67–82, 2014.

[8] Charles W. Krueger. Easing the transition to software mass customization. In *PFE*, 2001.

[9] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. ApkCombiner: Combining Multiple Android Apps to Support Inter-App Analysis. In *IFIP SEC*, 2015.

[10] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick Mcdaniel. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps. In *ICSE*, 2015.

[11] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *SANER*, 2016.

[12] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Yves Le Traon. Static analysis of android apps: A systematic literature review. Technical report, 2016.

[13] Li Li, Daoyuan Li, Tegawendé F Bissyandé, David Lo, Jacques Klein, and Yves Le Traon. Ungrafting malicious code from piggybacked android apps. Technical report, 2016.

[14] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Bottom-up adoption of software product lines: a generic and extensible approach. In *SPLC*, 2015.

[15] Jabier Martinez, Tewfik Ziadi, Mike Papadakis, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Feature location benchmark for software families using eclipse community releases. In *ICSR*, 2016.

[16] Muhammad Asim Noor, Paul Grünbacher, and Christoph Hoyer. A collaborative method for reuse potential assessment in reengineering-based product line adoption. In *CEE-SET*, 2007.

[17] Wiebke Petersen. A set-theoretical approach for the induction of inheritance hierarchies. *Electr. Notes Theor. Comput. Sci.*, 53:296–308, 2001.

[18] Israel J. Mojica Ruiz, Bram Adams, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, and Ahmed E. Hassan. A large-scale empirical study on software reuse in mobile apps. *IEEE Soft.*, 31(2), 2014.

[19] Zhenchang Xing, Yinxing Xue, and Stan Jarzabek. A large scale linux-kernel based benchmark for feature location research. In *ICSE*, 2013.

[5] http://diversify-project.eu/data/