Recognizing development behaviours corresponding to Android OS evolution and release life-cycle

Pavel Senin
CSDL laboratory, ICS Department
University of Hawaii at Manoa
Honolulu, Hawaii
senin@hawaii.edu

***Abstract*—Android OS is an open-source Linux-based operating system for mobile devices developed by Open Handset Alliance led by Google. We attempted to recognize recurrent behaviors within Android development process through data mining of software process artifact trails corresponding to eight years of continuous development. By temporal partitioning of development activity artifacts, their symbolic approximation and successive indexing, we have built dictionaries of observed patterns. Further application of unsupervised learning and density estimation techniques allowed us to recognize behavioral signals corresponding to Android release life cycle.**

***Keywords*-component; formatting; style; styling**

I. Introduction

As many other open-source projects, the Android OS was in the development for many years. There are about two millions of change records registered in the Android SCM by more than eleven thousands of contributors. The change metadata is shown to be a rich source of software process and social characteristics. For example it is possible to discover recurrent behaviors by using Fourier Analysis of change events [1], and relate them to the software product quality [2]. In this work I am extending previous research by introducing a framework for the temporal partitioning of the software change data and successive recurrent behaviors and knowledge discovery.

Previously it was found that software development as any other human activity could be successfully partitioned by the time of the day reflecting our lifestyle and habits [3] [4]. However external constraints, such as employment constraints, software release cycle [5] and management constraints [6] found to be able to significantly alter natural activity patterns. Furthermore, within the open-source projects with diverse development community scattered over the globe and following often undocumented development process the natural human activity cycles are often discarded as well as the development and release cycles are significantly altered. Thus, the only way to discover open-source software process is to analyze its artifacts trails such as SCM logs, bug-and issue tracking systems and mailing lists archives. Essentially these trails are event-series where every event is time-stamped and enclose a set of metadata. In order to discover recurrent processes (frequent activity patterns) these trails must be partitioned by the event generating activity and further mined for frequent time-patterns.

Time series data mining research is a well-established research field with an enormous wealth of methods, algorithms and data structures. To overcome the complexity of the temporal data it can be transformed into virtually any of the other data representations such as spectral, polynomial, wavelets, piece-wise, symbolic, etc. However, in spite of the fact that there are dozens of techniques for transform and comparison the overall complexity of data and the noise levels making data mining of temporal data difficult.

In this paper, I investigate the application of the Symbolic Aggregate Approximation [7] and the TF∗IDF statistics [8] to the problem of discovering recurrent behaviors in the Android software process.

My main findings include: (1) the SAX and TF∗IDF combination is applicable to the problem of partitioning of users and the software project development activities; (2) SAX and TF∗IDF application to the "*late night*" commit frequencies stream allows the pre-release and post-release pattern classes discovery (3) SAX and TF∗IDF application to the "*daytime*" commit frequencies stream allows the contributor social characteristics pattern classes discovery

II. Research question

The research question I am trying to resolve with this work is the discovery of recurrent behaviors within the Android SCM trails by SAX application. Previously a variety of time-series mining algorithms was applied to the problem of finding of periodicity and **recurrent behaviors** within the **software change artifacts trails** such as Linear predictive coding and cepstrum coefficients [9], Fourier Transform [1] and coding [10]. Two last cited papers show that corresponding techniques allow the discovery of the strong signal corresponding to the release of MySQL. Lin&Keogh [7] discovered and explored universal application power of Symbolic Aggregate Approximation to variety of time-series data.

In this work I explore the application of the SAX to the problem of discovering recurrent behaviors from software change artifacts trails.

To be precise in this writing I will check if I can see that there is a difference in recurrent behaviors before and after software release.

III. Experimental methods

A. Data collection and organization

Two XML files offered for the MSR challenge contain the most of the information obtainable from Google-hosted Android git repository as well as Google-hosted bug and issue tracking system. While the issues and comments trail contains nearly complete information, the change trail provided for a challenge contains only the high-level change information.

The thirteen data fields of the change trail XML file provide information about the revision tree, author and a committer identification, change message and affected targets. Since I am focusing on the mining of temporal patterns for inferring recurrent behaviors, in addition to the existing data I collected auxiliary information about change. By creating a local mirror and by iterating over existing commits hashes I was able to recover auxiliary data for 68% of existing commits. The rest of changes which is about 32% of total change information belongs to legacy projects and is unrecoverable due to the changes in Android repository.

For every recoverable change record I collected a summary of added, modified or deleted files as well as a summary about LOC changes: added, modified or deleted lines. All this information was stored in the Trajectory database backend. Main tables of the database correspond to change and issue events; these tables accompanied with change targets tables and issue comments as well as tables for contributors. Overall, the database was normalized and optimized for the fast retrieval of change and issue information using SQL language.

Following the previous research I partitioned the change trails by the time of the day and by natural time periods of one week, two weeks and a month. These data was aggregated into the intermediate representation within MySQL database.

B. Symbolic approximation and indexing

SAX indexing depends on the three parameters which are required. First parameter is the sliding window, second is the PAA approximation size and third is the SAX Alphabet size.

In this work I have selected three sizes for sliding window: 7 days, 14 days and 30 days as these represent an intuitive and logical intervals of a week, two weeks and a month. For the PAA reduction I choose 4 steps for 7 days window, 6 steps for a bi-weekly interval, and 10 steps for a monthly window. Finally the alphabet, I choose 3 letters for weekly window, and 5 letters for bi-weekly and monthly windows. All these are summarized in the Table I.

Table I
Sliding window, PAA and alphabet size choices.

Sliding window size	PAA size	Alphabet size
one week (7 days)	4	3
two weeks (14 days)	6	5
month (30 days)	10	5

By applying SAX to the pre-aggregated data I have obtained their symbolic temporal representation. Do I need a figure explaining SAX transform here?

C. Token-based distance metrics

For the experiments I have selected three similarity metrics. First one is based on the SAX min distance and the Euclidean distance which applied to vector of tokens sorted by frequency observed in the aggregated symbolic stream corresponding to selected stream class, aggregation parameters, user, project and time-interval of the interest.

$$D(S,T) = \sqrt{\sum SAXdist(s_i, t_i)^2} \quad (1)$$

where $SAXdist$ is the the SAX distance based on the Normal alphabet.

As an alternative I have tried the Jaccard similarity between two sets S and T which is simply

$$J_\delta(S,T) = \frac{|S \cup T| - |S \cap T|}{|S \cup T|} \quad (2)$$

and the $TF * IDF$ similarity or which defined as a dot product

$$TFIDF(S,T) = \sum_{\omega \in S \cap T} V(\omega, S) \cdot V(\omega, T) \quad (3)$$

where

$$V(\omega, S) = \frac{V'(\omega, S)}{\sqrt{\sum_{\omega'} V'(\omega, S)^2}} \quad (4)$$

is a normalization of $TF*IDF$ (product of token frequency and inverse document frequency):

$$V'(\omega, S) = \log(TF_{\omega,S} + 1) \cdot \log(IDF_\omega) \quad (5)$$

where IDF_ω is a measure of the general importance of the pattern among all users

$$IDF_\omega = \frac{|D|}{DF(\omega)} \quad (6)$$

where $|D|$ is cardinality of D - the total number of users, and $DF(\omega)$ is the number of users having ω pattern in their activity set.

IV. Clustering

While the application of the SAX distance and Jaccard similarity yields a single number allowing the use of the convenient clustering libraries such as $hclust()$ and $kmeans()$ of R for manipulation with sparse vectors produced by the $TF * IDF$ application I used $sparcl()$ R package for hierarchical clustering and my own implementation of $k - means$ clustering.

V. Results

A. Kernel-OMAP life cycle patterns discovery

I arbitrary selected the Android kernel-OMAP as one of the large Android OS sub-projects. OMAP is a proprietary system on chips (SoCs) for portable and mobile multimedia applications based on general-purpose ARM architecture processor provided by Texas Instruments.

As a training set for building dictionaries of pre and post-release patterns I choose three Android releases: *Android 1.0*, *Android 1.5* and *Android 2.0*. For each release in the set I selected four weeks before the release as *pre-release* and four weeks after release as *post-release* training intervals. As an additional constraint for this experiment I have selected contributors affiliated with $@android.com$ e-mail domain and a telemetry stream of $added\_lines$. After pre-generated patterns retrieval with SQL query I applied hierarchical clustering based on $TF * IDF$ similarity as a sanity test. The almost perfect clustering picture 1 indicates that there are significant differences in the pre-release and post-release weekly behaviors among selected contributors.

While hierarchical clustering is a good sanity test, the K-means clustering is the most commonly used data mining clustering algorithm [11], so for completeness of the test I performed k-means on the symbolic representation of data using $TF*IDF$ similarity. Algorithm converged after three iterations separating pre- and post-release dictionaries.

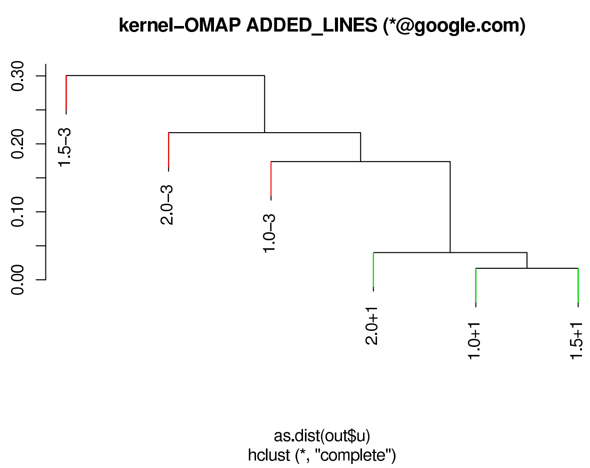

Figure 1. Hierarchical clustering of pre- and post- commit patterns. Labels with -3 (red) indicate pre-release, labels with +1 (green) post-release.

VI. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

The authors would like to thank... more thanks here

References

[1] A. Hindle, M. W. Godfrey, and R. C. Holt, “Mining recurrent activities: Fourier analysis of change events,” in *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE, May 2009, pp. 295–298. [Online]. Available: http://dx.doi.org/10.1109/ICSE-COMPANION.2009.5071005

[2] J. Eyolfson, L. Tan, and P. Lam, “Do time of day and developer experience affect commit bugginess?” in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR ’11. New York, NY, USA: ACM, 2011, pp. 153–162. [Online]. Available: http://dx.doi.org/10.1145/1985441.1985464

[3] J. A. Leech, W. C. Nelson, R. T. Burnett, S. Aaron, and M. E. Raizenne, “It’s about time: A comparison of Canadian and American timeactivity patterns,” *Journal of Exposure Science and Environmental Epidemiology*, vol. 12, no. 6, pp. 427–432, Nov. 2002. [Online]. Available: http://dx.doi.org/10.1038/sj.jea.7500244

[4] I. Gorton and S. Motwani, “Issues in co-operative software engineering using globally distributed teams,” *Information and Software Technology*, vol. 38, no. 10, pp. 647–655, Jan. 1996. [Online]. Available: http://dx.doi.org/10.1016/0950-5849(96)01099-3

[5] M. M. Lehman, “Programs, life cycles, and laws of software evolution,” *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980. [Online]. Available: http://dx.doi.org/10.1109/PROC.1980.11805

[6] E. Yourdon, *Death March (2nd Edition)*, 2nd ed. Prentice Hall, Nov. 2003. [Online]. Available: http://www.worldcat.org/isbn/013143635X

[7] J. Lin, E. Keogh, L. Wei, and S. Lonardi, “Experiencing SAX: a novel symbolic representation of time series,” *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 107–144, Oct. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10618-007-0064-z

[8] T. Roelleke and J. Wang, “TF-IDF uncovered: a study of theories and probabilities,” in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR ’08. New York, NY, USA: ACM, 2008, pp. 435–442. [Online]. Available: http://dx.doi.org/10.1145/1390334.1390409

[9] G. Antoniol, V. F. Rollo, and G. Venturi, “Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories,” in *Proceedings of the 2005 international workshop on Mining software repositories*, ser. MSR ’05, vol. 30, no. 4. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1145/1082983.1083156

[10] A. Hindle, M. W. Godfrey, and R. C. Holt, “Release Pattern Discovery via Partitioning: Methodology and Case Study,” in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007. [Online]. Available: http://dx.doi.org/10.1109/ICSEW.2007.181

[11] *Initialization of Iterative Refinement Clustering Algorithms*, 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.3469