ON AUTOMATICALLY DETECTING SIMILAR ANDROID APPS

By Michelle Dowling
Motivation

Searching for similar mobile apps is becoming increasingly important:
- Looking for substitute apps
- Opportunistic code reuse
- Prototyping
- Market analysis
- Finding a application similar to one in development
- Identifying salient features in successful apps
- Discovering plagiarism or clones across markets
- Learning how to use APIs
The Problem

- Due to the number of mobile apps available, determining similar apps is becoming increasingly harder

- This is worsened by the complexity of detecting similar apps
  - *Detecting high level requirements that match semantically*
  - *Some app repositories contain poorly functioning projects*
  - *Matching words in requirement documents with words in descriptions or source code does not mean it is relevant to the requirements*
  - *High numbers of matches in low-level functionality does not mean high-level functionalities match*
  - *Market-specific search engines only rely on text descriptions*
The Solution: CLANdroid

- Detects similar Android apps in free app markets or open source repositories
- Uses a set of semantic anchors to detect similarities between apps
  - Previously used semantic anchors (e.g. identifiers and API calls)
  - Explicit and implicit intents used by the apps
  - User permissions declared in the manifest files
  - Sensors used by the apps
- Capable of detecting similar apps, even when source code is not available
- Available as an online tool for detecting similar Android apps in which users can specify options and semantic anchors used
The Concept

- Uses ideas from CLAN:
  - Similar apps share some semantic anchors (e.g. API calls)
  - Requirements are implemented using combinations of different semantic anchors
  - Some semantic anchors are more descriptive of salient features
- CLANdroid adds intents, permissions, and sensors to the API semantic anchors
The Math

- Single Value Decomposition (SVD) is used to create a Term-Document Matrix (TDM) at app level that captures the semantic anchors and their frequencies.
- Using Latent Semantic Indexing (LSI) on each TDM extracts concepts for each semantic anchor.
- A similarity measure (cosine distance) can be computed using these results to determine how similar two apps are.
The Workflow

1. Apps are downloaded from Google Play in APK format or from open source repositories.
2. Semantic anchors are extracted from the apps statically by converting the APK files into JAR files and extracting information from the source files.
3. TDMs are created for each type of semantic anchor for each app (API calls, Sensors, Intents, Permissions, Identifiers).
4. The LSI algorithm is applied to each TDM between both apps. Using the results from the LSI algorithm, the cosine distance is computed between the apps.
5. Users can select a target app and semantic anchor in CLANdroid to perform searches and retrieve results based on the computed similarity index.
Empirical Study on CLANdroid

Goals:
- Evaluate which semantic anchors are most useful in detecting similar apps
- Analyze the impact of 3rd-party libraries and obfuscated code when trying to detect similar apps
Empirical Study on CLANdroid

Research Questions:

1. What semantic anchors used in CLANdroid produce better results when compared to the others?
   ■ Evaluates whether Android-specific semantic anchors outperform API calls

2. How orthogonal are the apps detected by CLANdroid as compared to Google Play?

3. Do 3rd-party libraries and obfuscated apps impact the accuracy of CLANdroid?
Empirical Study on CLANdroid

Answering RQ1 (What semantic anchors used in CLANdroid produce better results when compared to the others?):

- 12 apps were randomly selected from the pool of 14,450 apps downloaded from Google Play
- The top 3 similar apps from the same pool that belonged to the same category were determined based on 1 of 6 instances of CLANdroid:
  - (1) API calls, (2) Intents, (3) Permissions, (4) Sensors, (5) Identifiers, (6) API calls + Identifiers
- 6 versions of a 12-question survey (cross-validation design) were generated which asked 27 participants to rate the similarity of the 3 detected apps to the original app using a 4-point Likert scale
- To determine whether there were significant differences in the similarity rankings (using the Likert scales) of each CLANdroid instance, they used the Kruskal-Wallis test with post-hoc test procedure for pairwise comparisons
Empirical Study on CLANdroid

Answering RQ2 (How orthogonal are the apps detected by CLANdroid as compared to Google Play?):

- For each of the 14,450 apps, each of the previously mentioned 6 instances of CLANdroid were run to determine a set of similar apps (in any category of app)
- These rankings of similar apps from each of the CLANdroid instances were compared to the rankings of similar apps listed by Google Play
  - \( TOP↓R \) = The top rank of any application in Google Play's list that appears in the CLANdroid list
  - \( AVG↓R \) = The average rank of the applications from Google Play's list that appear in the CLANdroid list
- To evaluate the results, they compared the \( TOP↓R \) and \( AVG↓R \) series of the CLANdroid instances with the Kruskal-Wallis test with post-hoc procedure
Empirical Study on CLANdroid

Answering RQ3 (Do 3rd-party libraries and obfuscated apps impact the accuracy of CLANdroid?):

- Repeat of the process in RQ2, except obfuscated code and 3rd-party libraries are excluded
- To measure the impact of 3rd-party libraries and obfuscated code on CLANdroid's results, they only used pairwise comparisons using Mann-Whitney between the values of
  - (i) $TOP↓R$ and $AVG↓R$ with and without third party libraries, and
  - (ii) $TOP↓R$ and $AVG↓R$ with and without obfuscated code
Empirical Study on CLANdroid

Limitations

- There is no guaranteed symmetry in the lists of similar apps provided by Google Play (i.e., if app A is shown as similar to app B, app B may not be shown as similar to app A)
- The list of apps produced by Google Play is reduced to only contain apps within the set of 14,450 downloaded apps
- Between these two points, some apps in Google Play have no similar apps that are within the set of downloaded apps. However, these apps are still included in the study since removing them will lead to other apps having no similar apps to use
- Strong assumptions that Google Play's lists of similar apps are always right
- Similarity rankings are determined across categories, meaning searches for apps within categories are still affected by apps outside those categories
- 649 applications could not have third-party libraries distinguished from the rest of the code, so no information was extracted from the source code of these libraries
- The apps downloaded from Google Play are not necessarily representative of all apps in Google Play, and thus their conclusions cannot be generalized to all of Google Play
- The opinions of the participants on the similarity of apps is not representative of all user opinions
Results

RQ1 (What semantic anchors used in CLANdroid produce better results when compared to the others?):
- Using API calls to determine similar apps produces the best results
- Using identifiers was a close second, followed by permissions, then intents
- Sensors performed the worst and was determined to be statistically significant
Results

RQ2 (How orthogonal are the apps detected by CLANdroid as compared to Google Play?):
- On average, using API calls and intents result in the worst $TOPIR$ ratings; sensors were by far the best, followed by identifiers and permissions
- Similarly, sensors has the best average $AVGJR$ rating and API calls and intents were the worst
- These results are statistically significant in all cases except when comparing the results of identifiers vs. permissions and intentions vs. API calls combined with intentions
- On average, $TOPJR$ using any semantic anchor was high
  - Only 471 apps had an app from Google Play's list in its top position
  - This number is increased to 1,134 when apps of other categories are removed from CLANdroid's lists (663 apps had an app from a different category ranked higher)
  - Evaluation of the top rated similar apps in Google Play vs. CLANdroid suggests that the apps in Google Play's lists are not necessarily functionally similar; perhaps Google Play uses text descriptions and sensors more to determine app similarity
Results

RQ3 (Do 3rd-party libraries and obfuscated apps impact the accuracy of CLANdroid?):
- The results using permissions doesn't change as the manifest file doesn't change based on 3rd-party libraries or obfuscated code
- In terms of $AVG↓R$, identifiers are the best approach and API calls are the worst
- The results of $TOP↓R$ are similar
- $TOP↓R$ and $AVG↓R$ significantly improve by a large amount when 3rd-party libraries are excluded
- $AVG↓R$ significantly improves by a negligible amount when apps with obfuscated code are excluded
- $TOP↓R$ performed worse when apps with obfuscated code are excluded
Related Work

- AnDarwin extracts semantic vectors from source code methods and compares them, looking for code clones between apps to determine similarity.
- Dstruct decompiles APKs and walks through directories and files to determine directory structure. Trees representing directory structure are used to determine app similarity.
- Chen et al. detect similar apps by comparing centroids created from dependency graphs at the method level, but only flag pairs of apps as clones or not clones. They evaluated their approach across Android markets, but did not use Google Play.
- Gorla et al. use LDA to determine categories of apps, then cluster apps based on these categories using k-means.
- Desnos used method signatures to detect similar Android apps, where the signatures were composed of string literals, API calls, control flow structures, and exceptions.
- Wang et al. first remove code of 3rd-party libraries from APKs, then use fingerprints containing API calls to detect repackaged or cloned apps across different markets.
- Shao et al. initially cluster apps using resources (e.g., strings and images) and statistical features, then clusters on structural features.
- Thung et al. also base their work on CLAN, but use the tags for the systems in SourceForge instead of API calls to determine app similarity.
Discussion

- Overall, do you think the paper succeeded in addressing the problem posed?
Discussion

- What strengths does this paper have?
Discussion

- What weaknesses does this paper have?
Discussion

- How useful do you think being able to detect functionally similar apps is? How might the ability to detect functionally similar apps be used?
Discussion

- What other methods of evaluation might they use to strengthen their argument that CLANdroid provides an effective method for detecting functionally similar apps?
Discussion

- How might this work be extended in the future?