Milestone 1: Mobile and Desktop Release Cycles

Kendall Bailey
Oregon State University

Hongyan Yi
Oregon State University

1. PROPOSED SOLUTION

We will collect the release notes or documentation of suitable applications that are a good mix of mobile applications, desktop applications, or mobile applications with a desktop applications sibling. Suitable applications will have:

1. A version history length of at least 15 releases
2. The date of each release
3. Release notes that mention bugs, features, enhancements, etc.

To collect the data we grab text from any release histories found on the application developers' website or application distributors' documentation. To get release history data from locations that block conventional copy and pasting (e.g., iTunes), we use optical character recognition (OCR) software (Capture2Text) to quickly transcribe the version history of applications.

Finally, we will process the text into a standard format that has yet to be determined. We will perform statistical analysis on this standardized dataset to best answer the three research questions below.

2. RESEARCH QUESTIONS

Our goal is to collect the data from 20 desktop applications, 20 mobile applications, and 20 sibling applications. If we maintain our minimum number of releases at 15 releases, this will give us at least 900 releases to use for analysis.

2.1 RQ1: Do mobile applications and desktop applications display different release cycle behavior?

Research question one will be used to establish a baseline for the behavior of both only mobile applications and only desktop applications.

We suspect that mobile applications have shorter release cycles than desktop applications. However, if the behavior of both types of applications is essentially identical, we plan to see if release cycles are different considering the purpose of the application (e.g., productivity, games, media creation, etc.).

2.2 RQ2: Do applications with "siblings" behave more like mobile or desktop apps?

Our second research question aims to determine whether the release cycles of sibling applications tend closer to desktop or mobile behavior. If the behavior follows neither we will further explore how and why the behavior differs.

We define sibling applications as applications that have the same name and are released by the same company for different platforms. For this research sibling applications refers to a mobile application (iOS or Android) and desktop application (OSX, Windows Vista/7/8, or Linux) pairing. An example of sibling applications would be OmniGraffle, which maintains both a iOS and OSX application of the same name with different version histories, though often version histories of sibling applications are not separated.

2.3 RQ3: Are bugs/features/enhancements more common in desktop, mobile, or sibling applications?

This research question would involve us classifying the contents of each release into four categories: bugs, new features, enhancements, and irrelevant (which includes things such as advertisements and developer contact information). We would then compare the frequency of each category per release and per time to see if any patterns exist.

2.3.1 RQ3 specific barriers

The third research question is still tentative because though we find it interesting, we worry that it is inherently flawed. Because there is a significant business pressure to create sibling applications on the market, there must be a reason that desktop applications were unable to migrate to mobile platforms. If the reason has to do with computational power, function, or code size, then it stands to reason that more complicated applications may have more bugs, making our comparison unbalanced.

Developers are also, whether intentionally or not, vague about the number of bugs or enhancements they address

in each release. We may consider every instance of "bug fixes" or "enhancements" as 2, but in reality we will never know the exact number.

Additionally, this process would be quite time consuming and would force us to reduce our initial goals for the number of applications down to around 12 for each category. Without research question 3 it is relatively easy to add new data to our corpus which could be done towards the end of the semester.

We originally looked to machine learning to discover if there was an option that would automatically classify our collected data, though while being easy to implement, a classifier requires a very large training set. If we were to exceed 800 classifications, we may be able to train a naive Bayes classifier to quickly add new data, but in the time remaining in the term we may not be able to classify that much data.

3. CHALLENGES

Had this project been executed 5 to 10 years ago it would be easier to find desktop only applications. Business pressure and interests have driven most companies that are willing or organized enough to post release notes to develop sibling applications . As a result we may have to reduce the number of applications we add to the corpus so as to not unbalance our data.

Another challenge we have already overcome is the taking of release histories from iTunes, which blocks copy and paste attempts. Web scraping was unsuccessful, the method of using an OCR application has allowed us to overcome that barrier. Though OCR is more tedious than other methods, iTunes maintains a large collection of release histories for each of the applications it sells. On the other hand, the Google Play Store only maintains the current version, meaning we have to search for developers to provide the information we need.

We had not factored in web applications to our original research plan. For the time being we plan to not consider web applications as either mobile or desktop applications and therefore not a part of our study.

4. RELATED WORK

Gall et al. [1] developed a product release database for telecommunication switching system that automatically generated release notes based on the source code. Like Gall et al., we plan to analyze software using the release histories provided by developers to derive information about software. However, instead of focusing on a single application, we will use information to compare applications on different platforms. In addition, the release notes we analyze are not generated based on the source code, but are disclosed by the developer.

Tsay et al. [2] paper presents experiences, including the tools, techniques and pitfalls, in mining open source release histories, and specifically shows that many factors make automating the retrieval of release history information difficult, such as the many sources of data, a lack of relevant standards and a disparity of tools used to create releases. These are the similar problems we've found in our research. The different thing from our recent research is that the projects quantity in this paper is very small, but number of versions of each project is very big. For example they collected 1579 distinct releases from 22 different open source projects, so on average 72 release versions per project. However, we focus on more types of software but less release version.

Yu [3] uses text data mining to extract information from change logs and release notes develop a mathematical model to represent software evolution based on predetermined keywords. Using the model, Yu traced the recurrences of issues across a single project. Similarly our research mines release notes to gather information about the nature of software bugs and features, but our attempts to use the data to compare different applications. Apart from version numbers and dates, our research steers away from filtering out information solely due to predefined keywords because positive spin, marketing terms, and buzz words interfere with simple collection of data. The inconsistencies in language across applications and companies is the reason we originally looked into naive Bayes classifiers to better assess the nature of statements in release notes.

5. REFERENCES

[1] H. Gall, M. Jazayeri, R. R. Kolsch, and G. Trausmuth, "Software evolution observations based on product release history," in *Proceedings of the International Conference on Software Maintenance 1997.*

[2] J. Tsay, H. K. Wright, and D. E. Perry, "Experiences mining open source release histories," in *Proceedings of the 2011 International Conference on Software and Systems Process.*

[3] L. Yu, "Mining change logs and release notes to understand software maintenance and evolution," vol. 12, no. 2.