Sanity check and the Trajectory workflow test

Pavel Senin

February 22, 2012

Abstract

This document has dual purposes. First of all, by writing it I am following the common Trajectroy analysis workflow. The second purpose I am fulfilling is to compile a primer of how the data is getting transformed within the Trajectory and how one may itrepret results.

1 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 information concerning contributors and projects.

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;
```

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

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.

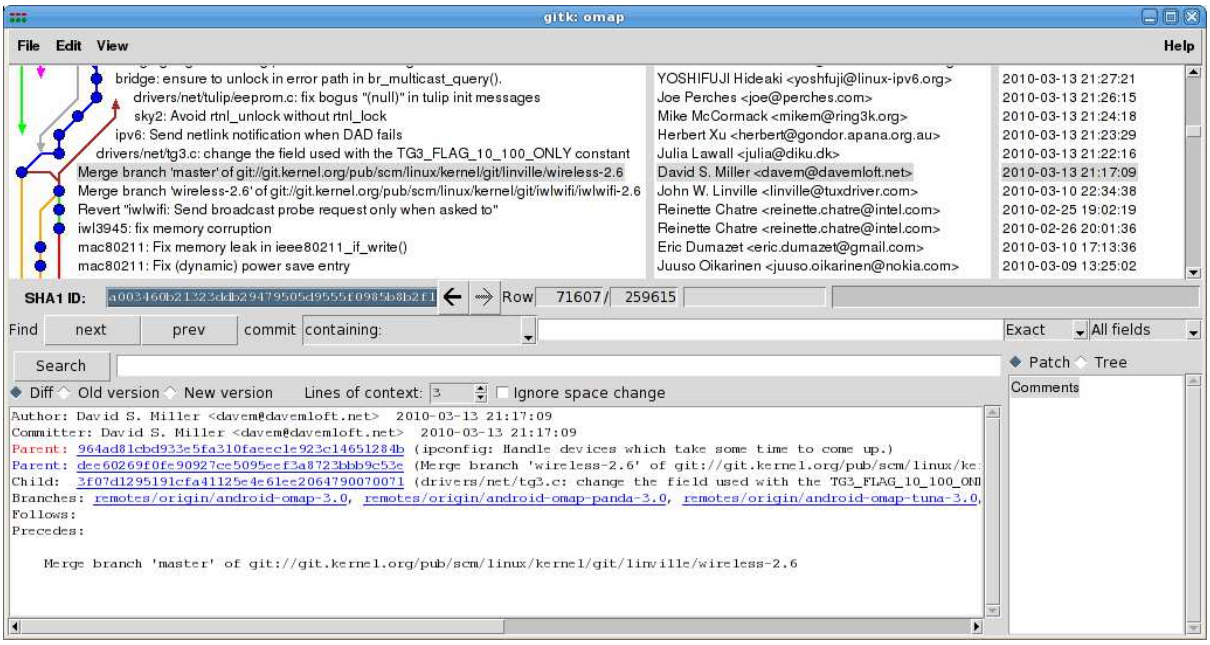

Figure 1: The GUI git viewer screenshoot illustrating commit statistics.

It is nice to see that TrajectoryDB has all the data immediately available in Git, however the result in the Table 1 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.

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;
```

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

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.

2 Indexing aggregated commits statistics with SAX

Defining SAX alphabets, Lin and Keogh [1] it is assumed that the series are

operates transform I normalize time series to have mean of zero and a standard deviation of one before converting it to the SAX representation , since it is well understood that it is meaningless to compare time series with different offsets and amplitudes

The data from the Table 3 is getting then 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.

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.

Figure 2: 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

The Figure 2 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).

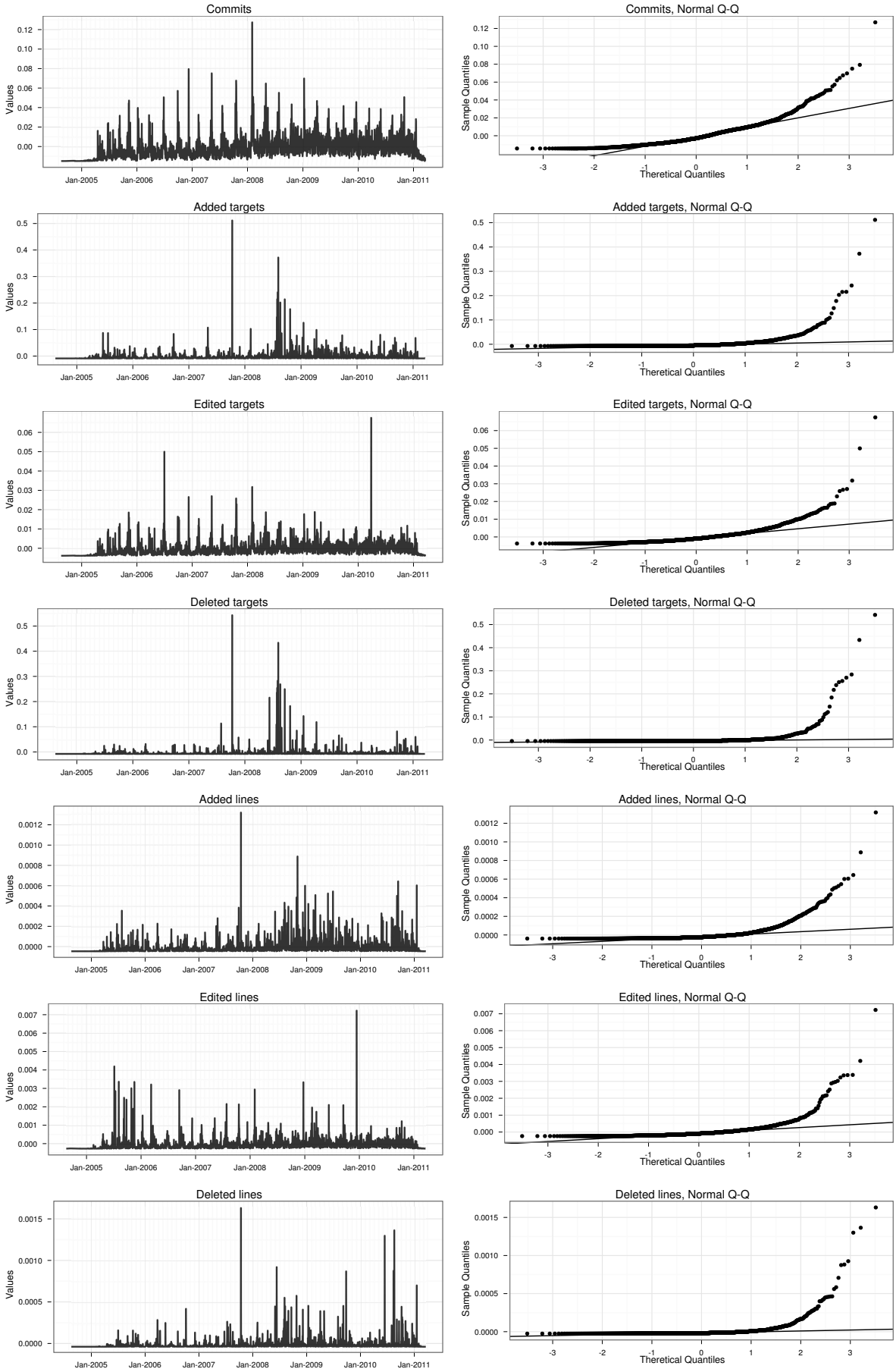

Figure 3: The Distribution of values after normalization.

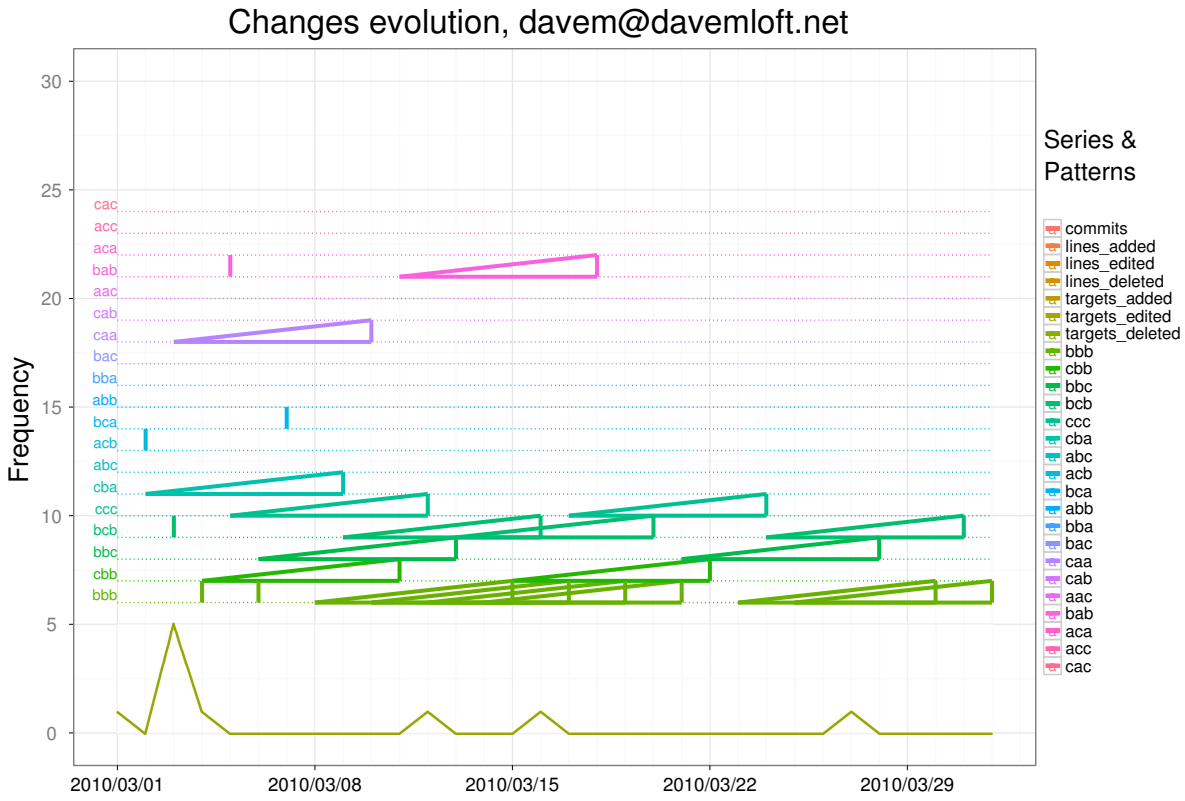

Figure 4: The illustration of the temporal distribution of patterns from Figure 2. Here I superimpose patterns over the commit frequency plot by using colored triangles. The curve at the very bottom is the commits frequency curve. Triangles representing patterns are placed by the leftmost corner the 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)

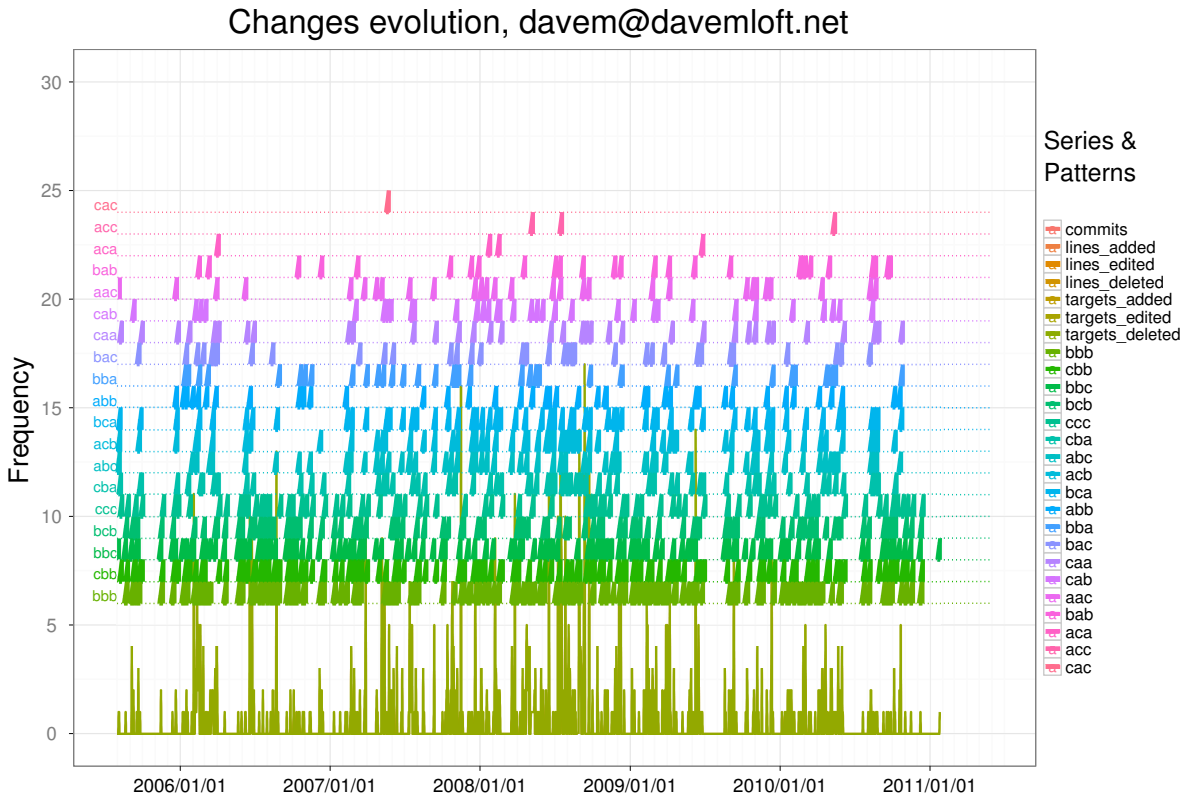

Figure 5: The illustration of the temporal SAX patterns distribution at the larger scale.

3 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] 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.