Sanity check and the Trajectory workflow test

Pavel Senin

February 22, 2012

Abstract

This document has dual purpose. First of all, by writing it I am following the common Trajectroy analysis workflow reviewing all steps and debugging for errors. The second purpose is to compile a primer of Trajectory-aided SCM data mining.

1 Outline

At the beginning I will discuss the research question.

First section 3 explains the data aggregation within the Trajectory DB. I discuss two issues: first is the data organization and its curation through preliminary analysis. Second issue is the pre-aggregation of data by time intervals (chunks) for future indexing.

In the second section 4 I address the Symbolic aggregation workflow. I cover the data specificity, SAX parameters section and the index organization.

In the third section 5 I address the mining question.

2 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 [1], Fourier Transform [3] and coding [2]. Two last cited papers show that corresponding techniques allow the discovery of the strong signal corresponding to the release of MySQL. Lin&Keogh [4] 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 behaviours before and after software release.

2.1 Definitions

Let's define

- the **sub-series of the *length* W, S_W** as the continous interval extracted from the larger time-series, analogous to sub-string;

- the **SAX representation of sub-series S_W by *PAA of size* P and an *Alphabet of size* A** as the symbolic representation of the sub-series S_W obtained by PAA and SAX transforms.

For example, consider the sub-series of size 7 from time-series of daily commits S_7 by some contributor: $\{0, 5, 2, 3, 4, 3, 0\}$. The contributor committed zero times at Sunday, five times Monday and so on. By applying PAA transform of length 3 and SAX transform with the Normal alphabet of size 3 we will get a P_3A_3 representation of S_7 which will be *bbb*. Note how choice of the week format (starting from Sunday or Monday) affects the result. If we will do the same transform for $\{5, 2, 3, 4, 3, 0, 0\}$ the result will be *cca*...

2.2 Methodology

I define series and events of interest. By applying the transform defined by W - sliding window size, P - paa length and A - alphabet size I extract all possible strings from all of the series of interest. By comparing and mining of these dictionaries I will discover recurrent behaviors.

2.3 Behaviors taxonomy

I propose to use and will explore the application of *"busy weekend"* pattern, *"consistent development"* and *"sporadic development"* patterns, and a *"long night"* pattern for the discovery of a software release.

3 Aggregating commits statistics in TrajectoryDB

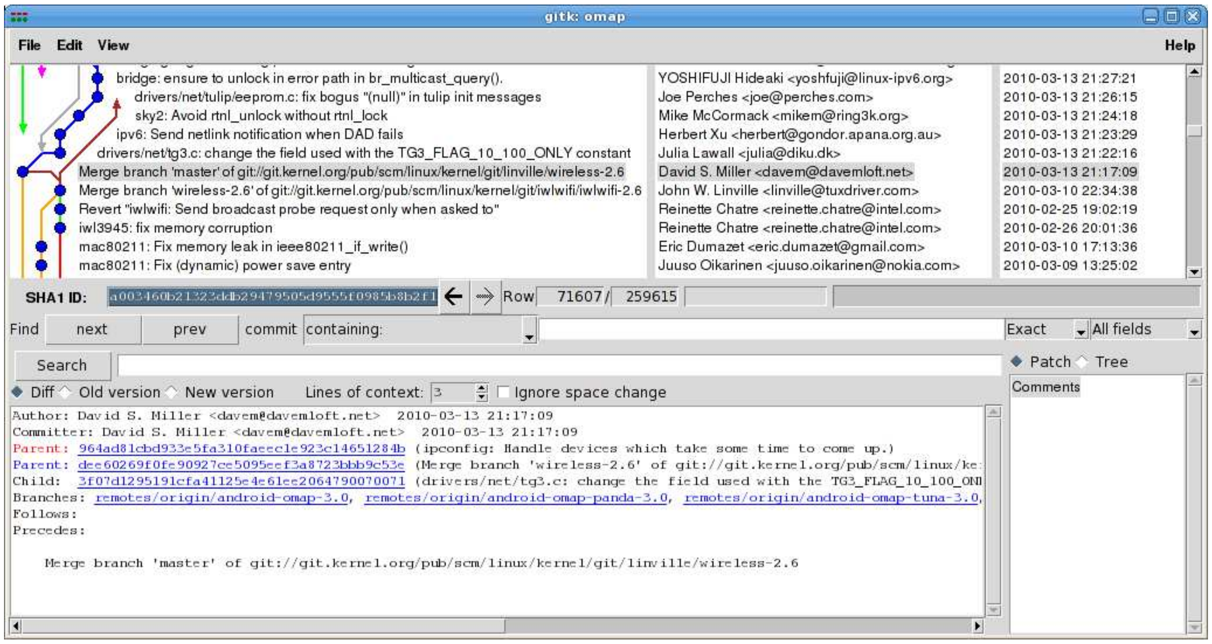
The raw statistics from an SCM system is stored within the TajectoryDB. There is information about every change, its impact on the source code tree and an auxiliary data about projects and contributors.

For the first test I have randomly selected a contributor with id *153* (davem@davemloft.net) and a project with id *18* (android-kernel-omap). As a daytime interval I have chosen the *nt* interval corresponding to night hours from *5PM to 00AM.* Let's the see records in the change database first by querying it with the query from listing 1:

Listing 1: Data summary retrieval SQL query

```
select c.id, c.author_date, c.subject, sum(c.added_files) ta,
 sum(c.edited_files) te, sum(c.removed_files) td, sum(c.added_lines) la,
 sum(c.edited_lines) le, sum(c.removed_lines) ld from android_change c
 where c.author_id=153 and c.project_id=18
 AND c.author_date between "2010-03-01" AND "2010-04-01"
 and date_format(c.author_date,'%H') between 17 and 23
 group by c.id order by c.author_date;
```

Here five of fourteen commits: *825982, 825288, 825287, 824847, 824620* are in fact merge commits. For these I am unable to collect any of the statistics. Thus there is a question - *should I count these merege commits (which have zero statistics) as an activity or should I skip commits which have zero changes attached?* I think I should include these into analyses since even if the change appears to be empty it anyway requires some of the developer's attention to actually make a merge decision, to perform the merge, and to check its results. I double-checked the information of MSR XML file and my pipeline by checking out OMAP repository locally and traversing the tree, see Figure 1. It seems to be that Git doesn't have any data about the merge. Maybe it can be recalled by traversing the tree up or down, but this is not implemented.



```
bridge: ensure to unlock in error path in br_multicast_query().   YOSHIFUJI Hideaki <yoshfuji@linux-ipv6.org>   2010-03-13 21:27:21
drivers/net/tulip/eeprom.c: fix bogus "(null)" in tulip init messages   Joe Perches <joe@perches.com>   2010-03-13 21:26:15
sky2: Avoid rtnl_unlock without rtnl_lock   Mike McCormack <mikem@ring3k.org>   2010-03-13 21:24:18
ipv6: Send netlink notification when DAD fails   Herbert Xu <herbert@gondor.apana.org.au>   2010-03-13 21:23:29
drivers/net/tg3.c: change the field used with the TG3_FLAG_10_100_ONLY constant   Julia Lawall <julia@diku.dk>   2010-03-13 21:22:16
Merge branch 'master' of git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-2.6   David S. Miller <davem@davemloft.net>   2010-03-13 21:17:09
Merge branch 'wireless-2.6' of git://git.kernel.org/pub/scm/linux/kernel/git/iwlwifi/iwlwifi-2.6   John W. Linville <linville@tuxdriver.com>   2010-03-10 22:34:38
Revert "iwlwifi: Send broadcast probe request only when asked to"   Reinette Chatre <reinette.chatre@intel.com>   2010-02-25 19:02:19
iwl3945: fix memory corruption   Reinette Chatre <reinette.chatre@intel.com>   2010-02-26 20:01:36
mac80211: Fix memory leak in ieee80211_if_write()   Eric Dumazet <eric.dumazet@gmail.com>   2010-03-10 17:13:36
mac80211: Fix (dynamic) power save entry   Juuso Oikarinen <juuso.oikarinen@nokia.com>   2010-03-09 13:25:02

SHA1 ID: a003460b21323ddb29479505d9555f0985b8b2f1   Row 71607 / 259615
```

```
Author: David S. Miller <davem@davemloft.net>  2010-03-13 21:17:09
Committer: David S. Miller <davem@davemloft.net>  2010-03-13 21:17:09
Parent: 964ad81cbd933e5fa310faeec1e923c14651284b (ipconfig: Handle devices which take some time to come up.)
Parent: dee60269f0fe90927ce5095eef3a8723bbb9c53e (Merge branch 'wireless-2.6' of git://git.kernel.org/pub/scm/linux/ke
Child:  3f07d1295191cfa41125e4e61ee2064790070071 (drivers/net/tg3.c: change the field used with the TG3_FLAG_10_100_ONI
Branches: remotes/origin/android-omap-3.0, remotes/origin/android-omap-panda-3.0, remotes/origin/android-omap-tuna-3.0,
Follows:
Precedes:

    Merge branch 'master' of git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-2.6
```

Figure 1: The GUI git viewer screenshoot illustrating commit statistics.

It is nice to see that TrajectoryDB has all the data immediately available in Git, however the result in the Table 3 is not very suitable for further analyses because changes retrieved are sequential. I decided to use MySQL functions to aggregate data. The query used within the trajectory workflow shown at the listing 2.

Table 1: Result of running the query 1 against the TrajectoryDB (the column names are truncated for illustration purposes).

id	author_date	subj	ta	te	td	la	le	ld
827962	2010-03-03 17:08	`sparc64: Kill off old sys_perfctr system call and state.`	0	12	0	0	18	195
827955	2010-03-03 18:06	`sparc64: Make prom entry spinlock NMI safe.`	0	1	0	0	7	0
827241	2010-03-05 22:41	`timbgpio: fix build`	0	1	0	1	0	0
826446	2010-03-10 22:05	`uartlite: Fix build on sparc.`	0	1	0	0	5	0
825982	2010-03-13 21:17	`Merge branch 'master' of git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-2.6`	0	0	0	0	0	0
825784	2010-03-15 23:23	`e100: Fix ring parameter change handling regression.`	0	1	0	0	1	0
825679	2010-03-16 22:37	`bridge: Make first arg to deliver_clone const.`	0	1	0	0	4	0
825678	2010-03-16 22:40	`sunxvr1000: Add missing FB=y depenency.`	0	1	0	0	1	0
825288	2010-03-20 22:41	`Merge branch 'vhost' of git://git.kernel.org/pub/scm/linux/kernel/git/mst/vhost`	0	0	0	0	0	0
825287	2010-03-20 23:24	`Merge branch 'master' of master.kernel.org:/pub/scm/linux/kernel/git/davem/net-2.6`	0	0	0	0	0	0
824847	2010-03-25 19:48	`Merge branch 'master' of git://git.kernel.org/pub/scm/linux/kernel/git/kaber/nf-2.6`	0	0	0	0	0	0
824789	2010-03-26 18:23	`Revert r8169: enable 64-bit DMA by default for PCI Express devices (v2)"`	0	1	0	4	3	8
824627	2010-03-29 22:08	`sparc64: Properly truncate pt_regs framepointer in perf callback.`	0	1	0	0	1	0
824620	2010-03-29 22:50	`Merge branch 'master' of git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-next-2.6`	0	0	0	0	0	0

Listing 2: Data summary retrieval SQL query with aggergation by date

```
select date_format(c.author_date, '%Y-%m-%d') as day, count(distinct(c.id)) as commits,
 sum(c.added_files) as added_files, sum(c.edited_files) as edited_files, sum(c.
     removed_files) as removed_files,
 sum(c.added_lines) as added_lines, sum(c.edited_lines) as edited_lines, sum(c.
     removed_lines) as removed_lines
 FROM android_change c
 where c.author_id=153 and c.project_id=18
 and c.author_date between "2010-03-01" AND "2010-04-01"
 and date_format(c.author_date,'%H') between 17 and 23
 group by date_format(c.author_date, '%Y%m%d');
```

Table 2: Result of running the query from Listing 2 against the TrajectoryDB.

day	commits	added_fs	edited_fs	rm_fs	added_ls	edited_ls	rm_ls
2010-03-03	2	0	13	0	0	25	195
2010-03-05	1	0	1	0	1	0	0
2010-03-10	1	0	1	0	0	5	0
2010-03-13	1	0	0	0	0	0	0
2010-03-15	1	0	1	0	0	1	0
2010-03-16	2	0	2	0	0	5	0
2010-03-20	2	0	0	0	0	0	0
2010-03-25	1	0	0	0	0	0	0
2010-03-26	1	0	1	0	4	3	8
2010-03-29	2	0	1	0	0	1	0

This looks much better - for every day I have aggregated data about number commits, added, edited, deleted files (*added_fs, edited_fs, rm_fs*), and lines (*added_ls, edited_ls, rm_ls*).

This data is digested within the Trajectory further - I save it the table *time_patterns* which keeps aggregated data along with the daytime tags and labels. This auxiliary information is being used for indexing and data mining in order to reduce the size of stored data and improve turn-around time of workflow. Here I use the *nt* tag for the "night commits", 5PM-12AM. By running the *SELECT* query from Listing 3 I am verifying that data is stored as it was - without any modification.

Listing 3: Data summary from time_patterns table

```
SELECT * FROM time_patterns WHERE project_id = 18
AND author_id=148 AND UPPER(dtag)=UPPER("NT")
AND day between "2010-03-01" AND "2010-04-01"
ORDER BY day ASC;
```

The repository trail data stored in this fashion considerably save the computation time. Running aggregation by time slot and summarizing changes as in query from Listing 2 could take a very long time over large repositories, whether running single *SELECT* over indexed by user, project, dtag and day *time_patterns* table takes a fraction of a second.

4 Indexing aggregated commits statistics with SAX

This step is essential for the reduction of the data complexity and for enabling of the data mining. The SCM trails data is getting transformed into sub-series chunks at first. These sub-series are converted

Table 3: Result of running the SELECT query from Listing 3 against the TrajectoryDB.

day	dtag	commits	t_ad	t_ed	t_dlt	l_ad	l_ed	l_dlt
2010-03-03	nt	2	0	13	0	0	25	195
2010-03-05	nt	1	0	1	0	1	0	0
2010-03-10	nt	1	0	1	0	0	5	0
2010-03-13	nt	1	0	0	0	0	0	0
2010-03-15	nt	1	0	1	0	0	1	0
2010-03-16	nt	2	0	2	0	0	5	0
2010-03-20	nt	2	0	0	0	0	0	0
2010-03-25	nt	1	0	0	0	0	0	0
2010-03-26	nt	1	0	1	0	4	3	8
2010-03-29	nt	2	0	1	0	0	1	0

into the symbolic representation and indexed at the next step.

4.1 Data distribution

Before explaining the Trajectory workflow with symbolic approximation and indexing, I would like to double-check some SAX preliminaries. First of all, when defining SAX alphabets, Lin&Keogh [4] assumed that the normalized series are following Gaussian (normal) distribution. Within their work they concluded that the most of the real-life timeseries (*"large family of the time series data in our disposal"*) are approximately normally distributed, which, unfortunately, is not the case of the Android SCM data in my disposal. I have tested the distribution of values from OMAP data. It does not follow the Normal distribution, see 6, thus I assume that the efficiency of SAX approximation will deteriorate. At this point I am unable to estimate to which degree it may happen since there is no relevant research I know about. However, according to Lin&Keogh, the correctness of the SAX algorithm is unaffected by distribution. The lower-bounding property of the SAX distance measure guarantees correctness of results.

4.2 Symbolic approximation

The data from the Table 3 is converted into SAX series on the next step. This process consists of five consecutive steps:

1. Three parameters are selected: the sliding window size W, the PAA size P and the alphabet size A. These parameters I store as the characteristic tag of the timeseries in form $WxPxAx$ where x is the corresponding parameter value.

2. By using sliding window, timeseries is chopped by sub-series each of which has the length of W. There will be $L_{series} - W$ of such sub-series (I use L as an abbreviation of length).

3. Each of the sub-series is getting "normalized by energy". The purpose of this step is to equal all the sub-fragments by amplitude but preserve their shape configuration.

Figure 3: Result of the conversion of data from 3 into SAX strings.

string_id	string	date
72	cba	2010-03-01
65	bab	2010-03-03
66	bbb	2010-03-05
59	abb	2010-03-06
66	bbb	2010-03-07
67	bbc	2010-03-09
60	abc	2010-03-10
63	bcb	2010-03-11
62	bca	2010-03-12
66	bbb	2010-03-13
70	cab	2010-03-14
...	...	...
66	bbb	2010-03-30
63	bcb	2010-03-31
66	bbb	2010-04-01

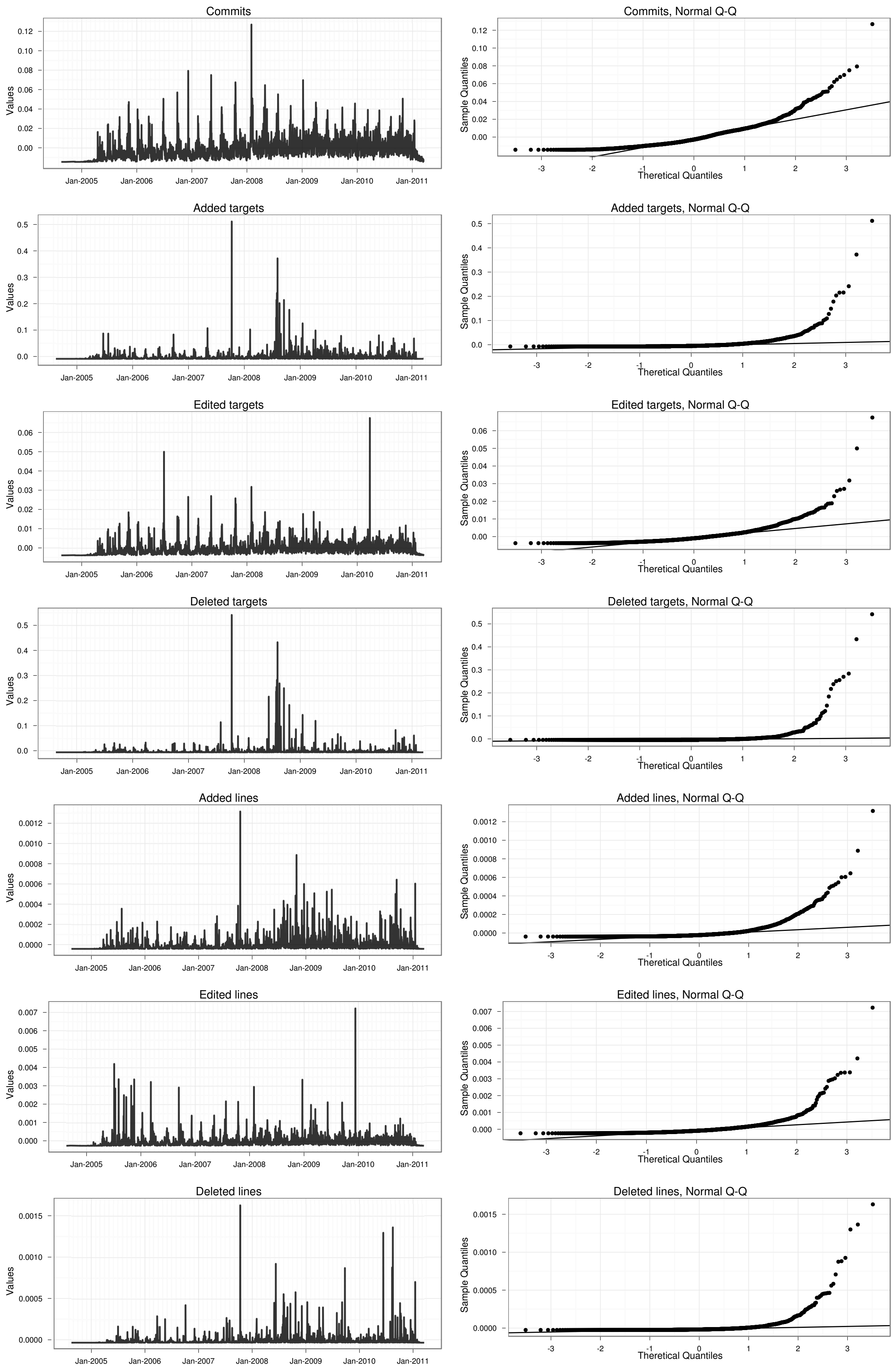


Figure 2: The distribution of time-series values after normalization. OMAP-kernel.

4. Each of the normalized pieces is approximated by use of "*P*iecewise *A*ggregate *A*pproximation". It works by slicing original time-series into P pieces and computing and assigning values for these slices (mean of all points within the piece).

5. Real-valued PAA series is converted into the string by using SAX. These strings will consist only of letters of pre-selected alphabet.

The Figure 3 illustrates the conversion of the discussed here data into symbolic result. This result are also shown at the Figure 4. Figure 5 displays the picture of overall nightly commit trail for the user *#153* (davem@davemloft.net) and a project *#18* (android-kernel-omap).

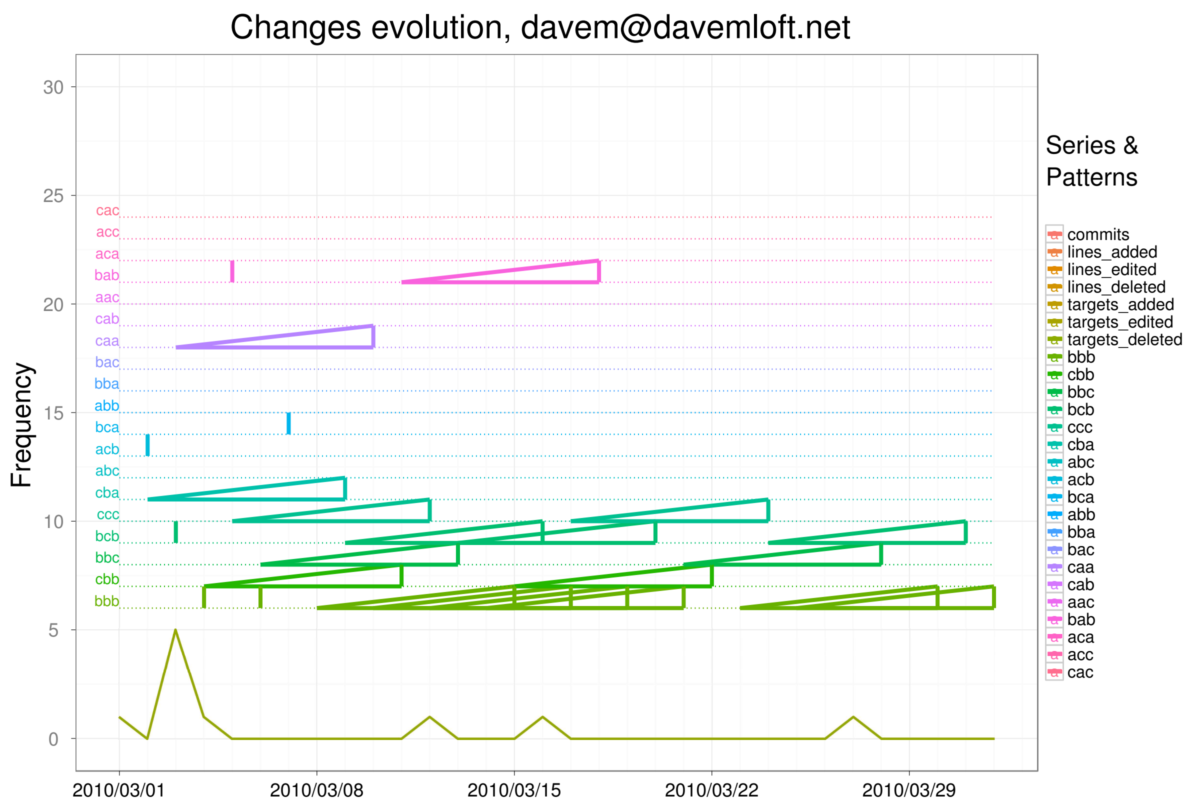


Figure 4: The illustration of the temporal distribution of patterns from Figure 3. Here I superimpose patterns over the commit frequency plot by using colored triangles. The curve at the very bottom is the commits frequency curve. Triangles representing patterns are placed by the leftmost corner the pattern occurence start. The triangle length corresponds to the sliding window size - seven days, the color used for a triangle corresponds to the symbolic pattern. The small vertical lines at left are the right-side artifacts of patterns which do not fit into the current viewport (from 2010-03-01 to 2010-04-01)

4.3 Choice of SAX parameters

The cornerstone question in SAX transformation is the choice of the parameters. By varying the sliding window length I am able to slice the search space to only see patterns within the time-interval which is the length of this window or it's miltiple (I need citation here about the choice of natural to data intervals)...

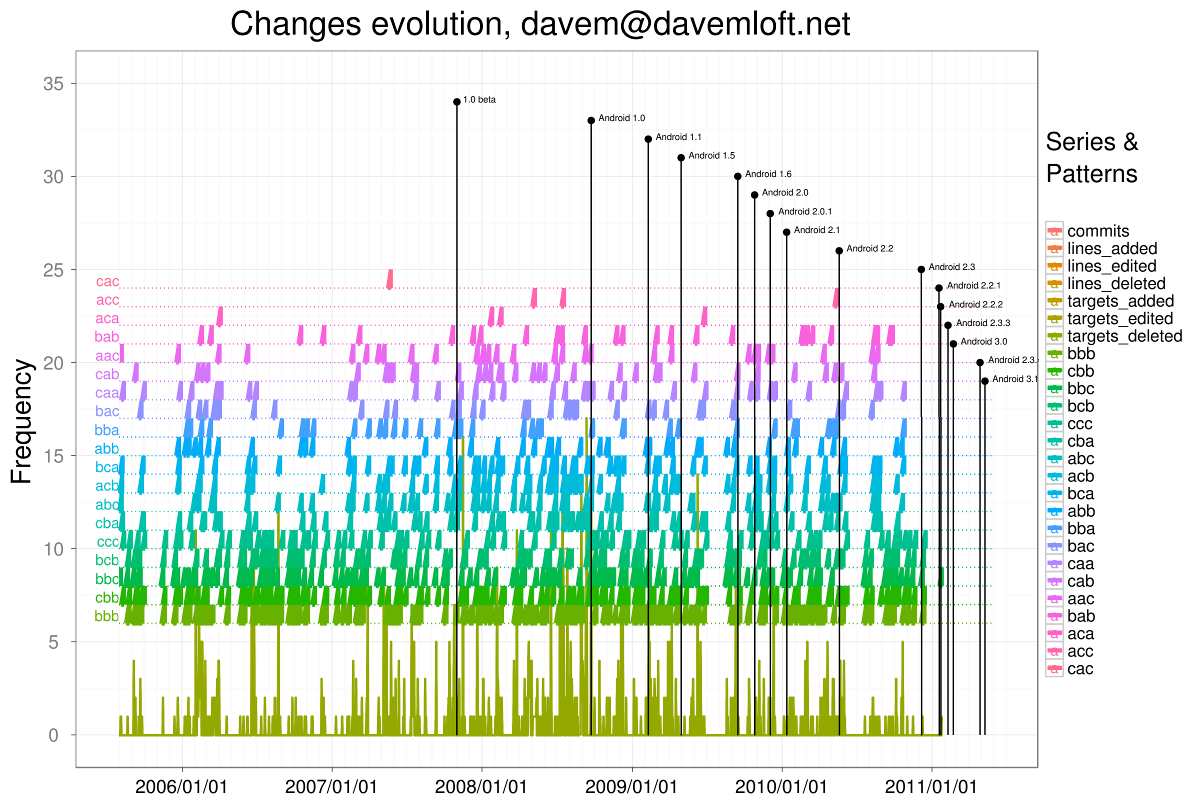


Figure 5: The illustration of the temporal SAX patterns distribution at the larger scale.

For this time I picked weekly, bi-weekly and a monthly intervals. In my opinion they correspond to natural units of time within the software process.

The choice of PAA size and an alphabet has a direct impact on the sensitivity of SAX algorithm and on the size of the resulting dictionaries. By varying the PAA size algorithm is able to smooth the high-frequency noise within the sliding window interval by aggregating values within the PAA piece to their mean value. By proper choice of PAA size it is possible to focus only on the certain-scale frequency within the data stream by analogy to FFT. The PAA size defines the length of the resulting dictionary words. The choice of the alphabet size not only limits the amount of letters within the resulting words, thus having direct influence on the size and complexity of the dictionary, but tunes the sensityvity of the search.

to this length discriminate only this

4.4 Choice of storage strategy

5 Data Mining

Here I discuss the results

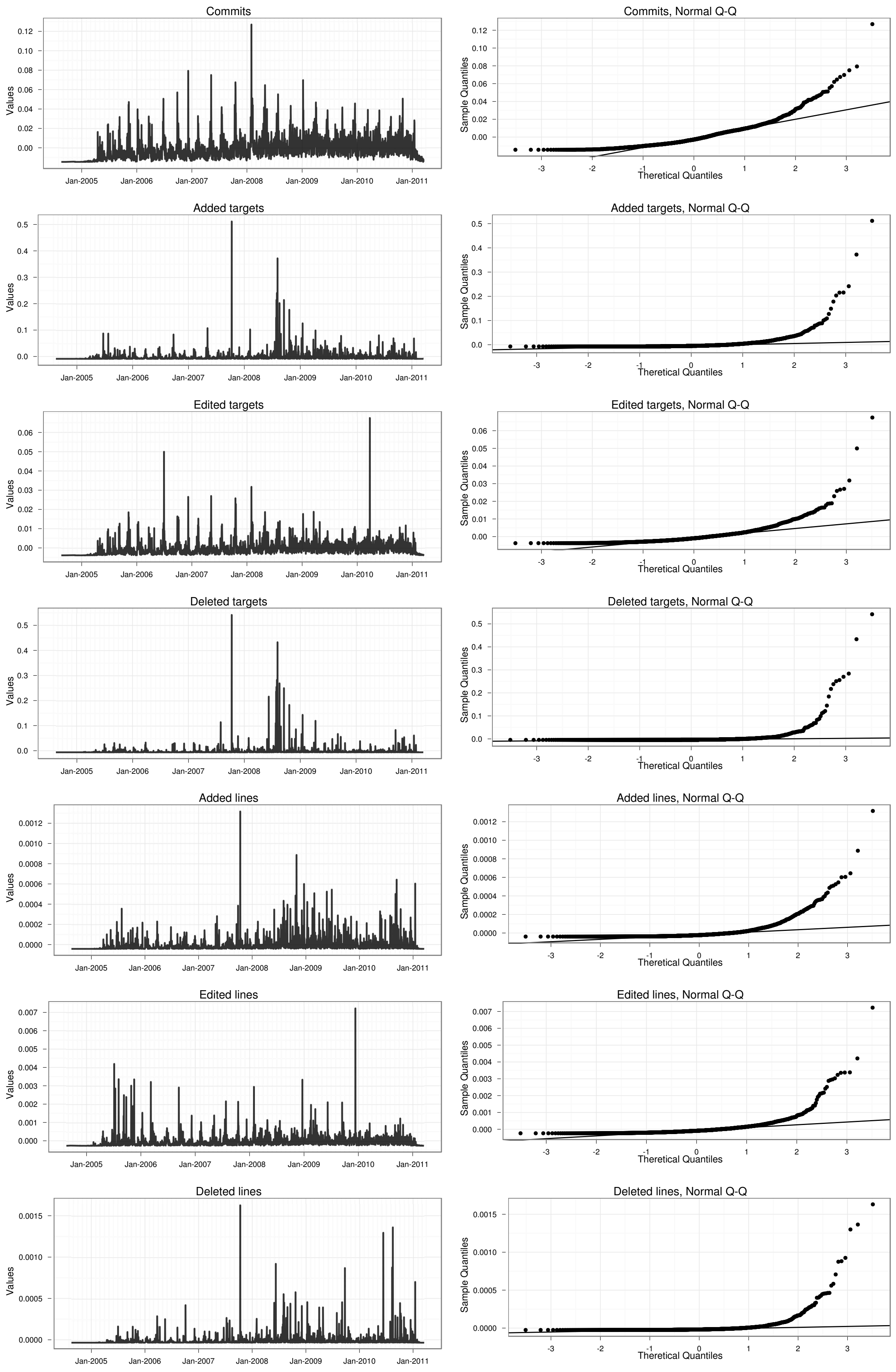


Figure 6: The distribution of time-series values after normalization. OMAP-kernel.

5.1 Patterns classification

5.2 Hierarchical clustering

5.3 K-means clustering

6 Note to myself: wrong counts

Listing 4: Wrong counts here in that second query due to the join. Look at the counts in third query to understand why

```
select date_format(c.author_date, '%Y-%m-%d') as day, count(distinct(c.id)) as commits,
sum(c.added_files) as added_files, sum(c.edited_files) as edited_files,
sum(c.removed_files) as removed_files, sum(c.added_lines) as added_lines,
sum(c.edited_lines) as edited_lines, sum(c.removed_lines) as removed_lines
FROM android_change c
where c.author_id=153 and c.project_id=18
and c.author_date between "2010-03-01" AND "2010-04-01"
 and date_format(c.author_date,'%H') between 17 and 23
group by date_format(c.author_date, '%Y%m%d');

select date_format(c.author_date, '%Y-%m-%d') as day, count(distinct(c.id)) as commits,
count(t.change_id) as t, sum(c.added_files) as added_files, sum(c.edited_files)
as edited_files, sum(c.removed_files) as removed_files, sum(c.added_lines) as added_lines,
sum(c.edited_lines) as edited_lines, sum(c.removed_lines) as removed_lines
FROM android_change c
left join change_target t on c.id=t.change_id
where c.author_id=153 and c.project_id=18
and c.author_date between "2010-03-01" AND "2010-04-01"
 and date_format(c.author_date,'%H') between 17 and 23
group by date_format(c.author_date, '%Y%m%d');

select c.*, t.*
FROM android_change c
left join change_target t on c.id=t.change_id
where c.author_id=153 and c.project_id=18
and c.author_date between "2010-03-01" AND "2010-04-01"
 and date_format(c.author_date,'%H') between 17 and 23;
```

References

[1] 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*, volume 30 of *MSR '05*, pages 1–5, New York, NY, USA, 2005. ACM.

[2] 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, 2007. IEEE Computer Society.

[3] 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*, pages 295–298. IEEE, May 2009.

[4] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, Oct. 2007.