Recognizing recurrent development behaviours corresponding to Android OS release life-cycle

Pavel Senin
Collaborative Software Development Laboratory
Information and Computer Sciences 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. Its SCM log was selected as a research subject for the 2012 MSR Challenge. I attempted to apply a novel data mining technique based on SAX approximation and indexing of time-series with $TF * IDF$ weights in order to discover recurrent behaviors within the Android OS development process.**

By mining software process artifact trails corresponding to OMAP kernel development, I was able to discover recurrent behaviors in the "new code lines dynamics" before and after release. By building a classifier upon these behaviors I was able to successfully recognize pre- and post-release behaviors within the same and similar sub-projects of Android OS.

***Keywords*-software process, recurrent behaviors, data-mining**

I. Introduction

As with many other large open-source projects, Android OS has been in the development for many years. Android is "an open-source software stack for mobile phones and other devices", http://source.android.com/ which is based on the Linux 2.6 monolithic kernel. Development of Android was begun by Android Inc., the small startup company. In 2005, the company was acquired by Google which formed the Open Handset Alliance - a consortium of 84 companies which announced the availability of the Android Software Development Kit (SDK) in November 2007. The Android OS code is open and released under the Apache License.

Git is used as a version control system for Android and the source code is organized into more than 200 of sub-projects by function (kernel, UI, mailing system, etc.) and underlying hardware (CPU type, bluetooth communication chip, etc.). There are about two million change records registered in the Android SCM by more than eleven thousands of contributors within an eight year span.

By the large body of previous research, it has been shown that change metadata is a rich source of software process and developers' social characteristics. The ability to discover recurrent behaviors with Fourier Analysis of change events is explained in [1] along with other work, such as [2] that relates activity time patterns and software product quality. Thus, potentially, it is possibile to relate recurrent behaviors to software product quality and to software process efficiency. The first part of a toolkit aiding such a research is an efficient mechanism to discover recurrent behaviors modulated by social and project-related constraints. In this paper I extend previous research by introducing a universal framework for temporal partitioning and mining of software change artifacts and evaluate this fremework on the Android SCM data.

II. Contribution

To the best of my knowledge, this work is the first attempt to study the applicability of symbolic aggregate approximation and term frequency–document frequency weight statistics to the mining of software process artifacts. This methodology has a number of advantages. First of all, SAX facilitates significant reduction of the large complexity (dimensionality and noise) of temporal artifacts and opens the door to application of a phetora of strings and text-mining algorithms. In addition, the TF$*$IDF statistics provides an efficient mechanism for discrimination of the signal by ranking text data. Finally the third component I have used - the relational database - facilitates efficient data slicing, indexing, and retrieval.

As an example of a possible data-mining workflow demonstrating the resolving power and correctness of the approach, I present a case study of building a classifier for pre- and post-release recurrent behaviors. Whereas this classifier demonstrates a good performance within the project it was trained on with less than 20% miss-classification, it has less than 15% miss-classification rate in similar Android OS kernel sub-projects.

III. Motivation

Software development is a human activity resulting in a software product. The amount of time and effort needed to complete a software project and the quality of the final product affected by software process. Thus, studying software processes is one of the important areas of software engineering.

Previously it was found that software development, as many other human activities, could be successfully partitioned by the time of the day reflecting our lifestyle and habits [3] [4]. However external constraints, such as employment and management constraints [5], software release cycle [6] have been found altering natural activity patterns significantly. Furthermore, within open-source projects with a diverse development community scattered over the globe and often following undocumented development process [7], natural human activity cycles are often discarded and development and release cycles are significantly altered. Thus, the only feasible way to discover an 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 time-stamped event has an attached set of metadata. However the complexity of this data and the precision of the process recall impose a great challenge for researchers.

These challenges are not new to the data-mining community and an enormous wealth of methods, algorithms and data structures have been developed to address these issues. While some of these approaches were already implemented within the MSR framework such as finding of trends, periodicity and recurrent behaviors through the linear predictive coding and cepstrum coefficients [8], Fourier Transform [1] and coding [7], many are yet to be tried.

In this paper, I investigate the application of Symbolic Aggregate Approximation [9] and the TF$*$IDF statistics [10] to the problem of discovering recurrent behaviors from software process artifacts with application to Android SCM data.

IV. Research question

In this exploratory work I am investigating the applicability of Lin&Keogh [9] symbolic approximation technique to the discovery of recurrent behaviors from SCM trails of Android OS. The research questions I am addressing are:

- Which kinds of SCM data need to be collected for such analyzes?
- What is the optimal approach to data representation and a data storage configuration?
- Which partitioning (slicing) is appropriate and which set of parameters should one use for SAX approximation?
- Which distance metrics serves best for measuring similarity and dissimilarity?
- What is the general mining workflow?

V. Experimental setup and methods

A. Data collection and organization

Two XML files were offered for the MSR challenge. These contain the most of the information obtainable from Google-hosted git source code repository as well as from bug and issue tracking system. While the issues and comment XML file contains nearly all information available in repository, the change XML provided for this challenge contains only a fraction of all of the 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 have collected additional auxiliary data about change. By creating a local mirror and by iterating over existing commit hashes I was able to recover the auxiliary data for 68% of existing commits. The remaining 32% of change information is irretrievable and belongs to legacy projects or is lost and unrecoverable due to the changes in Android repository.

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

B. Temporal data partitioning

Following the previous research [2], I have partitioned the change trails by the time of the day using time windows of

- Full day, 12AM - 12AM
- Late night, 12AM - 04AM
- Early morning, 04AM - 08AM
- Day, 08AM - 05PM
- Night, 05PM - 12AM

then aggregated within these windows values for commits, added/edited/deleted lines and targets. By having such a table it takes a fractions of a second to retrieve a summary of early morning added lines for January 2007 for a particular sub-project and a group of contributors having emails with a domain "...@ibm.com".

C. Symbolic approximation and indexing

SAX algorithm requires three input parameters where first is the sliding window size, second is the PAA approximation size, and the third parameter is the SAX Alphabet size.

In this work I experimented with three sizes for sliding window reflecting natural intervals: a week (7 days), two weeks (14 days) and a month (30 days). For the PAA reduction I chose size 4 for 7 days window, size 6 for a bi-weekly interval, and 10 PAA steps for a monthly window. Finally, for the alphabet, I used 3 letters for weekly window, 5 letters for bi-weekly and 7 for monthly windows.

By applying SAX to the pre-aggregated streams I obtained their symbolic representation which I stored in a MySQL database organized and indexed to allow fast retrieval of

symbolic data for a specific SAX parameters set, a project, a contributor, or a time-interval or any combination of these fields .

D. Token-based distance metrics

For the experiments I selected three similarity metrics which were used to compare vectors of SAX words frequencies for selected entities or their sets. Each of these vectors is two-dimensional and consists of the pairs $\{\langle word\rangle, \langle word\ frequency\rangle\}$

For distance, I experimented with a weighted by SAX Euclidean distance between frequencies of common to two sets words:

$$D(S,T) = \sqrt{\sum_{S\cap T}(SAXdist(s_i,t_i) * \|F_{s_i} - F_{t_i}\|)^2} \quad (1)$$

The second distance I tried is the Jaccard similarity coefficient between two sets S and T which is simply

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

The third distance is the $TF * IDF$ similarity 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.

VI. Clustering

As a universal tool for the exploration of derived pattern dictionaries through their partitioning, and for assessment of the metrics' performance, I used hierarchical clustering. The k-means clustering was used in the validation of the class assignment and for general assessment of the validity of the approach.

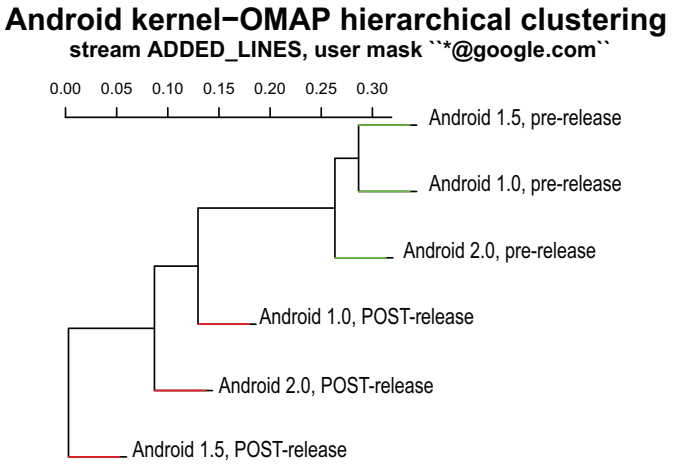

Figure 1. Hierarchical clustering of pre- and post- release patterns.

VII. 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 chose three Android releases: *Android 1.0*, *Android 1.5* and *Android 2.0*. For each of these I generated SAX words dictionaries corresponding to four weeks before the release - *pre-release*, and to four weeks after release - *post-release*.

After a number of clustering experiments, from three selected metrics only $TF * IDF$ demonstrated an acceptable performance when applied to weekly aggregated time-intervals of one month before the release and one month after release.

As an additional constraint for this experiment, I put the restriction on contributors, selecting affiliated with $google.com$ e-mail domain only, and on the telemetry stream of $added\_lines$. The almost perfect clustering picture (Figure 1) obtained with hierarchical clustering and $TF * IDF$ similarity measure indicates that there are significant differences in the pre- and post-release weekly behaviors of contributors in selected time-windows.

While hierarchical clustering is a good sanity test for the data exploration, the performance of K-means clustering is much more valuable [11]. I performed k-means on the symbolic representation of data using $TF * IDF$ statistics and Euclidean distance. The algorithm converged after two iterations separating pre- and post-release dictionaries with a single mismatch for the Android 2.0 pre-release.

By using centroids of two resulting clusters as a basis for pre- and post-release patterns I tested the classifier on the

Table I
PATTERNS OBSERVED WITHIN PRE- AND POST-RELEASE WINDOWS, THEIR $TF * IDF$ WEIGHTS AND SAMPLE CURVES.

release	”bbac”	”abca”	”babc”	”bbba”	”bcaa”	”bcbb”	”ccaa”	”cbaa”	”bbcb”	”bbbb”	”bbbc”
post-2.0	0.63	0	0.63	0	0	0	0	0.39	0.24	0.06	0
post-1.0	0	0.93	0	0	0	0	0	0	0	0.09	0.36
post-1.5	0	0	0	0	0	0	0	0	0.79	0.61	0
pre-1.5	0	0	0	0.23	0.23	0.91	0	0.14	0.18	0	0.09
pre-2.0	0	0	0	0	0	0	0	0	0	1	0
pre-2.0	0	0	0	0	0	0	0.79	0	0	0.08	0.61
Sample curves corresponding to patterns											

Table II
PRE- AND POST-RELEASE DEVELOPMENT PATTERNS CLASSIFICATION RESULTS FOR KERNEL-OMAP.

Release	Classification
1.6 -pre	-
1.6 -post	+
2.2 -pre	+
2.2 -post	+
1.1 -pre	+
1.1 -post	+
2.3 -pre	+
2.3 -post	+

Release	Classification
beta -pre	+
beta -post	+
2.0.1 -pre	+
2.0.1 -post	-
2.1 -pre	+
2.1 -post	+
2.2.1 -pre	+
2.2.1 -post	-

rest of Android releases for which kernel-OMAP remained an active project (some of the pre- and post- release interval contained none or only trivial patterns making them ineligible for classification). This classifier was able to successfully classify more than 81% of pre- and post-release behaviors (Table II). When applied to the similar project - kernel-TEGRA - it demonstrated the error rate less than 15%.

The classifier demonstrated a weak, almost random performance on other sub-projects. However, when re-trained on the platform-external-bluetooth-bluez, its performance on other bluetooth sub-projects such as platform-external-bluetooth-glib, platform-external-bluetooth-hcidump and platform-system-bluetooth recovered to 20% missclassification.

VIII. DISCUSSION

The presented approach and workflow employs two novel techniques in order to discover and rank recurrent behaviors from software process artifact trails. While the approach demonstrates satisfactory performance, the interpretation of the captured behaviors requires more work. The discovered behaviors are organized in Table I by their occurence: the first three columns belong to the post-release time-window while the four next columns belong to pre-release time-window, the rest are the behaviors observed in both. It appears that during pre-release most of the added lines within a week fall on the Monday and Tuesday, whereas during post-release time, most of the lines are added during the end of the week and the week-end. While an explanation of these findings requires an additional study to be made, one of the interpretations of such behavior could be based on the contributors employment profile. For example, if the coding activity of developers paid to work on Android (thus mostly commit during working days) has fallen below the activity of developers working on their own volition (who commit mostly off business hours), which in turn could be a consequence of removing of a pre-release code-freeze, or that the paid developers switched in post release time to design, documentation or bug-fixing activities.

IX. ACKNOWLEDGMENT

I thank to Philip Johnson for his time, useful discussions and comments.

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] N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann, “The national human activity pattern survey (NHAPS): a resource for assessing exposure to environmental pollutants.” *Journal of exposure analysis and environmental epidemiology*, vol. 11, no. 3, pp. 231–252, 2001. [Online]. Available: http://dx.doi.org/10.1038/sj.jea.7500165

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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

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