# Software Trajectory Analysis

an empirically-based method for software process discovery


**Pavel Senin**

University of Hawaii at Manoa

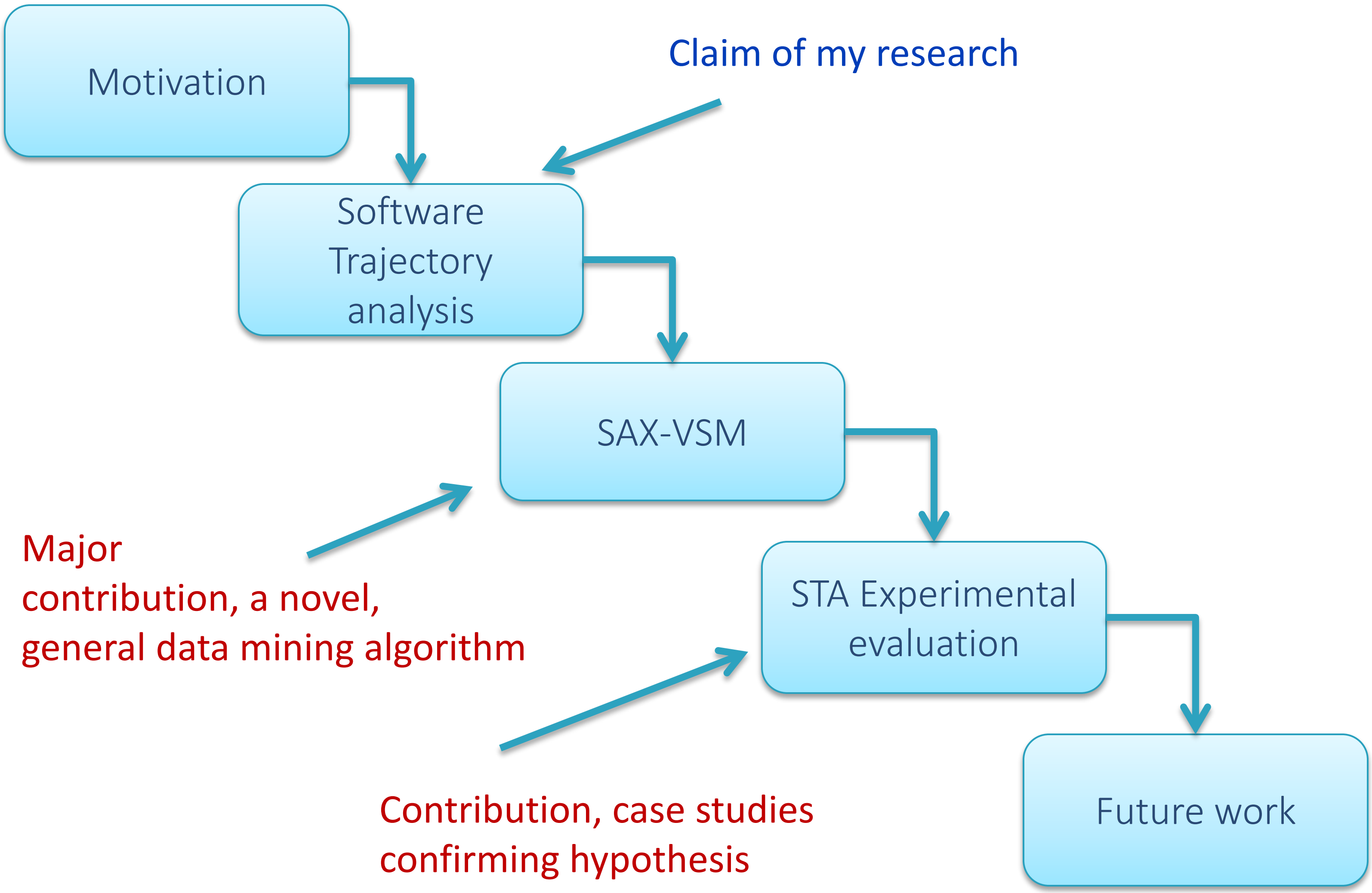
Department of Information and Computer Sciences

Collaborative Software Development Laboratory

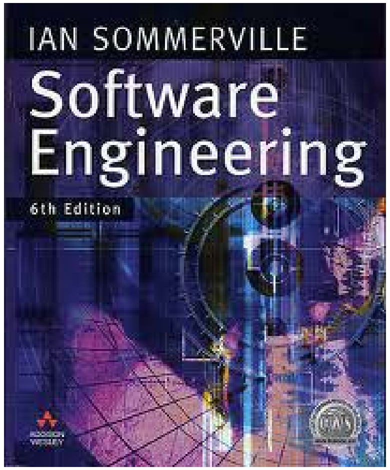
http://csdl.ics.hawaii.edu

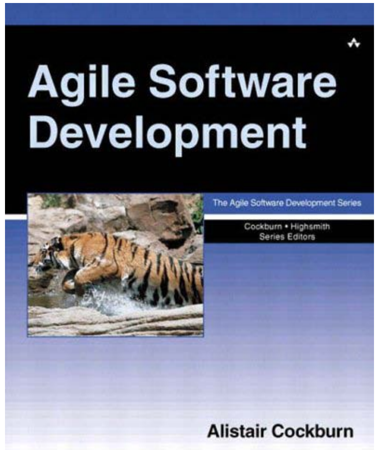
senin@hawaii.edu

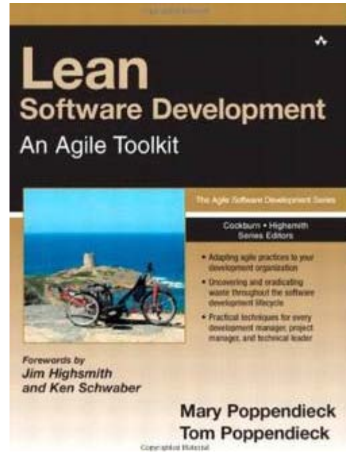
# Presentation outline, waterfall-style

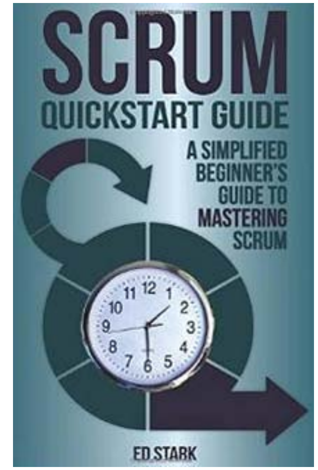


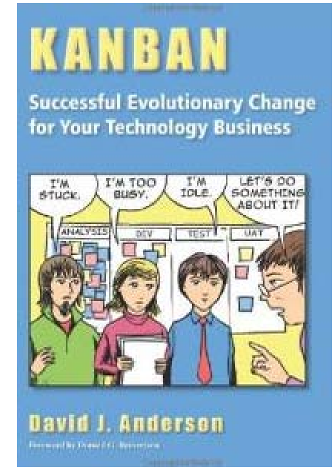
# We have a lot of software processes

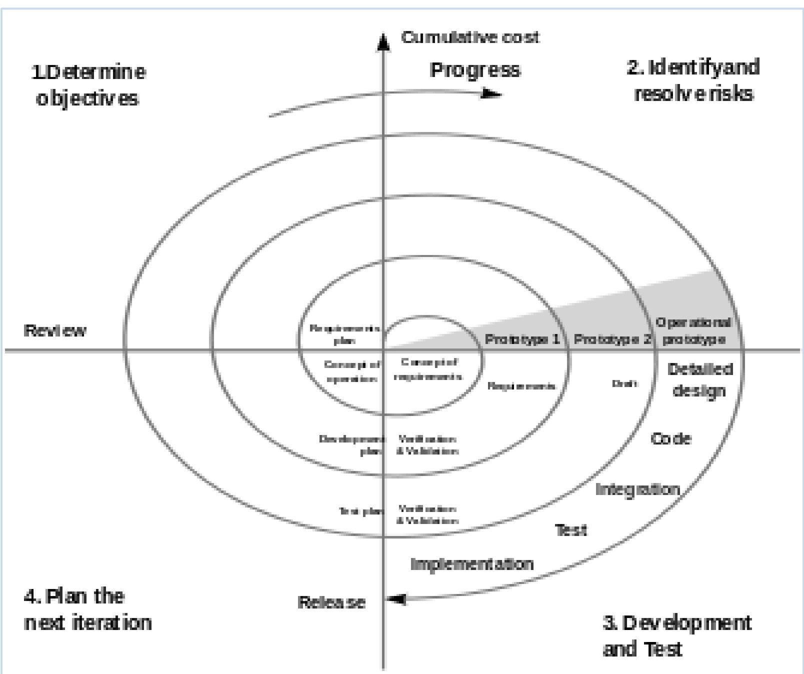


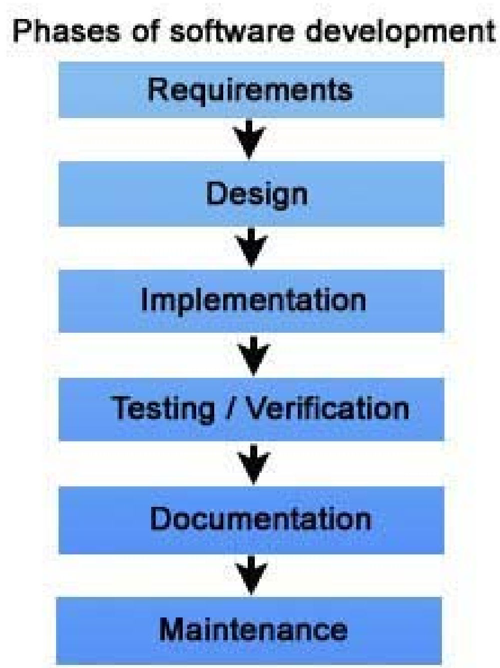



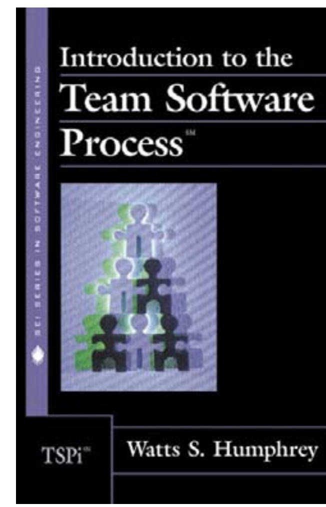



# Where do they come from?

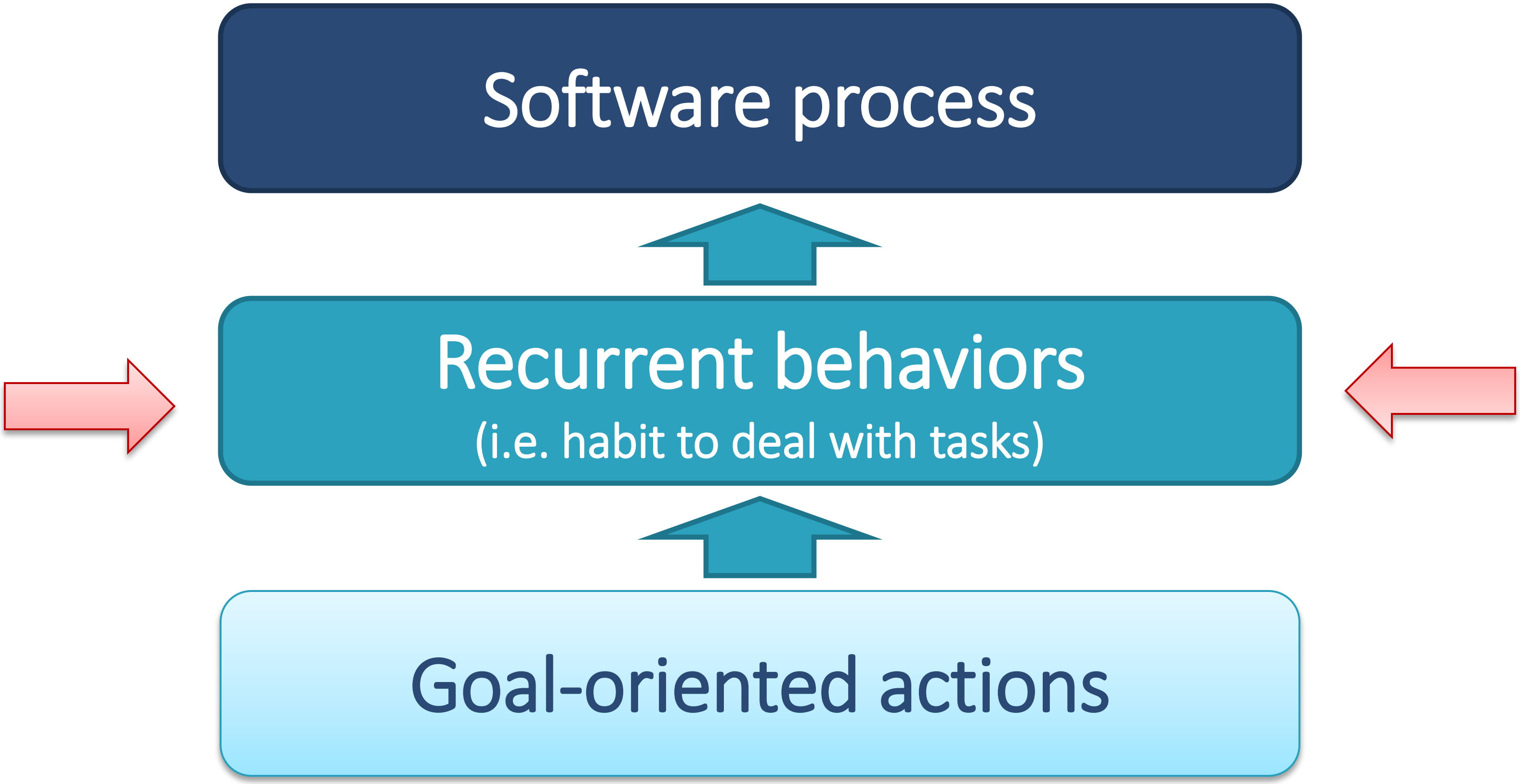
Top-down approach

- Some people (or groups) invent them
  - Waterfall: W. Royce, 1970
  - Spiral development, B. Boehm, 1988
  - Rapid Application Development, J. Martin, 1991
  - Agile, Agile Manifesto, 2001
  - TDD: Kent Beck, 2003

Bottom-up approach

- others built systems to discover them
  - Cook & Wolf, DAGAMMA, 1998. (Finite State Machines)
  - Van der Aalst et al., 2007. (Petri Nets, Transition Systems)
  - Jensen & Scacchi, 2007. (Reference Model)

# Software Trajectory Analysis (STA)
# a new bottom-up approach for software process design




The bottom-up approach is the iterative piecing together atomic elements to give rise to grander systems. Elementary actions linked together into recurrent behaviors, which, potentially can be linked into the larger entity – a software process.

# Claim of my research

***Initial***: It is possible to discover recurrent behaviors by systematic analysis of low-level development actions

***Revised***: It is possible to discover recurrent behaviors by systematic analysis of software process artifacts

# Contributions of my research

1. **Software Trajectory Analysis framework**
   - Peer-reviewed publication in ESEM-2010 doctoral symposium
2. **SAX-VSM – an algorithm for class-characteristic behaviors discovery**
   - Peer-reviewed publication in ICDM-2013
3. **SAX-VSM implementation and experimental evaluation, including the performance evaluation in hierarchical and k-means clustering**
   - Peer-reviewed publication in ICDM-2013, partially unpublished
4. **SAX-VSM discretization parameters optimization scheme**
   - Peer-reviewed publication in ICDM-2013, partially unpublished
5. **Novel time-series class-characteristic heatmap-like visualization**
   - Peer-reviewed publication in ICDM-2013
6. **JMotif, an open-source Java library for the time series classification, recurrent and rare patterns discovery, and the time series grammatical decomposition**
   - Peer-reviewed publications in ECML/PKDD 2014, EDBT 2015
7. **Software Trajectory Analysis empirical evaluation**
   - Publication in SERP-2012
   - Future publication summarizing the dissertation

# Discovering recurrent behaviors

Online observations

Expensive,  Intrusive, Misleading

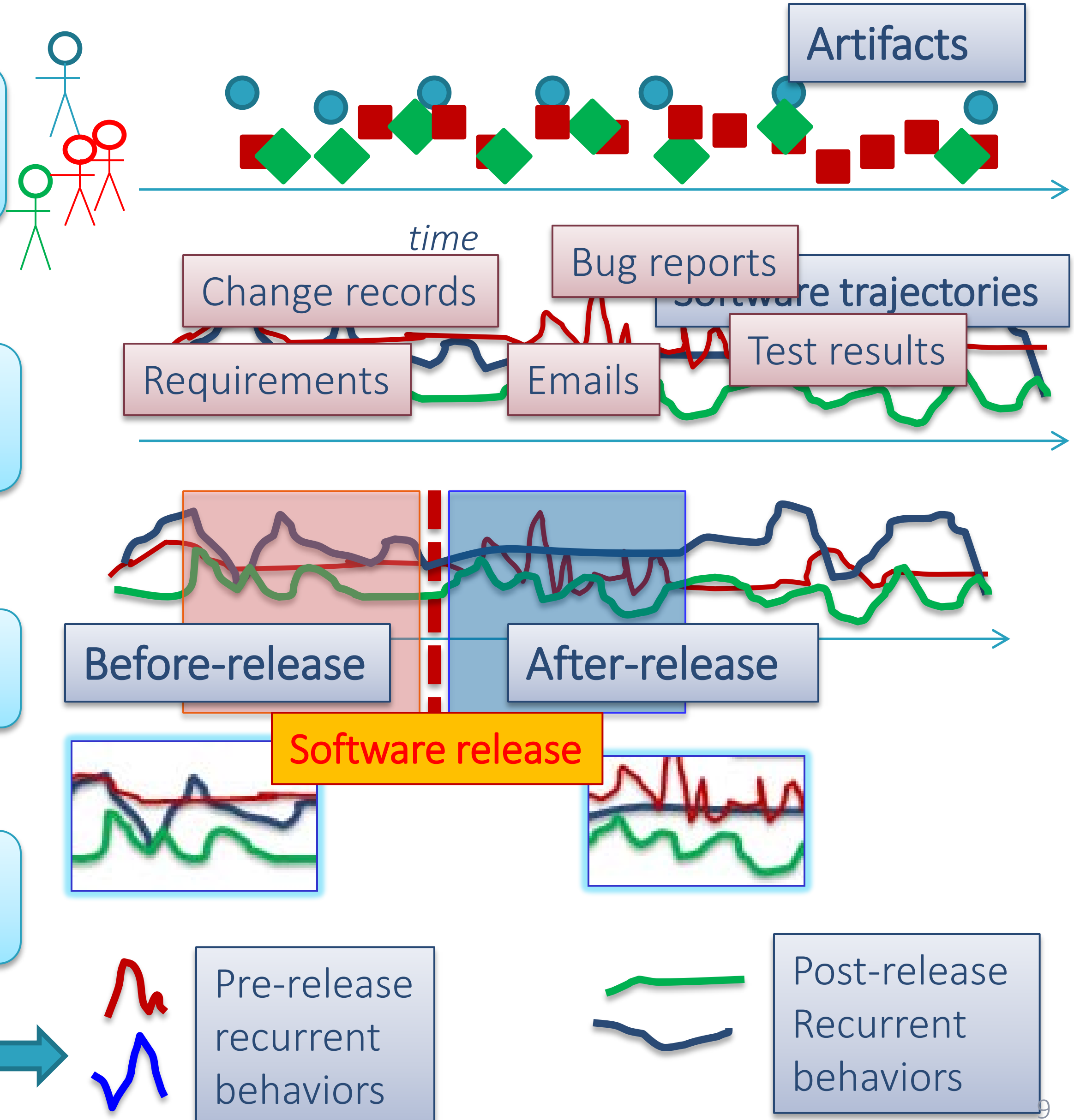
In- or post- process interviewing

Expensive, Intrusive, Misleading, Annoying

We want an inexpensive, non-intrusive tool which facilitates the <u>systematic</u> recurrent behaviors discovery

# Contribution #1: Software Trajectory Analysis framework.
# (shown in a nutshell)

1. Software artifacts collection
2. Software artifacts measurements
3. Software trajectory definition
4. Characteristic patterns discovery and ranking

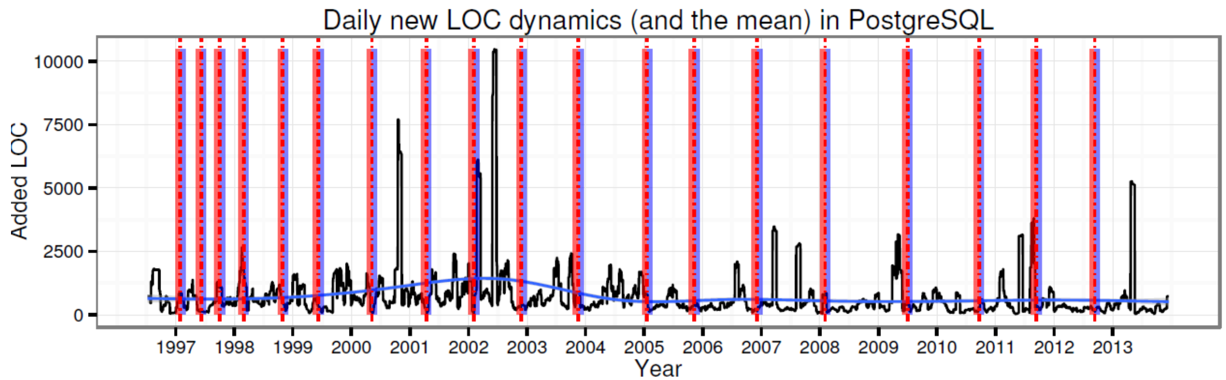


# Software Trajectory analysis application example

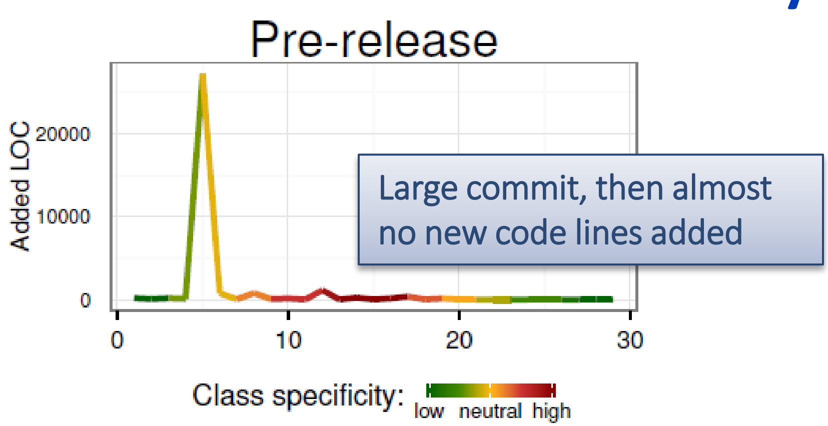
## 1. Software artifact – software change record

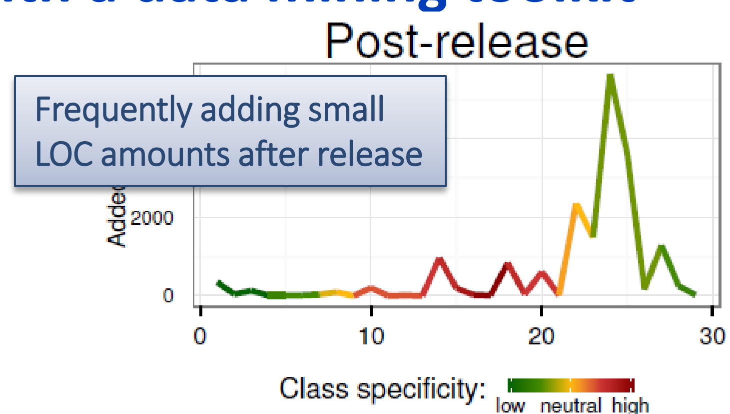
| | | |
|---|---|---|
| Fix previous patch to exprTypmod. | Tatsuo Ishii<ishii@postgresql.org> | 25 May 2005 04:13:48 |
| Inserting 5 characters into char(10) does not produce 5 padding spaces if t... | Tatsuo Ishii<ishii@postgresql.org> | 25 May 2005 01:15:05 |
| Previous fix for "x FULL JOIN y ON true" failed to handle the case where ther... | Tom Lane<tgl@sss.pgh.pa.us> | 24 May 2005 20:03:24 |
| Guard against duplicate IDs in input file in SortTocFromFile(). Per report fro... | Tom Lane<tgl@sss.pgh.pa.us> | 17 May 2005 19:30:53 |
| REL7_4_8 Update release notes for upcoming re-releases. | Tom Lane<tgl@sss.pgh.pa.us> | 9 May 2005 02:10:22 |

## 2 & 3. Systematic measurements => software trajectory synthesis

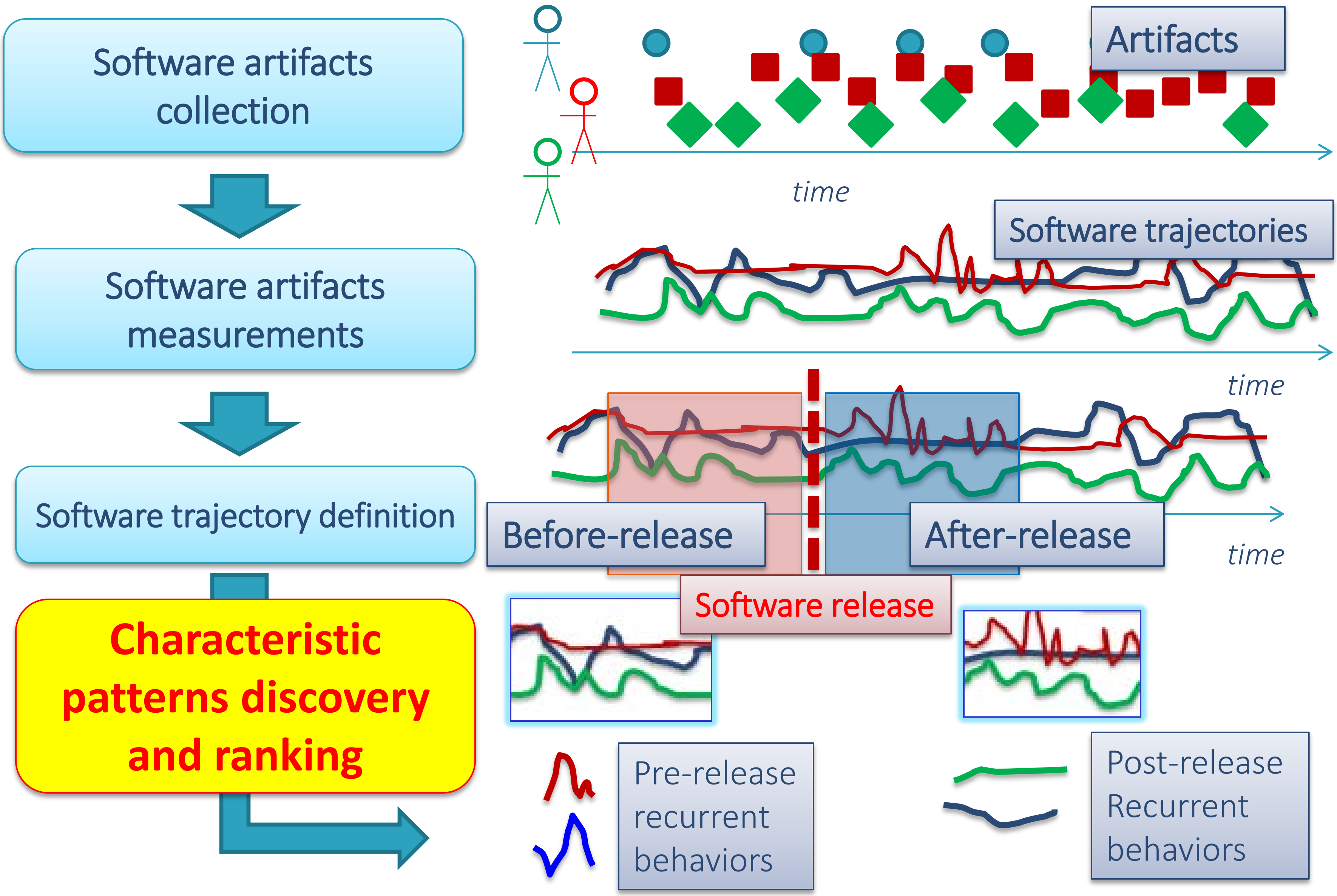



## 4. Recurrent behaviors discovery with a data mining toolkit



# What is really new?



# Software Trajectory Analysis framework: characteristic behavior discovery and ranking magic

- **Q**: How to discover class-characteristic recurrent behaviors in software trajectories and rank them by characteristic power?

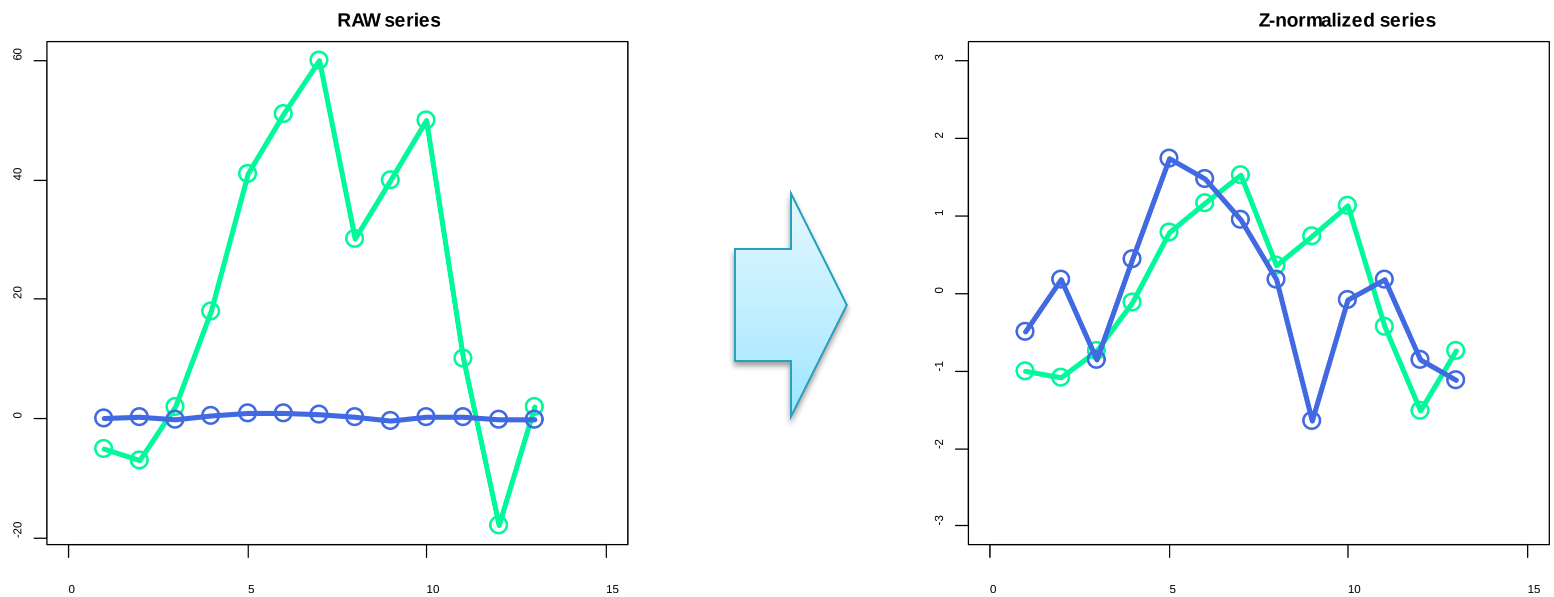
- **A**: I have invented an algorithm called **SAX-VSM** that enables class-characteristic behaviors discovery and ranking.

- *The algorithm embeds a number of other algorithms (solutions), which we are going to see next at the high level, as the preliminaries.*

# SAX-VSM step: z-normalization (to unit of energy)

$$x_i \in X \quad x_i^{'} = \frac{x_i - \mu}{\sigma},\ i \in \mathbb{N} \quad \text{yields the vector}$$

$$X_i^{'} \text{ such as } \mu_{X'} \approx 0 \text{ and } \sigma_{X'} \approx 1$$

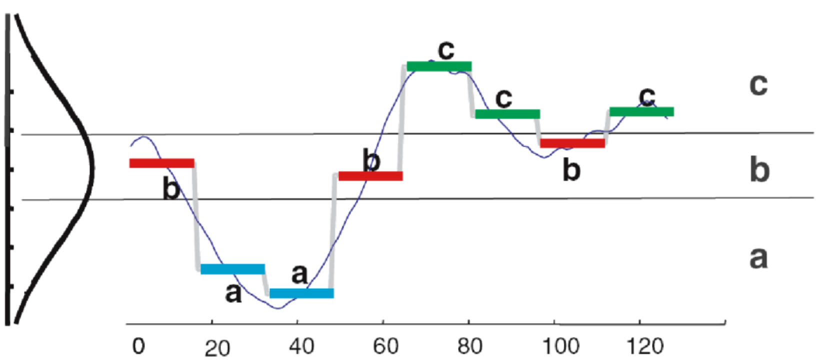



Allows to focus on temporal features, not scale

# SAX-VSM step: Symbolic Aggregate approXimation

1. Z-normalization
2. PAA – Piecewise **Aggregate** Approximation
3. SAX – Symbolic **Aggregate** approXimation

## Conversion of time series of size 128 into 8 symbols

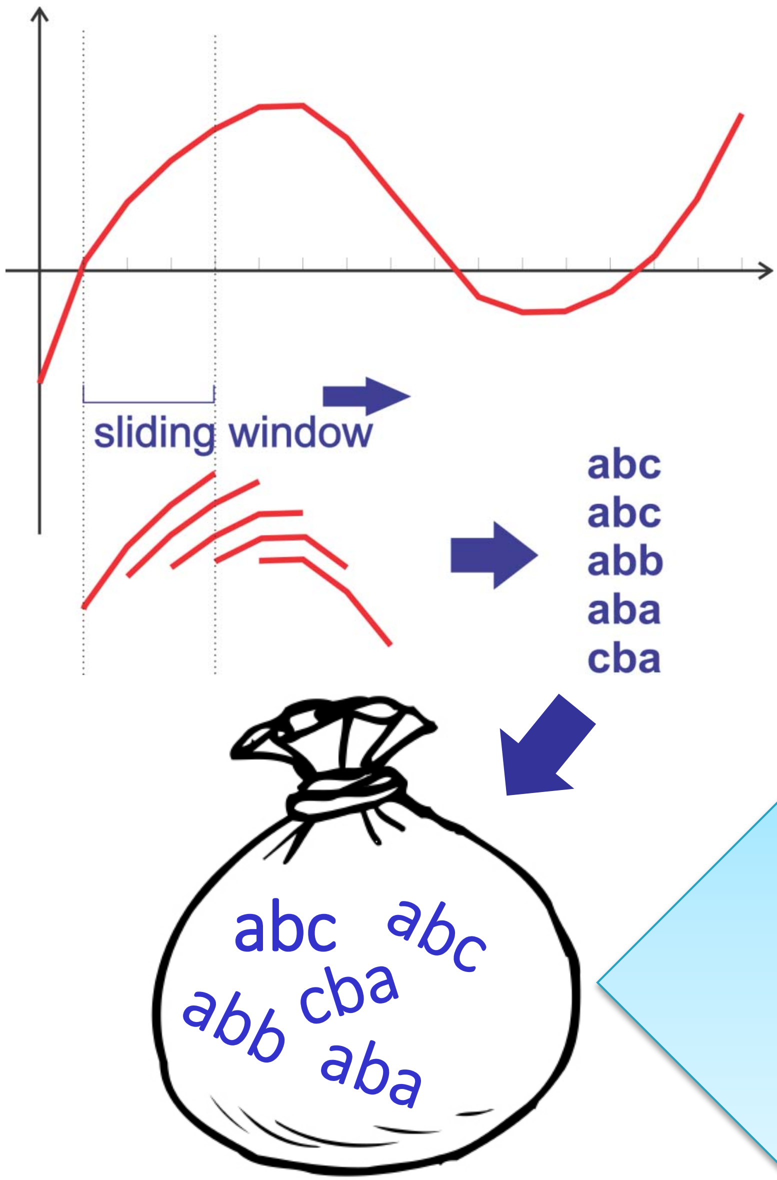



baabccbc

In PAA we aggregate many (16) points into single value – the mean value of these points

In SAX we convert these means into symbols based on the given alphabet

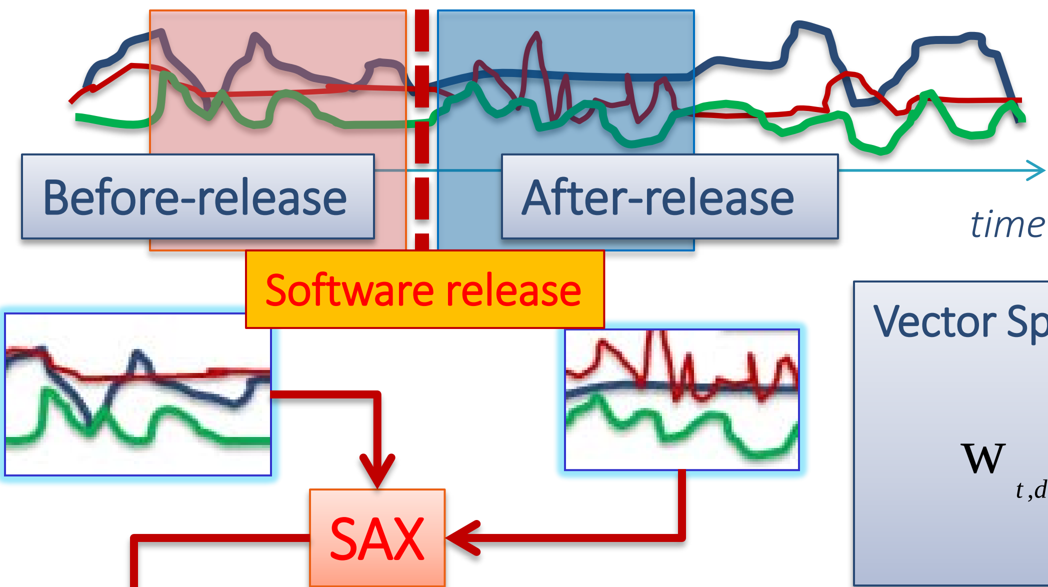
# SAX-VSM step: sliding window SAX discretization




1. Three parameters are required for this:
   1. sliding window size,
   2. number of PAA segments,
   3. the alphabet size.
2. By sliding a window we extract subsequences, each of them:
   1. z-normalized,
   2. discretized,
   3. placed into a "bag of words" that corresponds to a class.

- Note that at this point we do not care about the SAX words ordering anymore.

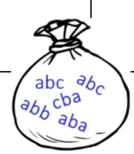
# SAX-VSM: VSM, words (behaviors) rating.
# TF*IDF statistics

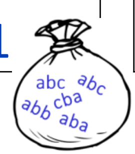



Vector Space Model . Salton, Wong, Yang, '71 '75

$$w_{t,d} = \underbrace{\log(1+\mathrm{tf}_{t,d})}_{\text{TF}} \times \underbrace{\log_{10}(N/\mathrm{df}_t)}_{\text{IDF}}$$

| Word | Counts: Pre-release | Counts: Post-release | Pre-release: TF | Pre-release: IDF | Post-release: TF | Post-release: IDF | ***Word weights***: Pre-release | ***Word weights***: Post-release |
|---|---|---|---|---|---|---|---|---|
| abc | 45 | 0 | 1.66 | 0.30 | 0.00 | 0.00 | **0.50** | 0.00 |
| cba | 32 | 23 | 1.52 | 0.00 | 1.38 | 0.00 | **0.00** | **0.00** |
| caa | 0 | 11 | 0.00 | 0.00 | 1.08 | 0.30 | 0.00 | **0.32** |



# SAX-VSM: How SAX gets the three parameters?

through the cross-validation procedure,
where the cost function is based on the Cosine similarity

It has been shown, that the Cosine similarity works much better for vectors of TF*IDF weights than any other distance metrics.

(Salton & Buckley, 1988) .

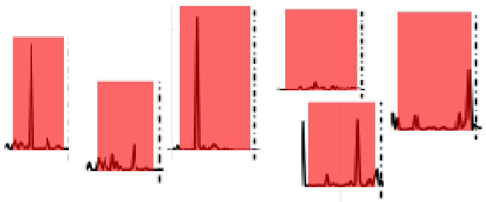
$$\cos(\vec{q},\vec{d}) = \underbrace{\frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|}}_{\text{Dot product}} = \underbrace{\frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|}}_{\text{Unit vectors}} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

$q_i$ is the tf-idf weight of word $i$ in the query
$d_i$ is the tf-idf weight of word $i$ in the document
cos($q$,$d$) is the cosine similarity of $q$ and $d$ … or,
equivalently, **the cosine of the angle between $q$ and $d$.**

larger cosine value => smaller the angle => weight vectors are more similar

# Contribution #2:
# SAX-VSM algorithm for trajectory-characteristic behavior discovery

Pre-release

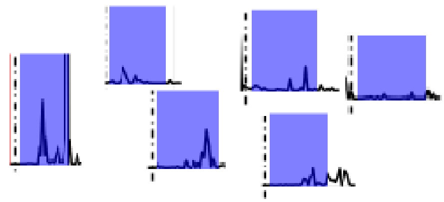


SAX

bag of words

accbb, cdaaa, cdaaa, ...

Post-release

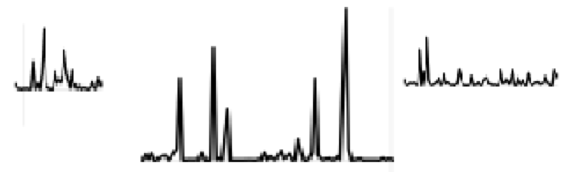


SAX

bag of words

acabb, cbcdc,ddbca,...

Unlabeled sfw. trajectory



SAX

bag of words

ccbaa, ccbba, cbbba...

TF*IDF

words

TF*IDF vectors

| | Pre | Post |
|---|---|---|
| accbb | 0.023 | 0.000 |
| cdaaa | 0.140 | 0.000 |
| … | … | … |
| ddbca | 0.000 | 0.010 |

weighted words

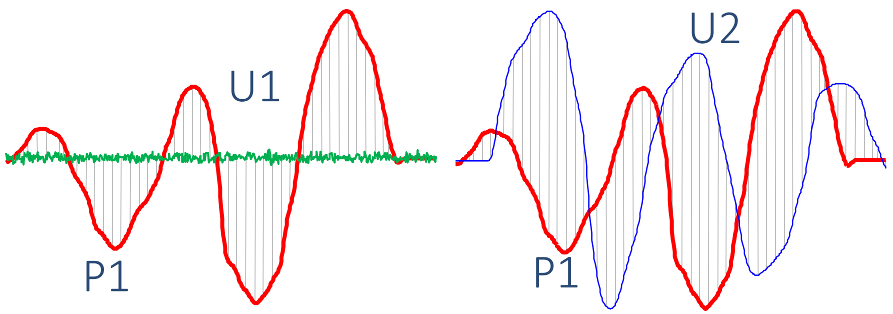
Class label =

$$\arg\max_i \{tfidf_i * freq_{unlabeled}\}$$

where $i \in (1,2)$

Cosine-similarity based classification

# Contribution #3: SAX-VSM performance evaluation

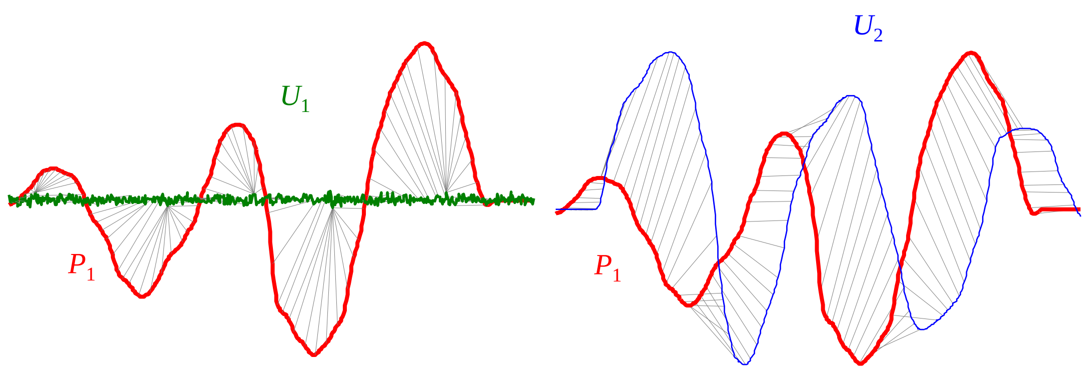



## Euclidean distance

"… it is clear that one-nearest-neighbor with Dynamic Time Warping (DTW) distance is exceptionally difficult to beat…"

*"Fast Time Series Classification Using Numerosity Reduction", Xi, Keogh, Shelton, Wei, 2006*

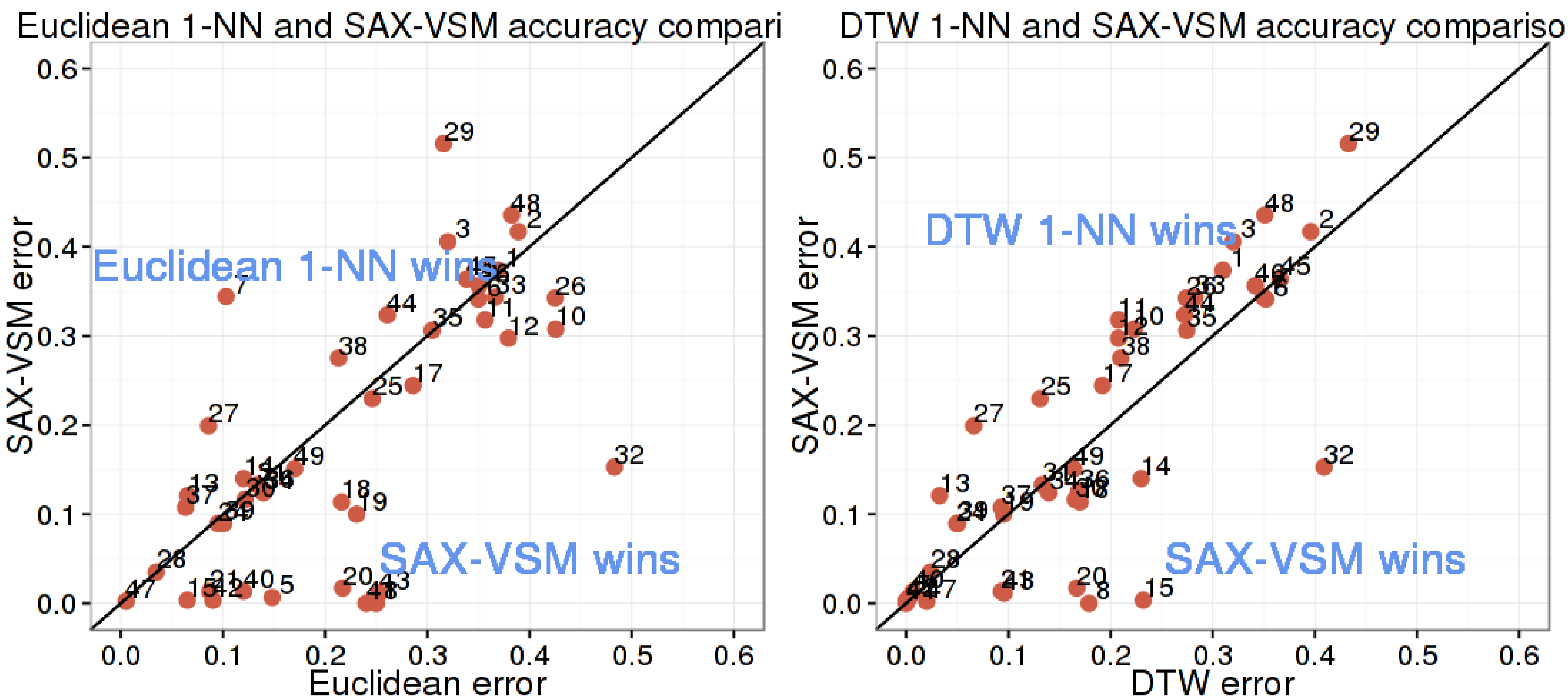
## DTW




Experimental Design:

Competing against 1NN classifiers built upon Euclidean distance and DTW

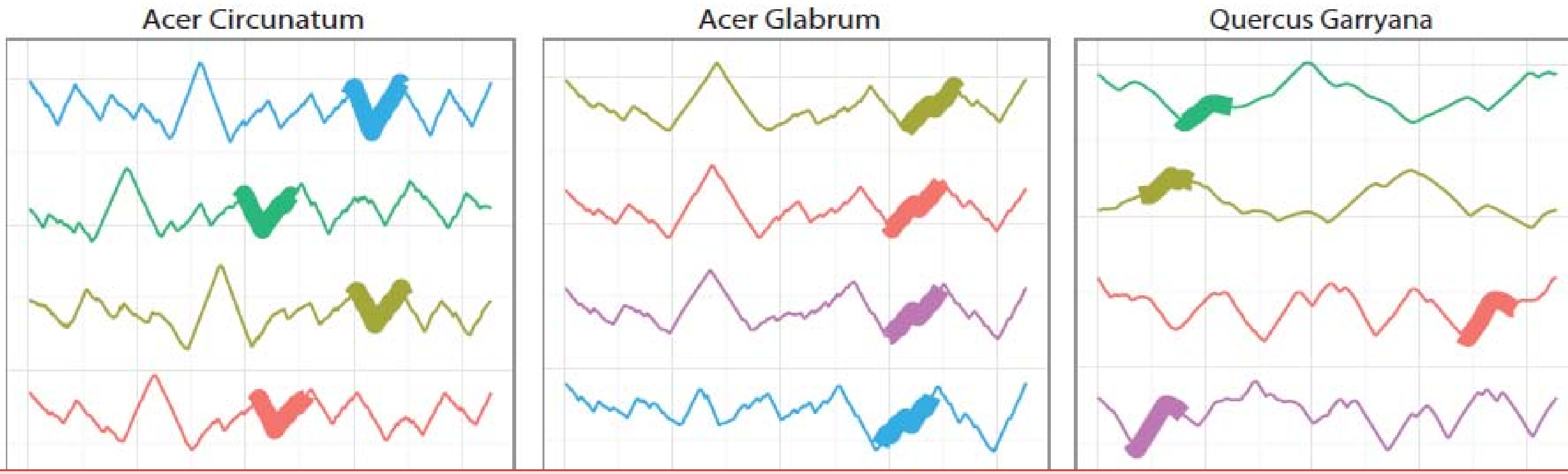
# SAX-VSM classification accuracy evaluation on the standard UCR dataset (48 sets)




Sum of errors: Euclidean 1NN 12.57, SAX-VSM 11.63, DTW 1NN 10.94

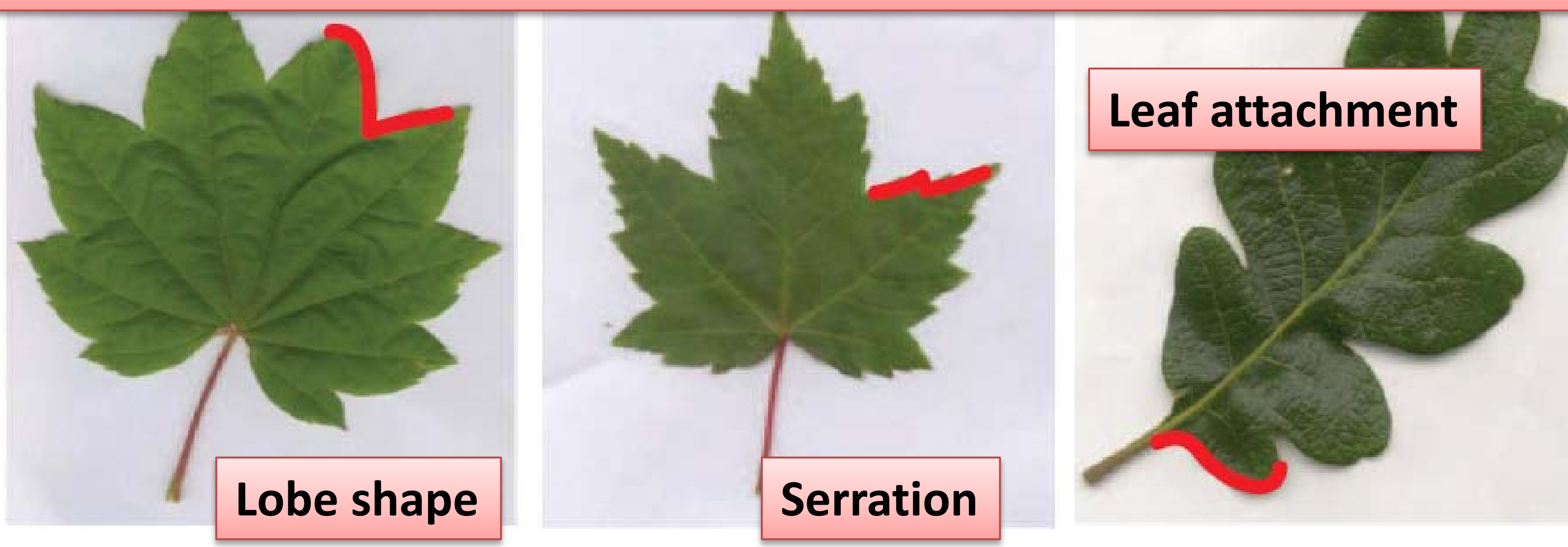
**On some datasets SAX-VSM accuracy is really good, on others it is average, exactly as the "No Free Lunch theorem" points out**

# Example of interpretable SAX-VSM patterns



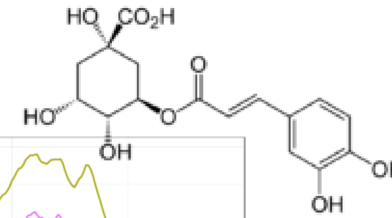

**Accuracy: Euclidean 51.7%, DTW 59.1%, SAX-VSM 92.2%**

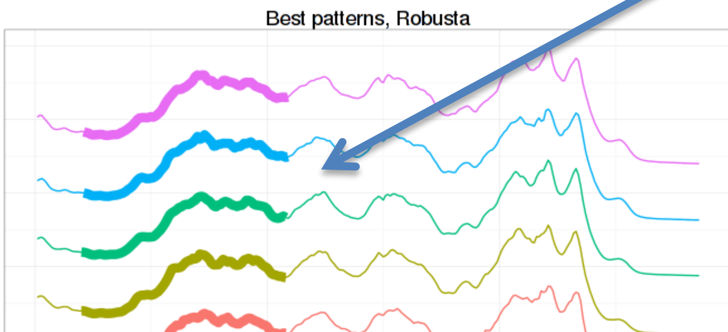
Moreover: SAX-VSM highlights same patterns which the field experts use

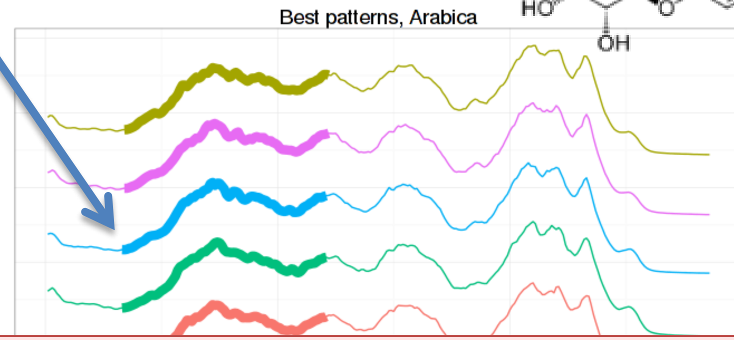


# Example of interpretable SAX-VSM patterns

Chlorogenic acid

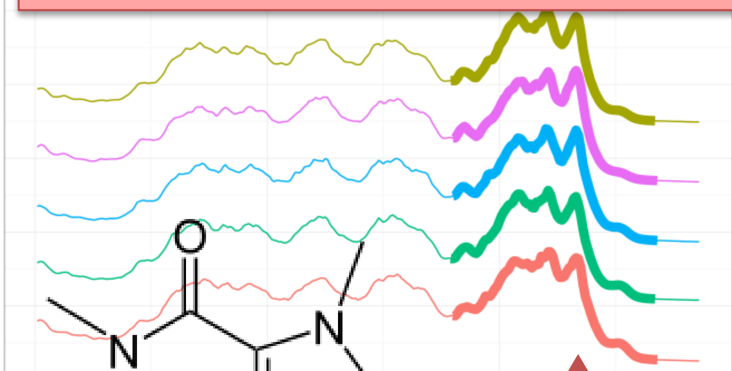


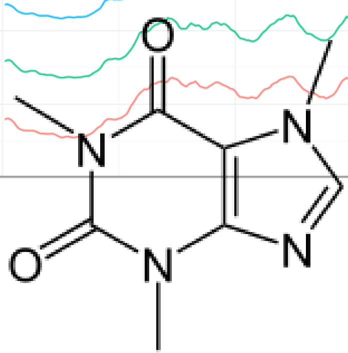



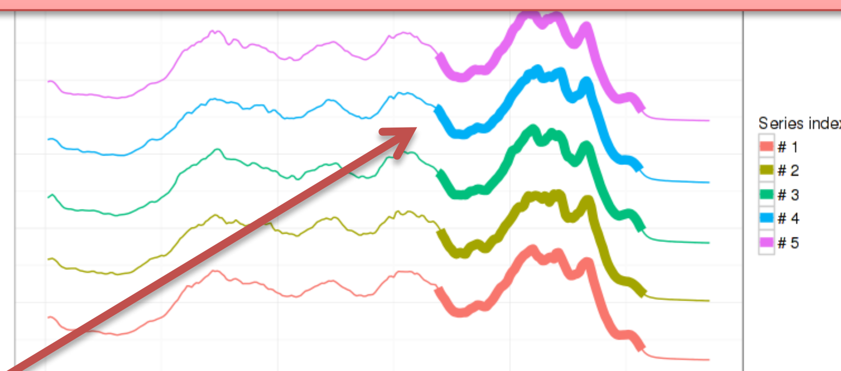




**Accuracy: Euclidean 75.0%, DTW 82.1%, SAX-VSM 100%**

Naturally, SAX-VSM highlights spectra of compounds that are responsible for the taste




Caffeine

| | Caffeine | Chlorogenic Acid |
|---|---|---|
| Arabica | 1.2% | 5.5-8.0% |
| Robusta | 2.2% | 7.0-10.0% |

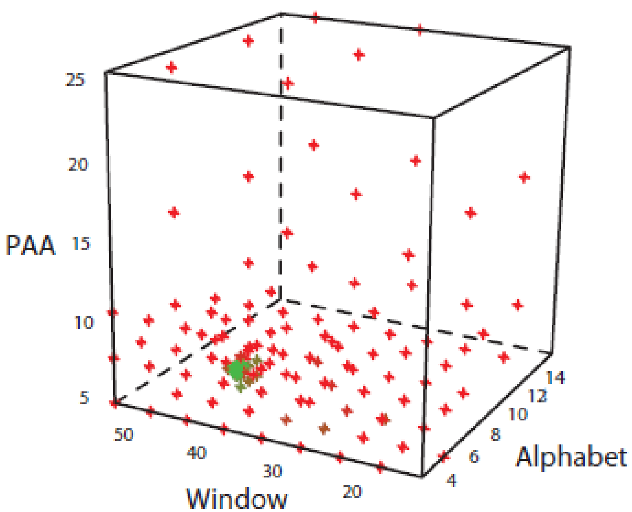
*http://code.google.com/p/jmotif/wiki/ArabicaRobusta*

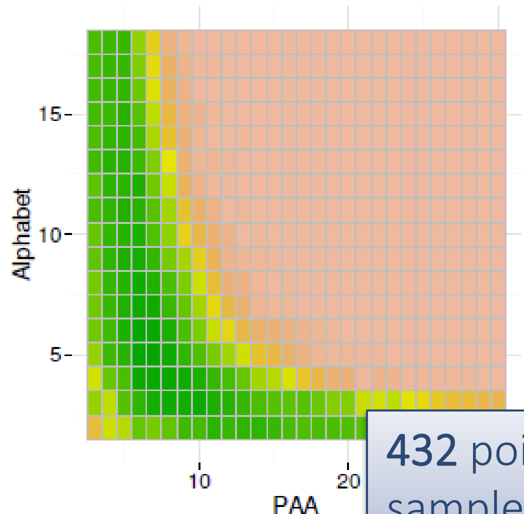
*http://www.coffeechemistry.com/index.php/General/Agriculture/differences-between-arabica-and-robusta-coffee.html*

# But, how do I choose discretization parameters? (Contribution #4)

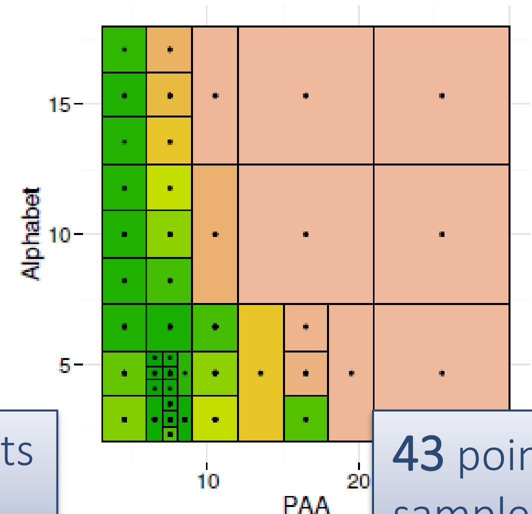
- For the first time, for SAX-based technique, I have implemented a discretization parameters optimization scheme.
- It is based on previously developed general parameters optimization scheme called DIviding RECTangles.
- SAX-VSM uses the cross-validation procedure to assess the classification error, and the parameters of this cost function are optimized by DIRECT.

(1) Sliding window size; (2) PAA number; (3) The alphabet size



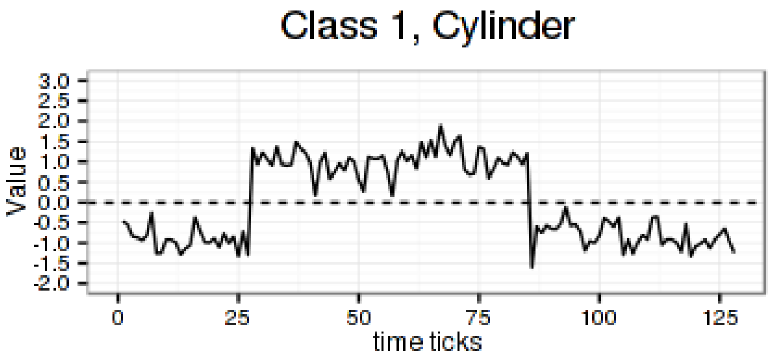

432 points sampled

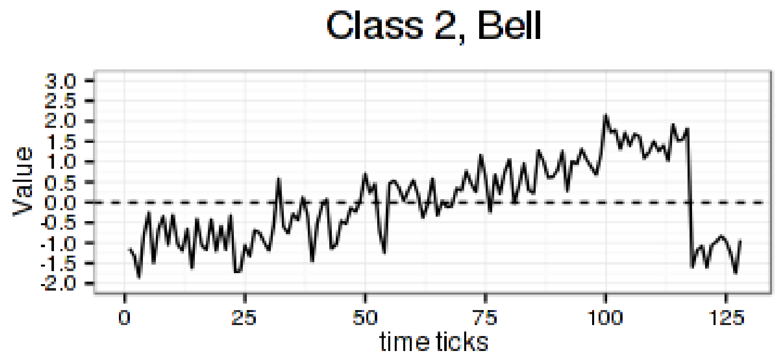


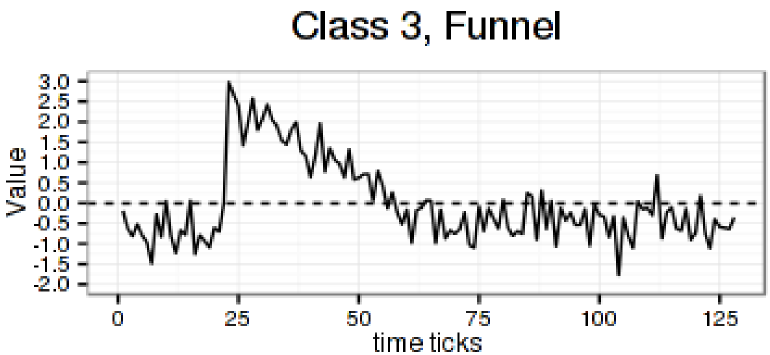

43 points sampled

DIRECT converges orders of magnitude times faster than the grid scan

# SAX-VSM example: Cylinder-Bell-Funnel dataset




$$c(t) = (6+\eta) * X_{[a,b]}(t) + \varepsilon(t)$$

$$b(t) = (6+\eta) * X_{[a,b]}(t) * (t-a)/(b-a) + \varepsilon(t)$$

$$f(t) = (6+\eta) * X_{[a,b]}(t) * (b-t)/(b-a) + \varepsilon(t)$$

$$X_{[a,b]} = \begin{cases} 0, t < a \\ 1, a \le t \le b \\ 0, t > b \end{cases}$$

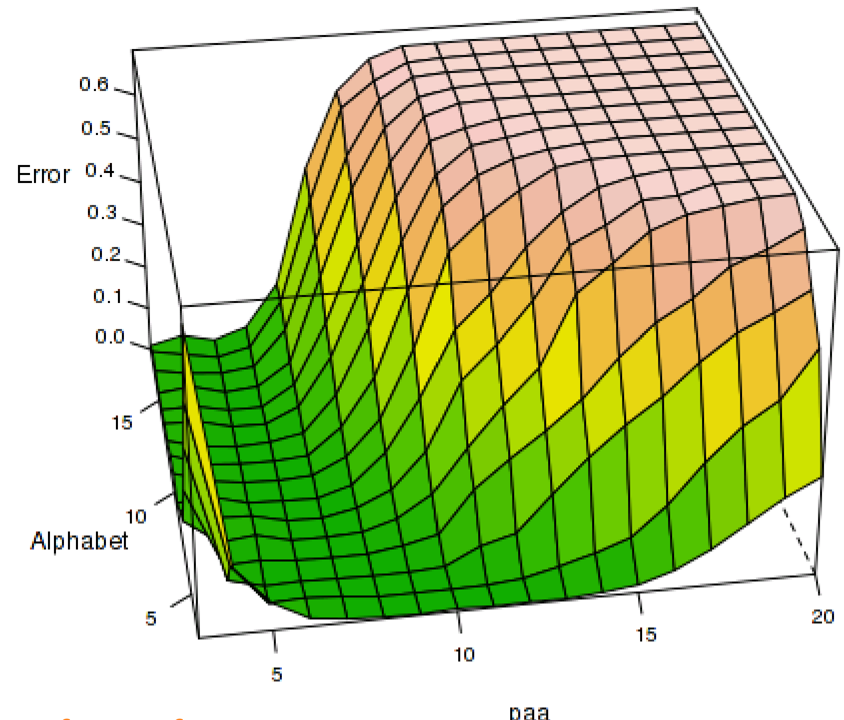
where $\eta$ and $\varepsilon(t)$ are drawn from $N(0,1)$
and $a$ is integer uniformly drawn from $[16,32]$
and $b-a$ is uniformly drawn from $[32,96]$

**Cylinder**: a plateau;
**Bell**: increasing linear ramp, sharp drop;
**Funnel**: a sharp rise, gradual decrease.
The class-characteristic feature start, its duration and the amplitude are randomized. The Gaussian noise is also added

# CBF dataset, SAX-VSM accuracy is 99.99% in 4 parameters optimization iterations

| Features/Learner | Published error |
|---|---|
| Euclidean Distance (Keogh & Kasetty, 2002) | 0.003 |
| TClass/Naive Bayes (Kadous, 2002) | 0.0367 |
| Segments/Naive Bayes (Kadous, 2002) | 0.0620 |
| TClass/C4.5 (Kadous, 2002) | 0.019 |
| Segment/C4.5 (Kadous, 2002) | 0.0241 |
| TClass with AdaBoost/J48 (Kadous, 2002) | 0.0139 |
| Aligned Subsequence (Park et al. 2001) | 0.451 |
| Piecewise Normalization (Indyk et al. 2002) | 0.130 |
| Autocorrelation Functions (Wang & Wang 2000b) | 0.380 |
| Cepstrum (Kalpakis et. al. 2001) | 0.570 |
| String (Suffix Tree) (Huang & Yu 1999) | 0.206 |
| Important Points (Pratt & Fink 2002) | 0.387 |
| Edit Distance (Bozkaya et al.1997) | 0.603 |
| String Signature (Jonsson & Badal 1997) | 0.444 |
| Cosine Wavelets (Huntala et al. 1999) | 0.130 |
| Hölder (Struzik & Siebes) | 0.331 |
| Piecewise Probabilistic (Keogh & Smyth 1997) | 0.202 |

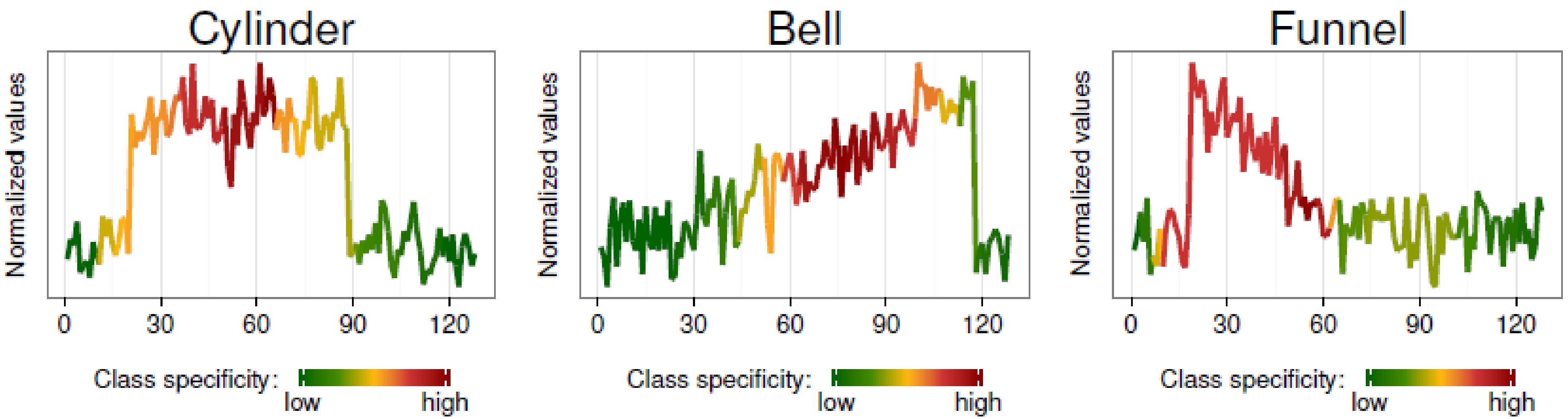
| | paa | Alphabet | WINDOW | acc | Error |
|---|---|---|---|---|---|
| 1410 | 6 | 7 | 40 | 1 | 0 |
| 1788 | 7 | 5 | 40 | 1 | 0 |
| 1789 | 7 | 5 | 42 | 1 | 0 |
| 1818 | 7 | 6 | 46 | 1 | 0 |
| 1871 | 7 | 8 | 44 | 1 | 0 |
| 1898 | 7 | 9 | 44 | 1 | 0 |
| 1900 | 7 | 9 | 48 | 1 | 0 |
| 1901 | 7 | 9 | 50 | 1 | 0 |
| 2191 | 8 | 4 | 36 | 1 | 0 |
| 2197 | 8 | 4 | 48 | 1 | 0 |
| 2219 | 8 | 5 | 38 | 1 | 0 |
| 2220 | 8 | 5 | 40 | 1 | 0 |




http://jmotif.googlecode.com/svn/trunk/RCode/cbf/par_surface.gif

# Contribution #5: Heatmap visualization in SAX-VSM

## CBF dataset class-characteristic subsequence segment visualization




- Each time series point  color intensity  is the sum of weights of point-containing subsequences.
  – If subsequence from the class, the weight value is added, if it is from other class, it is subtracted.

# Contribution #6:
# JMotif library, implements SAX-VSM and more

https://code.google.com/p/jmotif/

seni

**jMotif**

Main Language: Java

Total Lines of Code: 66,311

Active Contributors: 2

Commit Activity Timeline:


Updated Jan 13, 2015 — more at Open HUB

JMOTIF timeseries mining

# jmotif

A time series data-mining toolkit based on SAX and TFIDF statistics. Implements SAX-VSM.

Project Home | Downloads | Wiki | Issues | Source | Administer

Summary | People

**Project Information**

Starred by 48 users
Project feeds

**Code license**
GNU GPL v2

**Content license**
Creative Commons 3.0 BY-SA

**Labels**
timeseries, SAX, SAX-VSM, PAA, iSAX, tfidf, motif, discord, distance, clustering, jmotif, Academic, Java, R

**Members**
seninp, jessi.lin
1 committer

## Summary

JMotif implements in Java a number of methods for time series data handling, mining, and classification.

## News

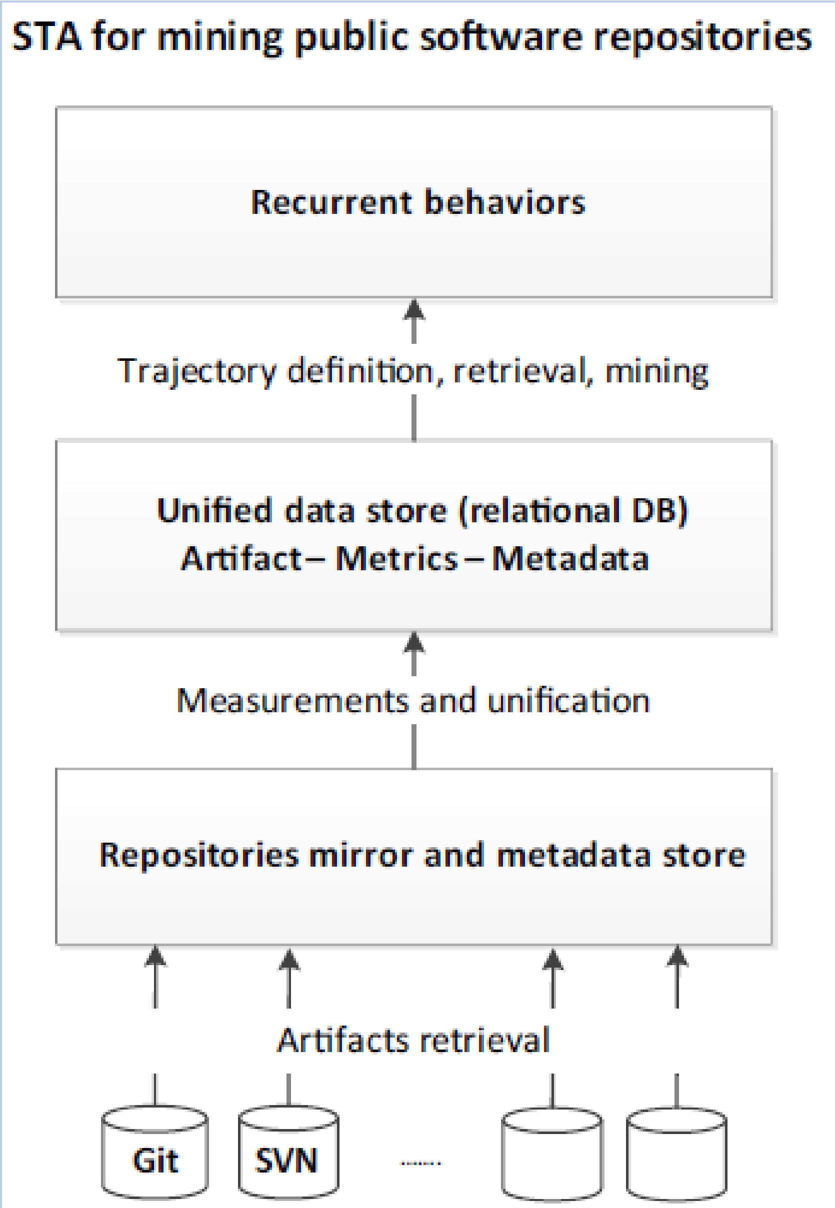
- **GrammarViz 2.0 has its own website and the code repository.**

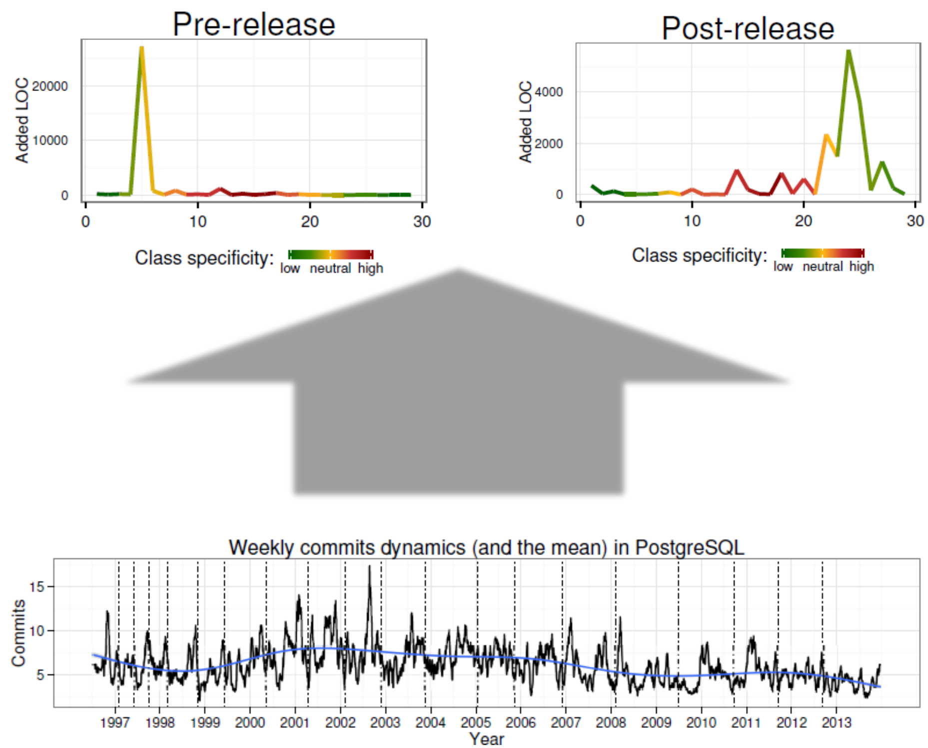
### Detailed summary

In particular, JMotif implements:

- **time series motif** discovery workflow based on Symbolic Aggregate Approximation (i.e. SAX)
- **time series discord** discovery workflow based on SAX, known as HOTSAX
- **time series characteristic patterns** discovery and classification workflow based on SAX, TF*IDF, and Vector Space Model (VSM), known as SAX-VSM
- **time series variable length motif and discord** discovery and visualization workflow based on SAX and Sequitur that extends the functionality of previously developed tool called GrammarViz

# SAX-VSM-based STA architecture



# STA evaluation experimental design:

1. Find software artifacts that can be labeled and are interesting for studying.

2. Measure them.

3. Synthesize software trajectory classes using labels.

4. Apply SAX-VSM learning procedure based on cross-validation, or to a splitting of the existing dataset into train and test data.

5. Try to make sense of discovered class-characteristic behaviors.

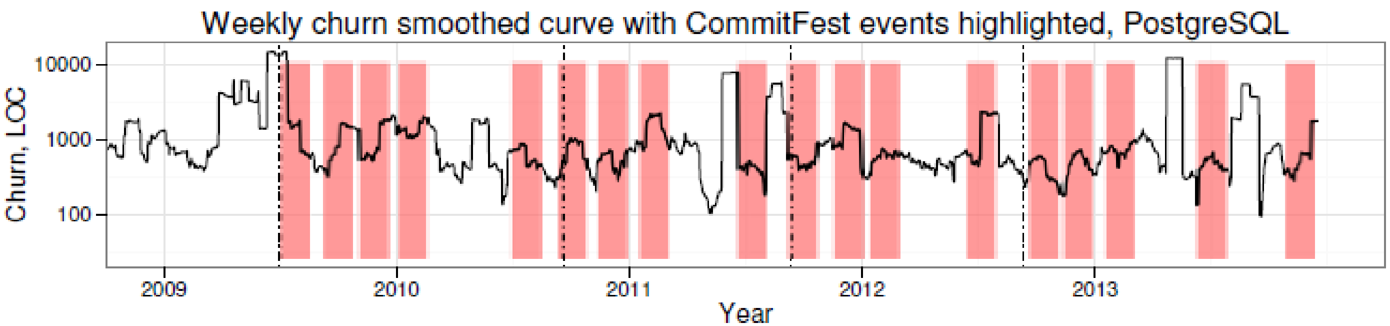
6. Conclude.

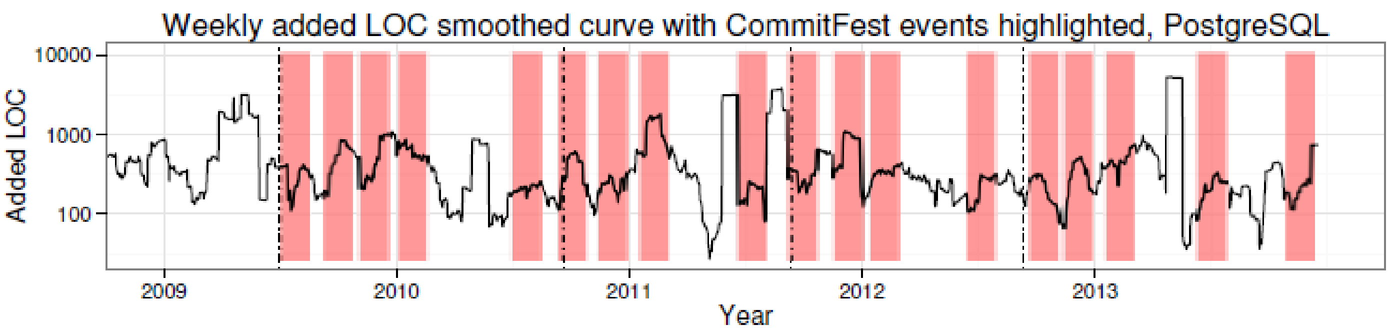
# Contribution #7: STA Case studies

- PostgreSQL:
  - CommitFest recurrent characteristic behavior discovery
  - Software Release recurrent characteristic behavior discovery
- Android OS:
  - Software Release characteristic recurrent behaviors discovery
- StackOverflow:
  - Top contributors daily activity behavior analysis using TF*IDF statistics

# PostgreSQL CommitFest recurrent behaviors
# case study design

PostgreSQL




1. Measure all change records.
2. Build software trajectories for CommitFest intervals and normal development cycle without differentiating contributors.
3. Run SAX-VSM parameters optimization using both classes.
4. Explore the discovered behaviors.

# PostgreSQL case study, results

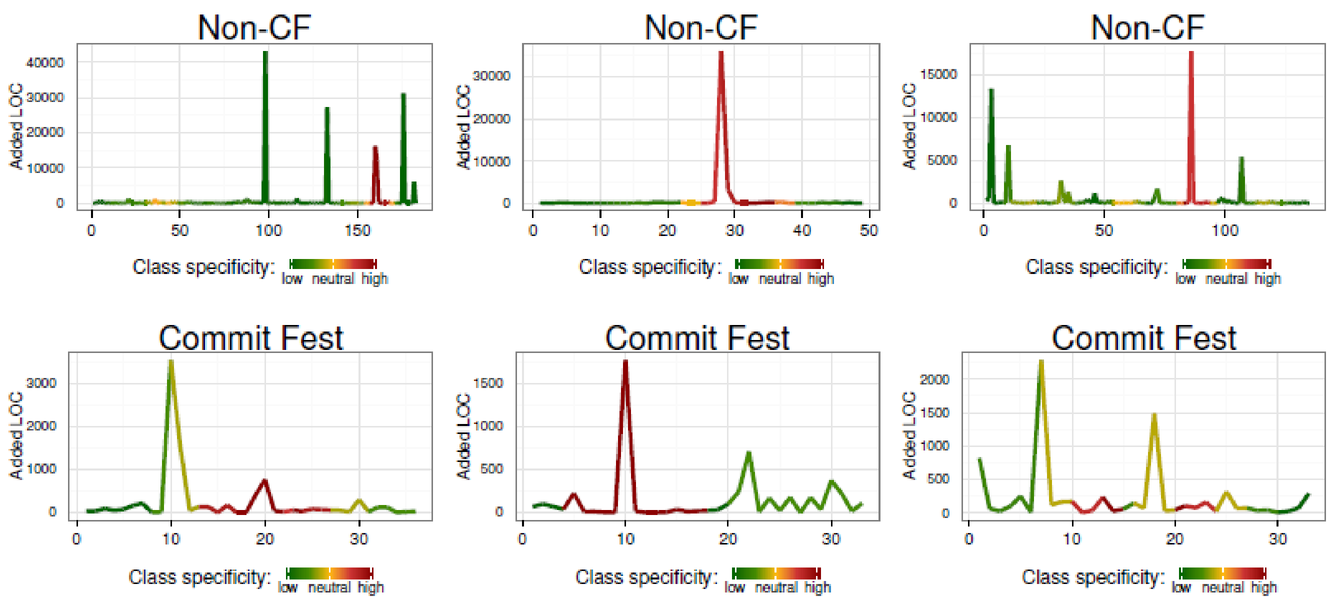
| Commit Fest behaviors experiment | | |
|---|---|---|
| Trajectory class | Discretization parameters | LOOCV accuracy |
| added LOC | 6,5,8 | 72.22% |
| edited LOC | 14,5,5 | **75.00%** |
| deleted LOC | 8,6,10 | **75.00%** |
| added files | 12,8,5 | 65.71% |
| edited files | 12,4,11 | 66.67% |
| deleted files | 27,7,3 | 55.17% |

| Software Release behaviors experiment | | |
|---|---|---|
| Trajectory class | Discretization parameters | LOOCV accuracy |
| added LOC | 14,5,7 | **80.56%** |
| edited LOC | 5,5,14 | 75.00% |
| deleted LOC | 10,5,11 | 72.22% |
| added files | 16,4,10 | 64.71% |
| edited files | 6,4,7 | **80.56%** |
| deleted files | 18,5,12 | 56.25% |

**Table 4.5**: The Leave One Out Cross Validation results for PostgreSQL aggregated trajectories. The discretization parameters are ordered as the sequence of sliding window size, PAA size, Alphabet size.

**Normal dev cycle –**
**Stand-alone spikes, no new code added**

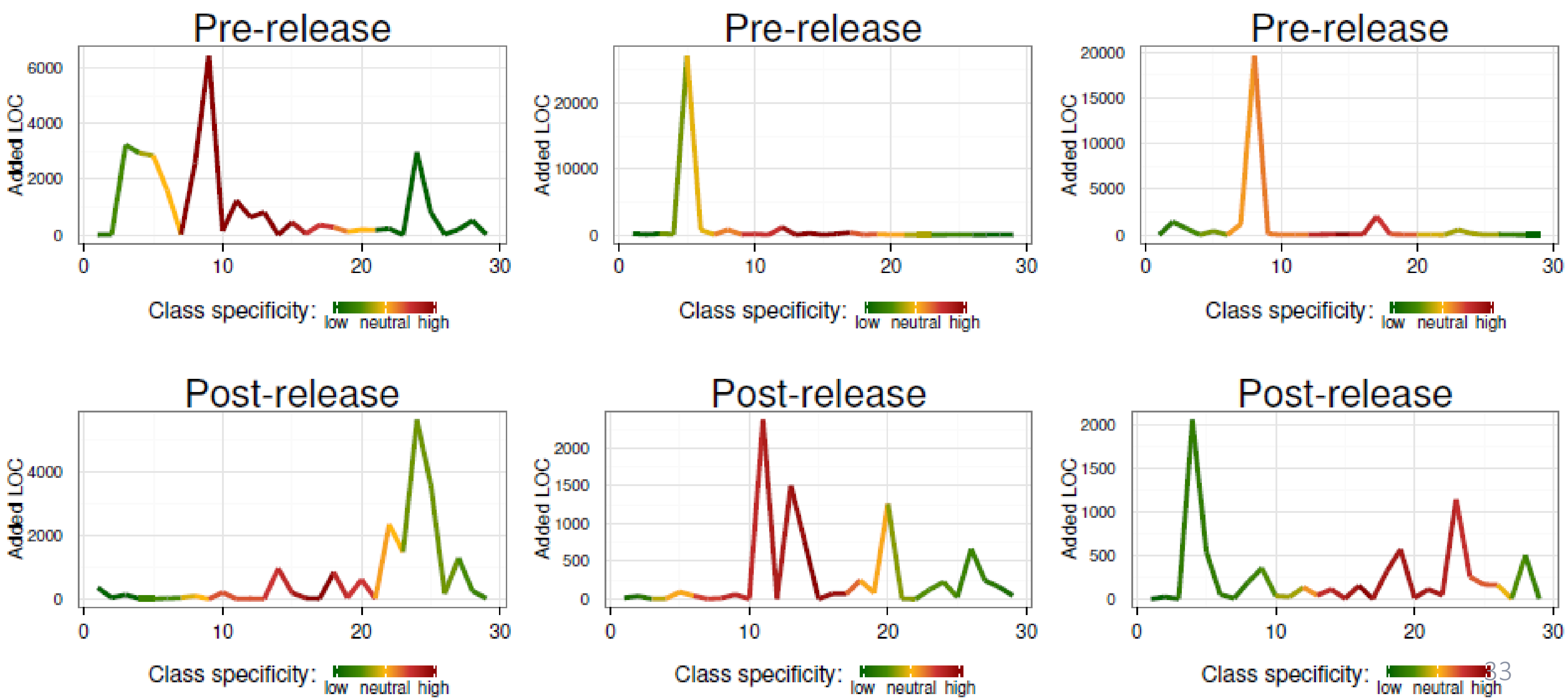
**CommitFest –**
**Continuous changes new code added daily**



# PostgreSQL Software Release case study

1. Measure all change records.
2. Build software trajectories for Pre-release using four weeks preceeding the release week.
3. Build software trajectories for Post-release using four weeks succeding the release week.
4. Run SAX-VSM parameters optimization on two classes.
5. Explore the discovered behaviors.

PostgreSQL



# Android OS software release case study

1. Measure all change records.
2. Randomly pick three releases and build **per-contributor trajectories** .
3. Run SAX-VSM LOOCV on Pre- and Post- trajectories.
4. Using the optimal parameters assess the performance of SAX-VSM classifier on other releases.

| Software metric | Train releases | Parameters | Accuracy | Note |
|---|---|---|---|---|
| added code lines | 1,3,5 | 18,7,12 | 54.00% | biased towards post- |
| added code lines | 4,6,9 | 15,15,5 | 58.33% | biased towards post- |
| added code lines | 5,8,11 | 12,10,10 | 66.66% | biased towards pre- |
| added code lines | 1,6,12 | 28,5,14 | 66.66% | biased towards pre- |
| edited lines | 1,3,5 | 24,10,4 | 62.50% | biased towards post- |
| edited lines | 4,6,9 | 24,5,12 | 58.33% | biased towards post- |
| edited lines | 5,8,11 | 22,7,7 | 62.50% | biased towards pre- |
| edited lines | 1,6,12 | 18,8,7 | 58.33% | biased towards pre- |
| deleted lines | 1,3,5 | 24,10,4 | 58.44% | biased towards pre- |
| deleted lines | 4,6,9 | 12,12,5 | **75.00**s% | |
| deleted lines | 5,8,11 | 24,5,7 | 61.50% | biased towards post- |
| deleted lines | 1,6,12 | 24,5,11 | 62.50% | biased towards pre- |

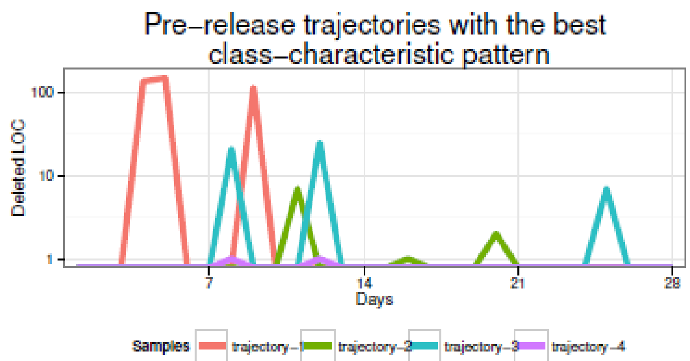
# Android OS case study, best class characteristic behaviors

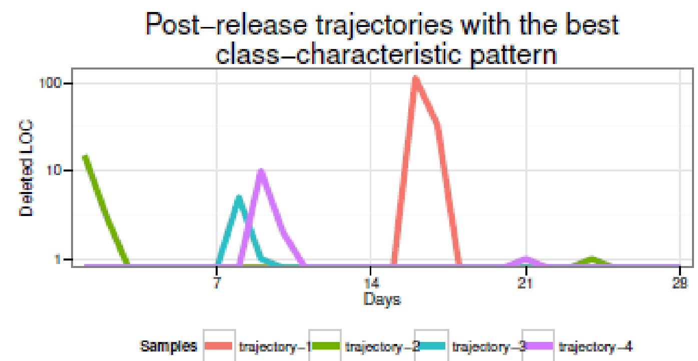
| Pre-release centroid pattern | weight |
|---|---|
| ebbbebbbbbbb | 0.1748588272 |
| bbbbbcbbbebb | 0.1083403654 |
| bbbbbbbdebbb | 0.0901908199 |
| bbbbbbbbdebb | 0.0901908199 |
| bbbbbbdebbbb | 0.0901 |
| . . . | . . |

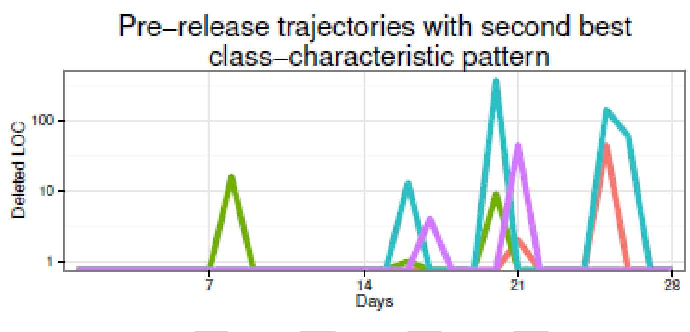
| Post-release centroid pattern | weight |
|---|---|
| edbbbbbbbbbb | 0.1995655982 |
| bbbbbebbcbbb | 0.1533399084 |
| bbbbbbebbcbb | 0.1533399084 |
| bbbbbbbebbcb | 0.1533399084 |

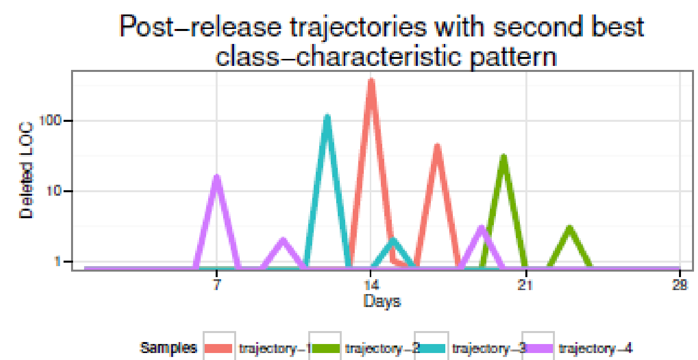
**Window: 12**
**PAA: 12**
**Alphabet: 5**

**Accuracy:**
**75%**




Behaviors are difficult to interpret

# John Skeet phenomenon:

## daily 200+ in reputation, for years

meta stackoverflow

Jon Skeet Facts

- Jon Skeet can divide by zero
- Jon Skeet IS the traveling salesman –
  only he knows the shortest route
- When Jon Skeet's code fails to compile,
  the compiler apologizes
- When invoking one of Jon's callbacks,
  he runtime adds "please”

# StackOverflow study design

The goal is to see if there are some interesting behaviors among Stackoverflow top contributors.

1. Account for answers submitted by 4 top contributors

2. Create software trajectories per contributor SO lifetime

3. No sliding window used.
   Software trajectory is chopped by one day window.
   24 hours aggregated into 8 points, discretized into a word, A=3.
   Bags of words built for each contributor.

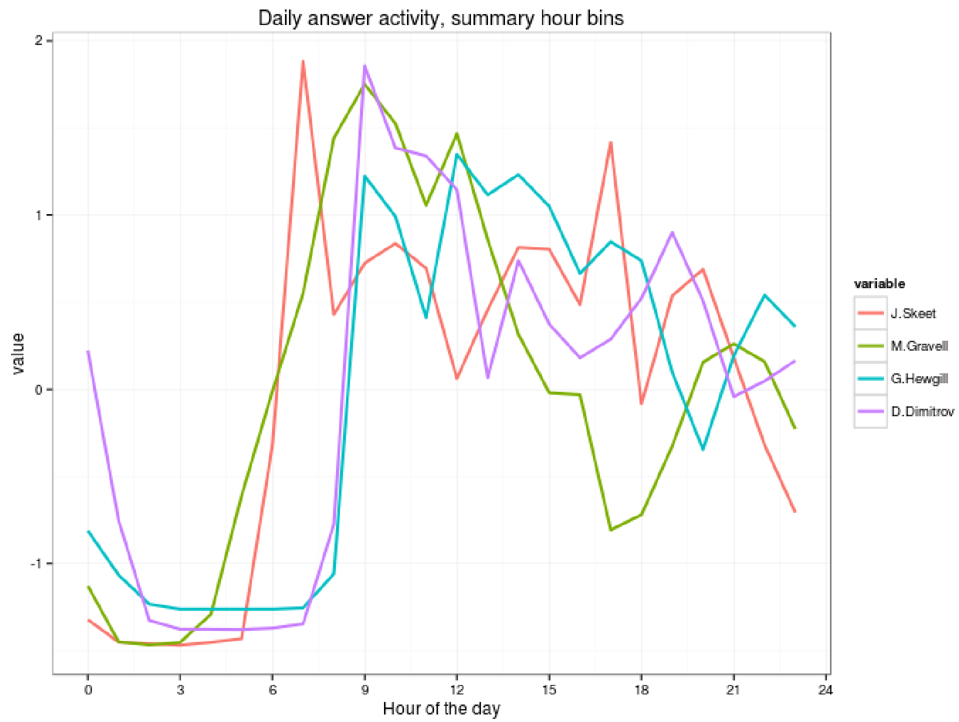
4. TF*IDF computed for four word bags.

# Daily Answers counts for four StackOverflow top contributors

Sliding window=24hrs

PAA=8 (bins by 3 hrs), Alphabet=3

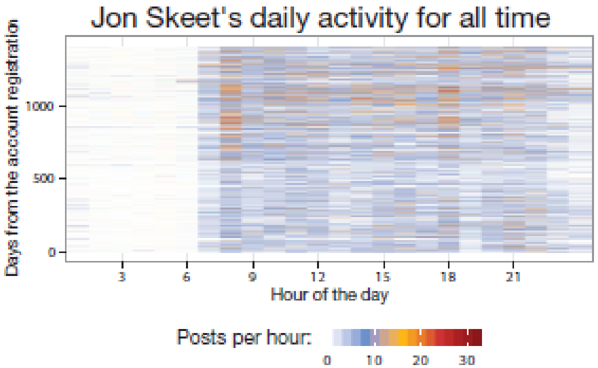
| | J.Skeet | M.Gravell | G.Hewgill | D.Dimitrov |
|---|---|---|---|---|
| 1 aabbbccb | 36 | 3 | 0 | 0 |
| 2 aaccbbbb | 30 | 13 | 3 | 0 |
| 3 aabbccbb | 30 | 2 | 5 | 0 |
| 4 aabccbbb | 29 | 19 | 6 | 6 |
| 5 aacbbcbb | 29 | 0 | 3 | 0 |
| 6 aabbbcbb | 27 | 5 | 7 | 0 |
| 7 aaccbcbb | 24 | 1 | 0 | 0 |
| 8 aabcbcbb | 23 | 5 | 8 | 2 |
| 9 aacbbccb | 23 | 0 | 1 | 0 |
| 10 aacbbcba | 22 | 2 | 3 | 0 |
| 11 aabcbbbb | 18 | 27 | 19 | 1 |
| 12 bbbcbbbb | 1 | 27 | 52 | 29 |
| 13 aabccbbb | 29 | 19 | 6 | 6 |
| 14 aaccbabb | 8 | 16 | 3 | 0 |
| 15 aaccbbba | 21 | 15 | 5 | 0 |
| 16 aabccabb | 5 | 15 | 2 | 0 |
| 17 aabccbab | 1 | 15 | 7 | 2 |
| 18 aabccbba | 20 | 14 | 9 | 6 |
| 19 aacccbaa | 6 | 14 | 6 | 0 |
| 20 aabcbabb | 3 | 14 | 3 | 0 |
| 21 bbcbbbbb | 4 | 9 | 57 | 1 |
| 22 bbbcbbbb | 1 | 27 | 52 | 29 |
| 23 bbbbcbbb | 0 | 6 | 26 | 17 |
| 24 bbbccbbb | 1 | 9 | 24 | 14 |
| 25 aabcbbbb | 18 | 27 | 19 | 1 |
| 26 bbbbbbcb | 4 | 10 | 18 | 53 |
| 27 bbccbbbb | 2 | 10 | 18 | 1 |
| 28 bbbbbcbb | 2 | 7 | 18 | 20 |
| 29 aabccbaa | 3 | 7 | 16 | 3 |
| 30 bbbbbbbc | 3 | 8 | 15 | 35 |
| 31 bbbbbbcb | 4 | 10 | 18 | 53 |
| 32 bbbbbbbc | 3 | 8 | 15 | 35 |
| 33 aaaccbbb | 5 | 9 | 3 | 30 |
| 34 bbbcbbbb | 1 | 27 | 52 | 29 |
| 35 bbbbbcbb | 2 | 7 | 18 | 20 |
| 36 aaaccbba | 0 | 5 | 3 | 20 |
| 37 bbbbcbbb | 0 | 6 | 26 | 17 |
| 38 aaacccba | 0 | 1 | 2 | 17 |
| 39 aaacbcbb | 4 | 4 | 1 | 16 |
| 40 bbbbbbcc | 3 | 3 | 12 | 15 |

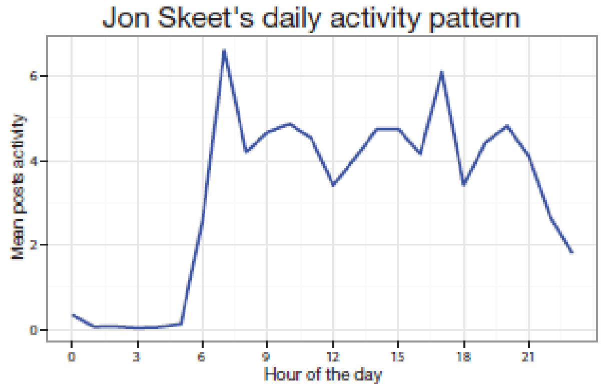



## Cosine distances

| | J.Skeet | M.Gravell | D.Dimitrov | G.Hewgill |
|---|---|---|---|---|
| J.Skeet | 0 | 0 | 0 | 0 |
| M.Gravell | 0,1699 | 0 | 0 | 0 |
| D.Dimitrov | 0,0096 | 0,04568 | 0 | 0 |
| G.Hewgill | 0,13533 | 0,18094 | 0,1533 | 0 |

# Since behaviors representation is symbolic, why not to try some existing visualization for symbolic data?





Activity levels: A-low, B-medium, C-high.

- *...I have a longish commute both ways each day: a 3G data dongle lets me answer questions during that time...*

- *...I spend a fair amount of time in the evening on my computer for whatever reason (coding, writing talks or articles, etc) - I pop onto SO every so often...*

- *...While at work, I tend to check SO while I have tests running, a deploy, or a build. I hope my colleagues wouldn't regard me as a slacker though...*

- Well, there is nothing special in the behavior -- a common working day activity pattern. It is really self-motivation and persistence.
  No special magic in the recurrent behavior was detected.

# STA evaluation summary

- Pros:
  - STA is capable to discover and to rank recurrent behaviors – the dissertation research claim is supported.

- Cons:
  - The discovered behaviors are sometimes difficult to interpret. More project-specific knowledge required.

# Contributions of my research

1. **Software Trajectory Analysis framework**
   - Peer-reviewed publication in ESEM-2010 doctoral symposium
2. **SAX-VSM – an algorithm for class-characteristic behaviors discovery**
   - Peer-reviewed publication in ICDM-2013
3. **SAX-VSM implementation and experimental evaluation, including the performance evaluation in hierarchical and k-means clustering**
   - Peer-reviewed publication in ICDM-2013, partially unpublished
4. **SAX-VSM discretization parameters optimization scheme**
   - Peer-reviewed publication in ICDM-2013, partially unpublished
5. **Novel time-series class-characteristic heatmap-like visualization**
   - Peer-reviewed publication in ICDM-2013
6. **JMotif, an open-source Java library for the time series classification, recurrent and rare patterns discovery, and the time series grammatical decomposition**
   - Peer-reviewed publications in ECML/PKDD 2014, EDBT 2015
7. **Software Trajectory Analysis empirical evaluation**
   - Publication in SERP-2012
   - Future publication summarizing the dissertation

# Future work

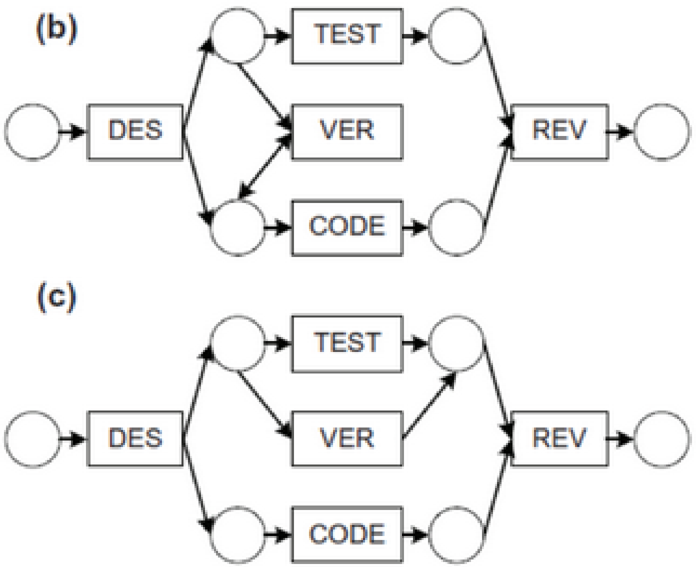
- Done already, based on the dissertation research:
  - a tool for recurrent behaviors discovery in a real-time from a live stream of software artifact measurements (Grammarviz 2.0, ECML/PKDD 2014)
  - a novel algorithm for the unusual behavior detection, i.e. anomaly, EDBT 15.

- In the queue:
  - SAX-VSM -**G** – an algorithm for time series classification and variable-length characteristic patterns discovery based on SAX, VSM, and grammatical inference.
  - Kolmogorov-complexity based patterns ranking (MDL principle). I hope that it will significantly improve the quality of discovered patterns.

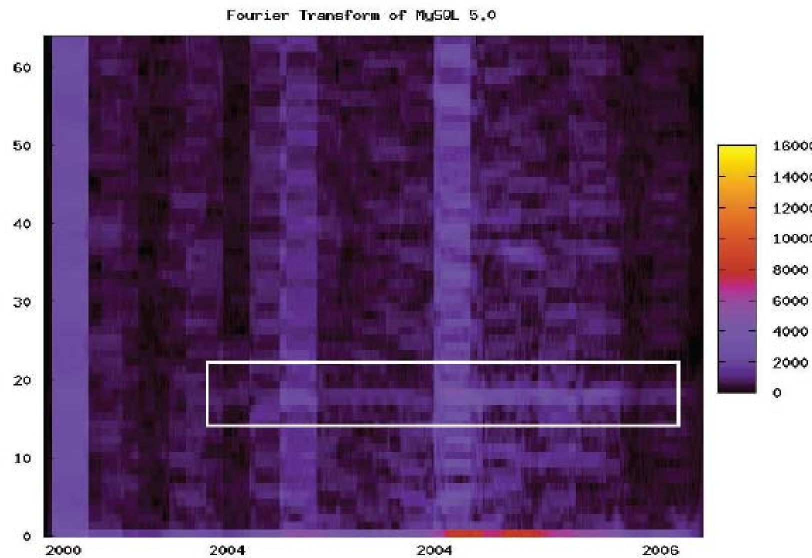
# Thank you!

# Related work in automated behavior discovery

## Event log analysis, Cook, Wolf, Dongen, Aalst




## Fourier-transform based decomposition, Hindle



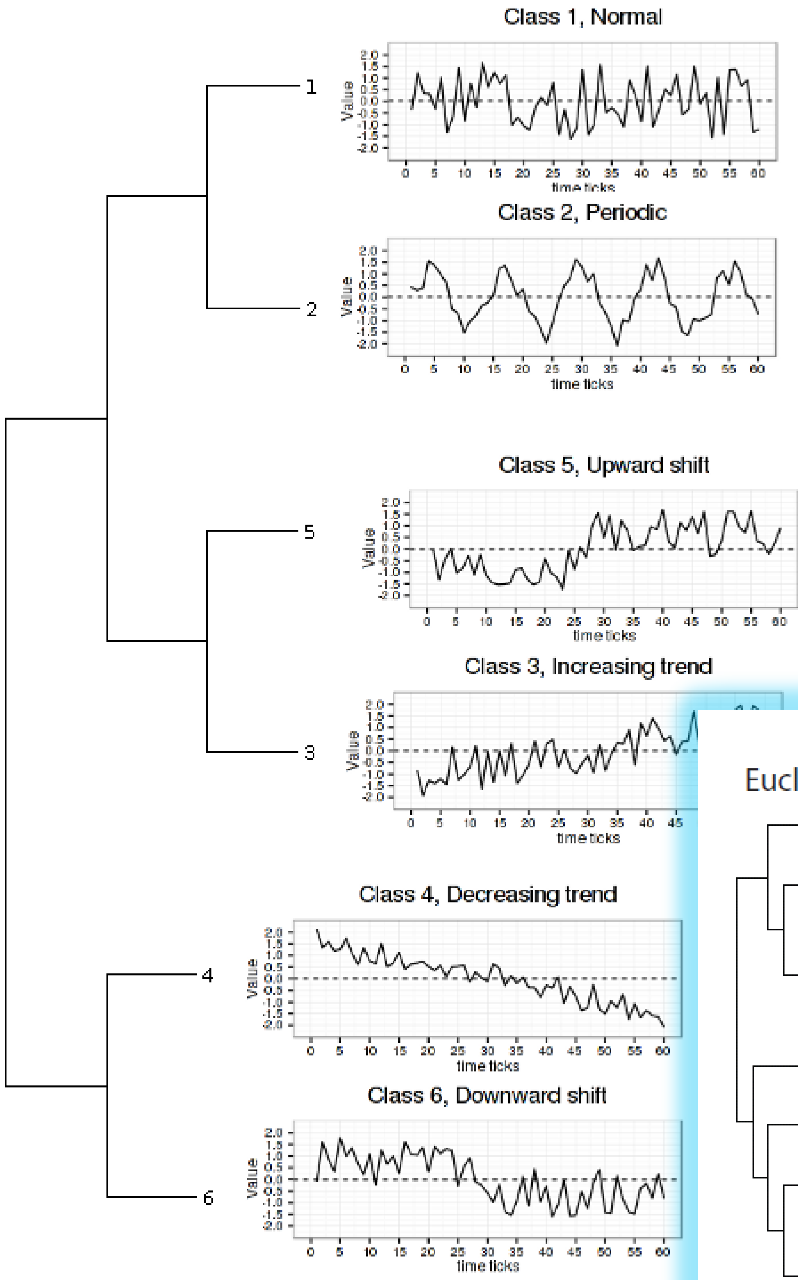

## STDB-notation, Hindle

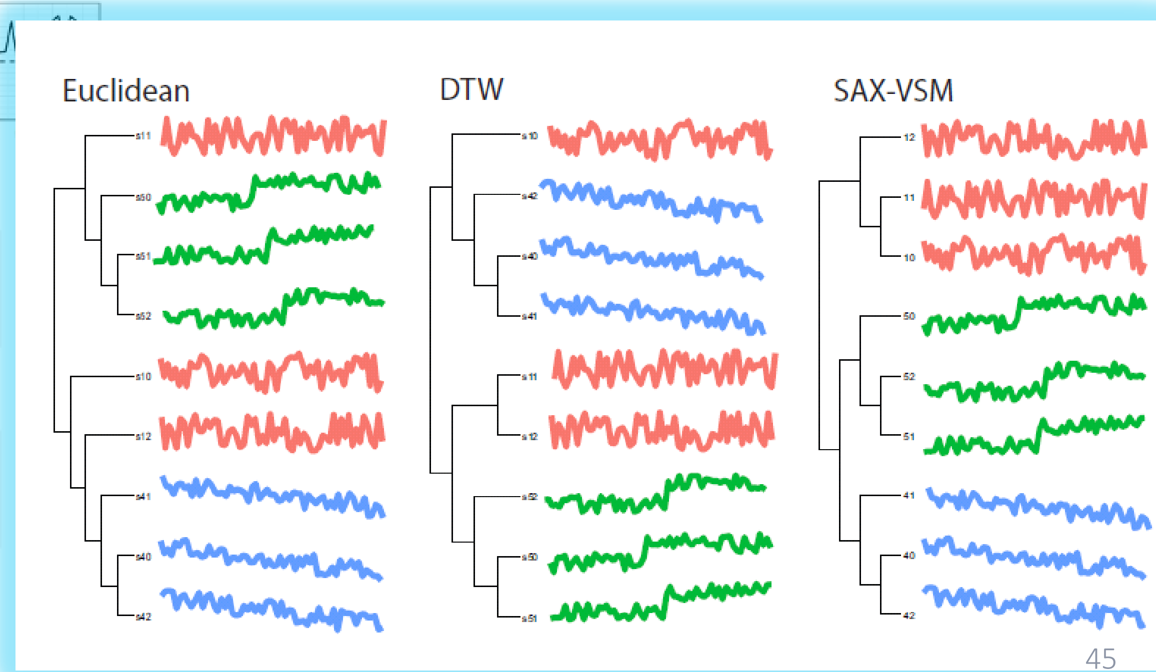
| Project | Major | Minor | All |
|---|---|---|---|
| Firebird | S▼T▲B▼D▲ | S▼T▼B▲D▼ | S▼T▲B▼D▲ |
| MaxDB | S□T▼B▼D□ | S□T▲B□D□ | S□T□B□D▼ |
| MySQL | S▼T▼B▲D▼ | S▼T▼B□D▼ | S▼T▼B□D▼ |
| PostgreSQL | S▲T▲B▲D▲ | S▲T▼B▼D▲ | S▲T▼B▲D▲ |

# SAX-VSM hierarchical Clustering:

# UCR Synthetic Control,

## each input time-series -> bag of words
## TF*IDF ranking
## cosine similarity-based clustering



# SAX-VSM, k-Means Clustering

## updating/normalizing centroids after each iteration
## i.e. "spherical k-means"

```
cluster #0 label: bell3
cluster #1 label: funnel2
cluster #2 label: funnel1

clusters centroids:
          "0",      "1",      "2"
"accc",  0.00000,  0.01349,  0.01349
"accb",  0.00000,  0.61944,  0.61947
"acba",  0.00000.  0.37271,  0.37270
"bcca",  0.74570,  0.00000,  0.00000
"caaa",  0.08387,  0.00000,  0.00000
"bacc",  0.15158,  0.00000,  0.00000
"bacb",  0.05595,  0.01349,  0.01349
"aaac",  0.00000,  0.22743,  0.22742
"abca",  0.36160,  0.00000,  0.00000
"aacb",  0.00000,  0.65214,  0.65212
"bcaa",  0.52918,  0.00000,  0.00000
```

```
cluster #0 [bell0, bell2, bell1, bell3]
cluster #1 [funnel0, funnel2]
cluster #2 [cylinder0, cylinder1, funnel1, funnel3, cylinder2]
```

```
cluster #0 [bell0, bell2, bell1, bell3]
cluster #1 [funnel1, funnel0, funnel3, funnel2]
cluster #2 [cylinder0, cylinder1, cylinder2, cylinder3]
```

Random centroids assignment made two clusters of the same class

Nevertheless, clustering recovered

Currently I employ further first strategy and restarting