Longitudinal Analysis of Collaboration Graphs of Forked Open Source Software Development Projects

Amirhosein (Emerson) Azarbakht
School of Electrical Engineering and Computer Science
Oregon State University
azarbaka@oregonstate.edu

Great Projects

Assumption: You know what open source is.

- Technical quality
  - duh?
- Operational health
  - Ongoing ability to incorporate new code contributions and new developers, and to be responsive to incoming bug reports
- Survivability
  - Ability to exist independently of any individual participant or sponsor; even if all of its founding members were to move on to other things

Open source is great
but not all the time

- Not everything works smoothly all the time in open source projects
- **Problems**:
  - Uncertainty?
  - Will the vendor stick around?
  - “Industry is just as afraid of "fly-by-night" open source packages as it is of fly-by-night vendors that may not be around tomorrow.”

What is this about?

My research focuses on **communication, collaboration problems**, especially

- What happens to trigger forking
- Can we do something to predict when the community started to disintegrate

What is this about?

Big Picture

- Developing complex software systems is complex
- Software Developers (SD) interact
  - They can have same/different goals, communication styles, values
- Interactions can be healthy or troubled
- Troubled interactions cause troubled communities -> failure
  - Some of these failures manifest as forks
  - Failure affects many people; developers and users

Can we save troubled projects?

What is forking?

- “When a part of a development community (or a third party not related to the project) starts a completely independent line of development based on the source code basis of the project.” (think "fork in the road".)

- It can be undesirable:
  - Dilution of workforce
  - Flaming & unhealthy dynamics
  - Redundant work

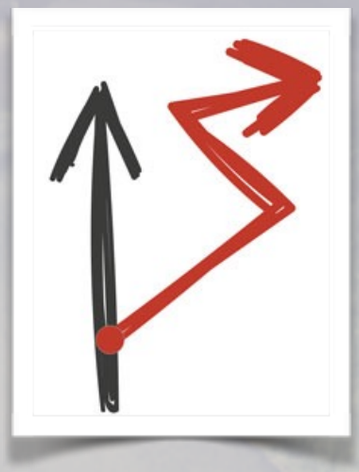
} people suffer

The more serious the threat of a fork becomes, the more willing people are to compromise to avoid it.

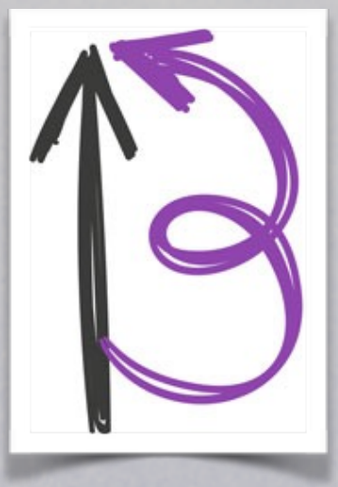
Forking Categories

Communities dissolve or evolve for a variety of different reasons:

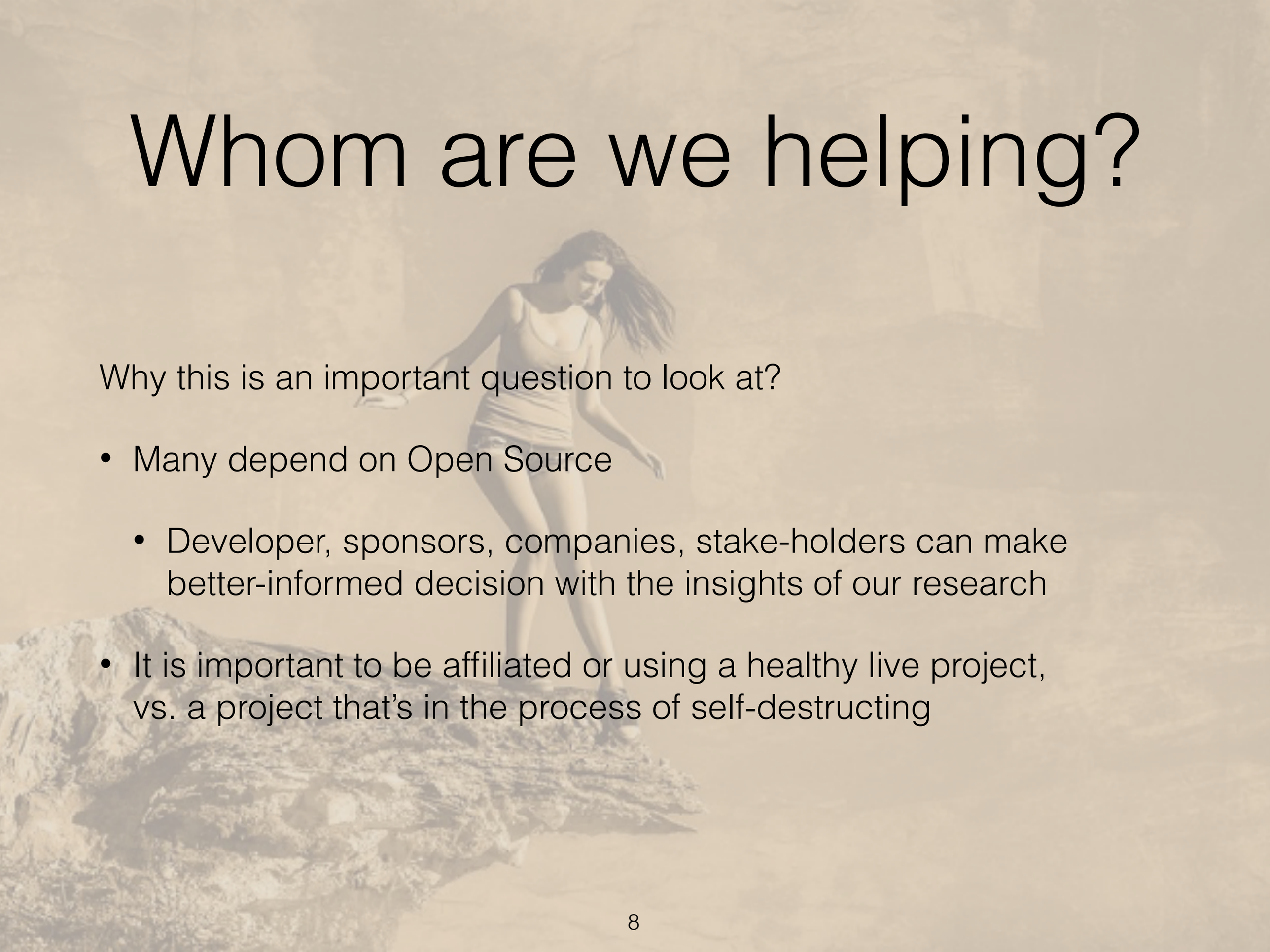
- **Conflict-driven**
  - Destructive
  - Lead to resentment
  - The forks are direct competitors, “you’re either with us, or against us”



- **Non-conflict-driven**
  - Developers interested in exploring different functionalities, or different ways of doing things
  - Not necessarily creating an all-or-nothing type of thing;
  - Developers participate in multiple communities, as long as it’s a constructive work



Whom are we helping?



Why this is an important question to look at?

- Many depend on Open Source
- Developer, sponsors, companies, stake-holders can make better-informed decision with the insights of our research
- It is important to be affiliated or using a healthy live project, vs. a project that's in the process of self-destructing

Social Health is Important

It'd be great if we could predict

- Whether our community is likely to fork? or
- When they are likely to fork?
- So that we can design some sort of intervention, Or,
- If you are an industry person, or even an individual contributor, to give you an objective external view of whether this is a community you want to get involved in, or whether you don't, because getting involved, or using a software requires a level of investment.

For all these reasons, and for all these stake-holders, understanding the social health, and social health evolution of projects is really important.

Related Work

What we can know about FOSS	What we do *not know*
Identifying knowledge brokers [47]	Static point of view; People are important, but how do these roles change, how they move around around
Post-forking porting of new features or bug fixes [7]	
Social structures and dynamics [10]	How these structures change through time
Identifying “key” events [1]	Compared only two snapshots; not longitudinal
Communication patterns [24]	Change through time?

Gap: the important **Run-up to forks** seldom studied

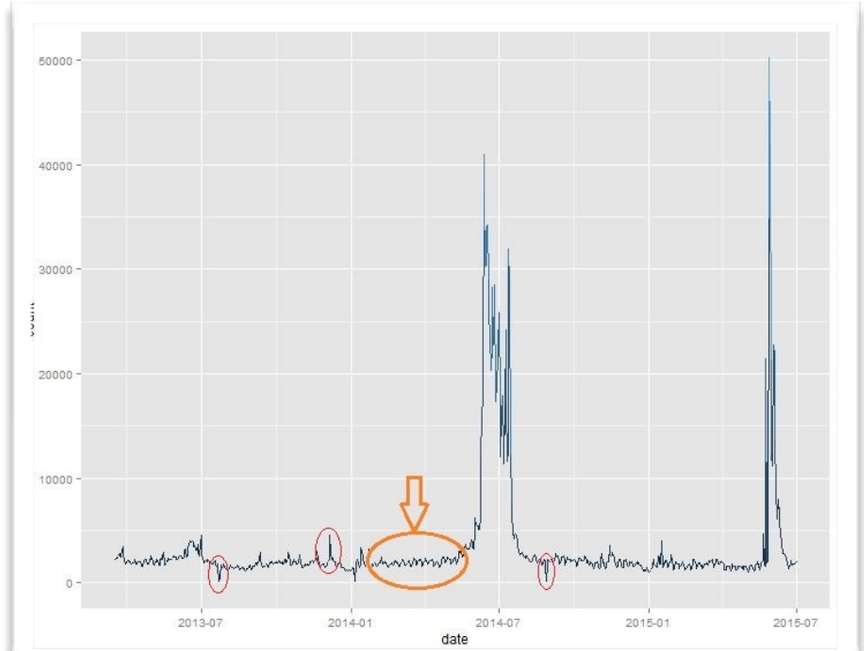
Research Goals

- The objective is to be able to intervene, or evaluate the health
- We need to figure out, are there key indicators left in the record, that can reliably be used to predict, or to identify problems?
  - Why these are important:
    Because if we know these, we could identify unhealthy dynamics and fix them before it's irreversible.

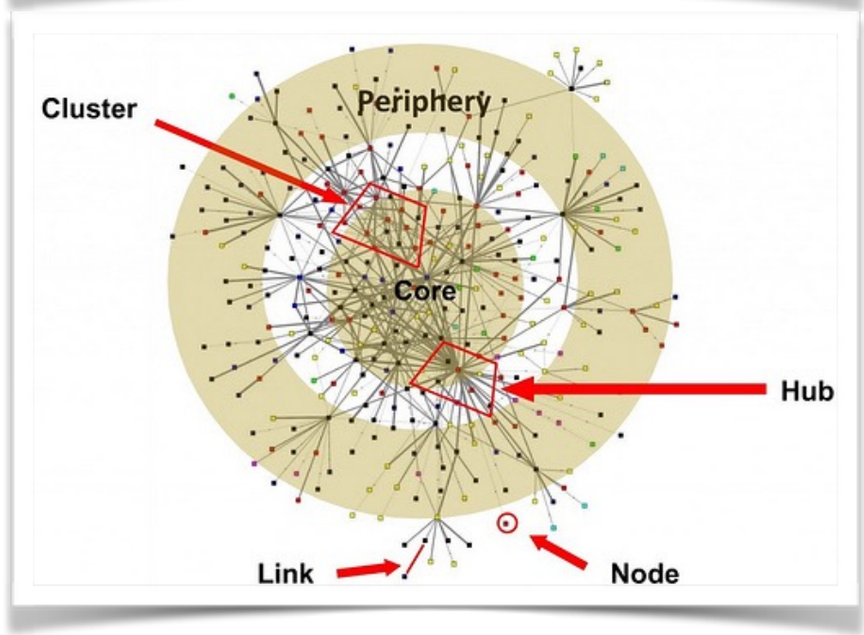
Methodology

I did the following two types of analysis:

- Time series analysis of open source developers' collaboration/ communication sentiments



- Social network analysis of open source developers' interaction graphs



Data Collection Projects

Projects	Reason for forking	Year forked	Type
Kamailio & OpenSIPS	Differences among developer team	2008	U.F.
ffmpeg & libav	Differences among developer team	2011	U.F.
Asterisk & Callweaver	More community-driven development	2007	U.F.
rdesktop & FreeRDP	More community-driven development	2010	U.F.
freeglut & OpenGLUT	More community-driven development	2004	U.F.
Amarok & Clementine Player	Technical (Addition of functionality)	2010	H.F.
Apache CouchDB & BigCouch	Technical (Addition of functionality)	2010	H.F.
Pidgin & Carrier	Technical (Addition of functionality)	2008	H.F.
MPlayer & MPlayerXP	Technical (Addition of functionality)	2005	H.F.
Ceph	Not forked	-	No.F.
Python	Not forked	-	No.F.
OpenStack Neutron	Not forked	-	No.F.
GlusterFS	Not forked	-	No.F.

Conflict-driven

Non-conflict-driven
Community-driven

Non-conflict-driven
Technical

No.F.

Methodology

Time series analysis of open source developers' communication sentiments

Here's my data:

- The contents of the messages sent and received on the projects developers mailing list, for the time period of 10-month run-up to the fork.

Here's why it matters:

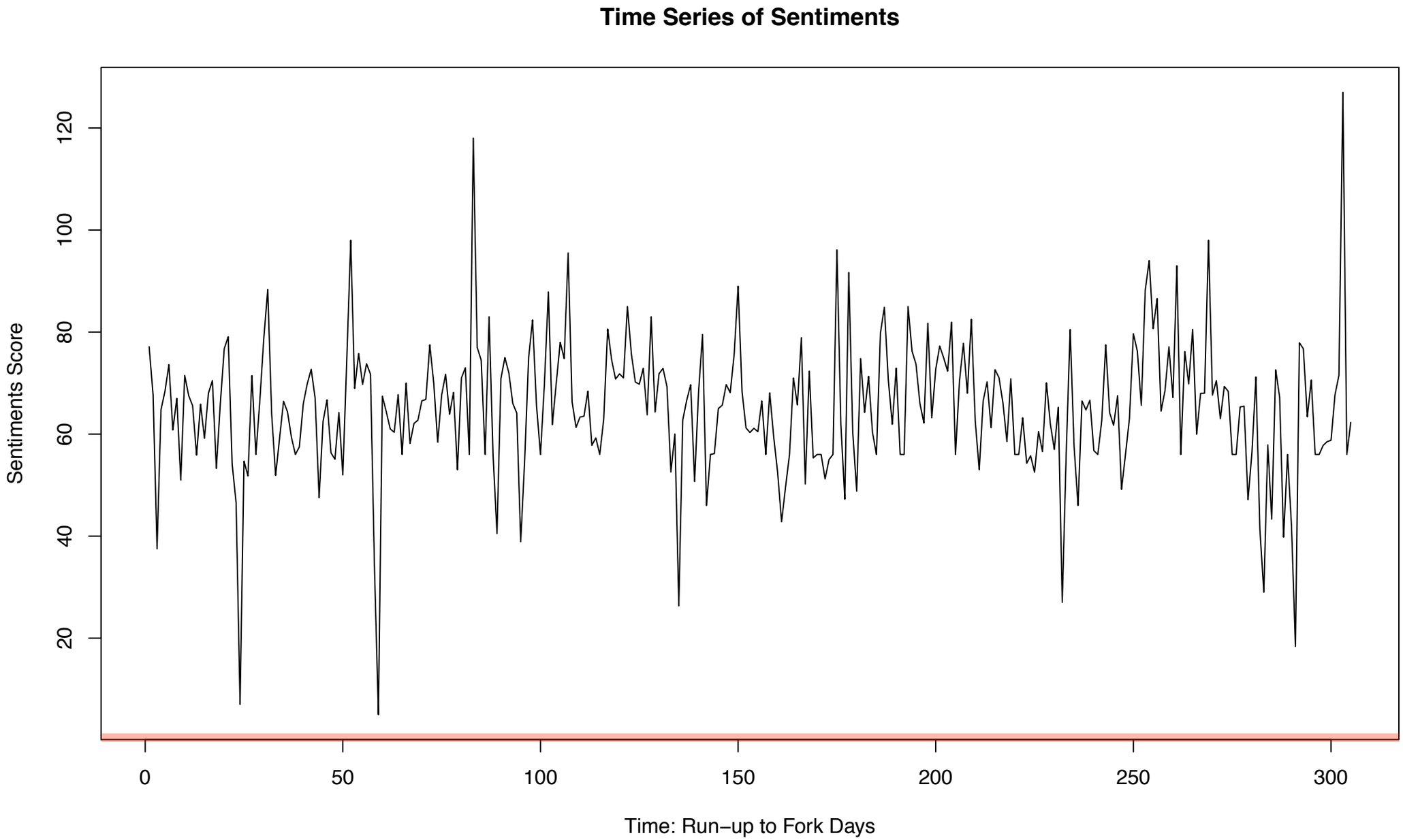
- The objective was to be able to intervene, or evaluate the health.

- Anomaly detection of the time series analysis may be used by the project stakeholders/investors as key indicators left in the record, that can reliably be used to predict, or to identify problems among the developers, and intervene if need be.

Sentiment Analysis

- Sentiment analysis
- Contents of messages sent/received by contributors
- A time series of sentiments, to analyze

Time Series Analysis of Sentiments



Sentiment Analysis

- To determine the attitude of a writer, the overall contextual polarity or emotional reaction to a document, interaction, or event

Used for

- “Flame” detection
- Bias identification in news sources
- Identifying (in)appropriate content for ad placement
- Targeting advertising/messages, gauging reactions, etc.

Sentiment Analysis:
Time Series Analysis

Used in

- Census analysis
- Business forecasting: Understanding fluctuations in sales
- Disease incidence tracking
- Monitoring unemployment rate; as an economic indicator used by decision makers

Sentiment Analysis:
Time Series Analysis

Used to

- **To understand the past, and predict the future**

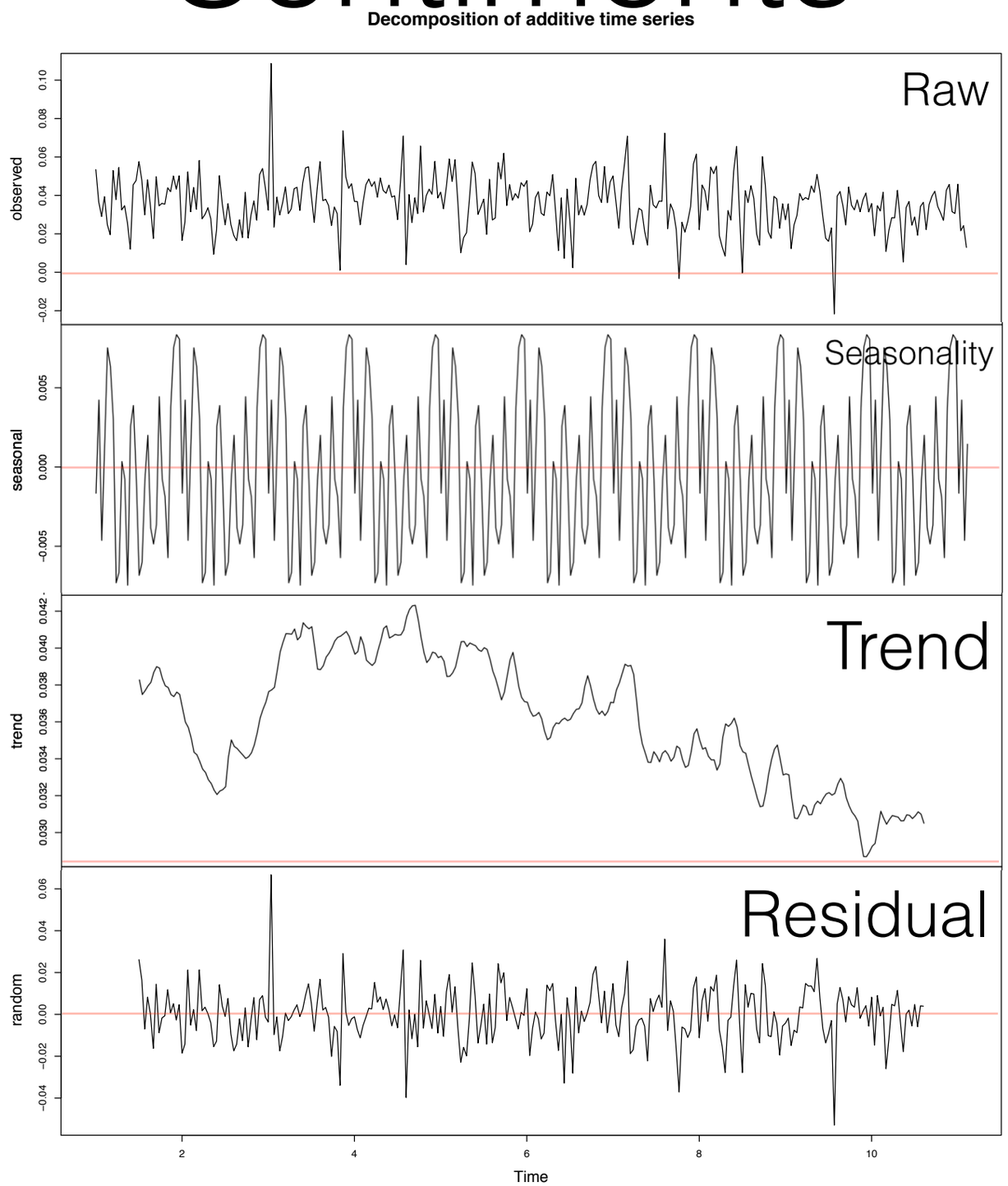
- **Forecasting** (predicting inference, a subset of statistical inference). assumes that present trends continue. This assumption cannot be checked empirically, but, when we identify the likely causes for a trend, we can justify the forecasting(extrapolating it) for a few time-steps at least

- **Anomaly detection**

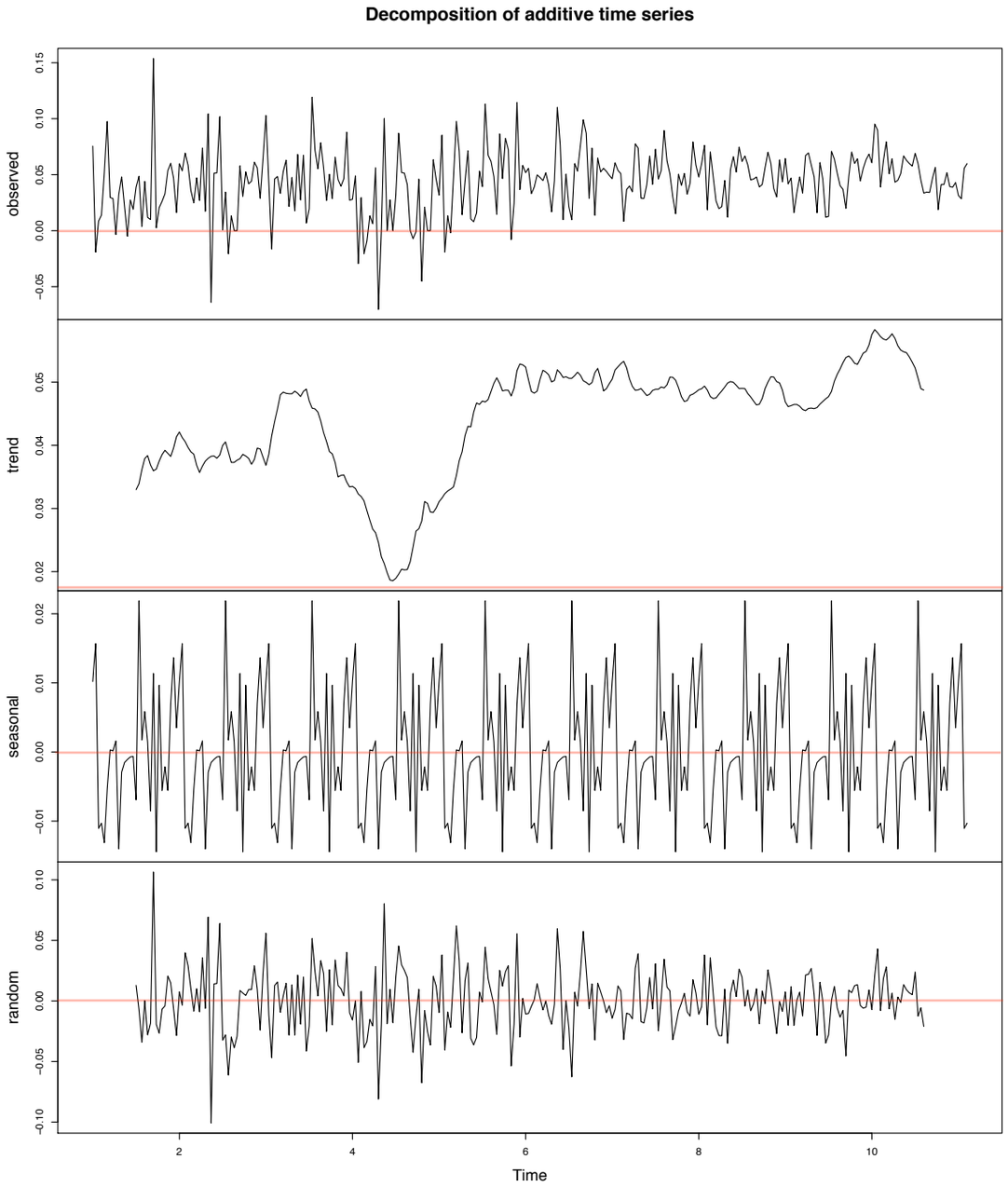
- Classification (assigning a time series pattern to a specific category: e.g. gesture recognition of hand movements in sign language videos)

- Query by content ~ content-based image retrieval

Time Series Analysis of Sentiments

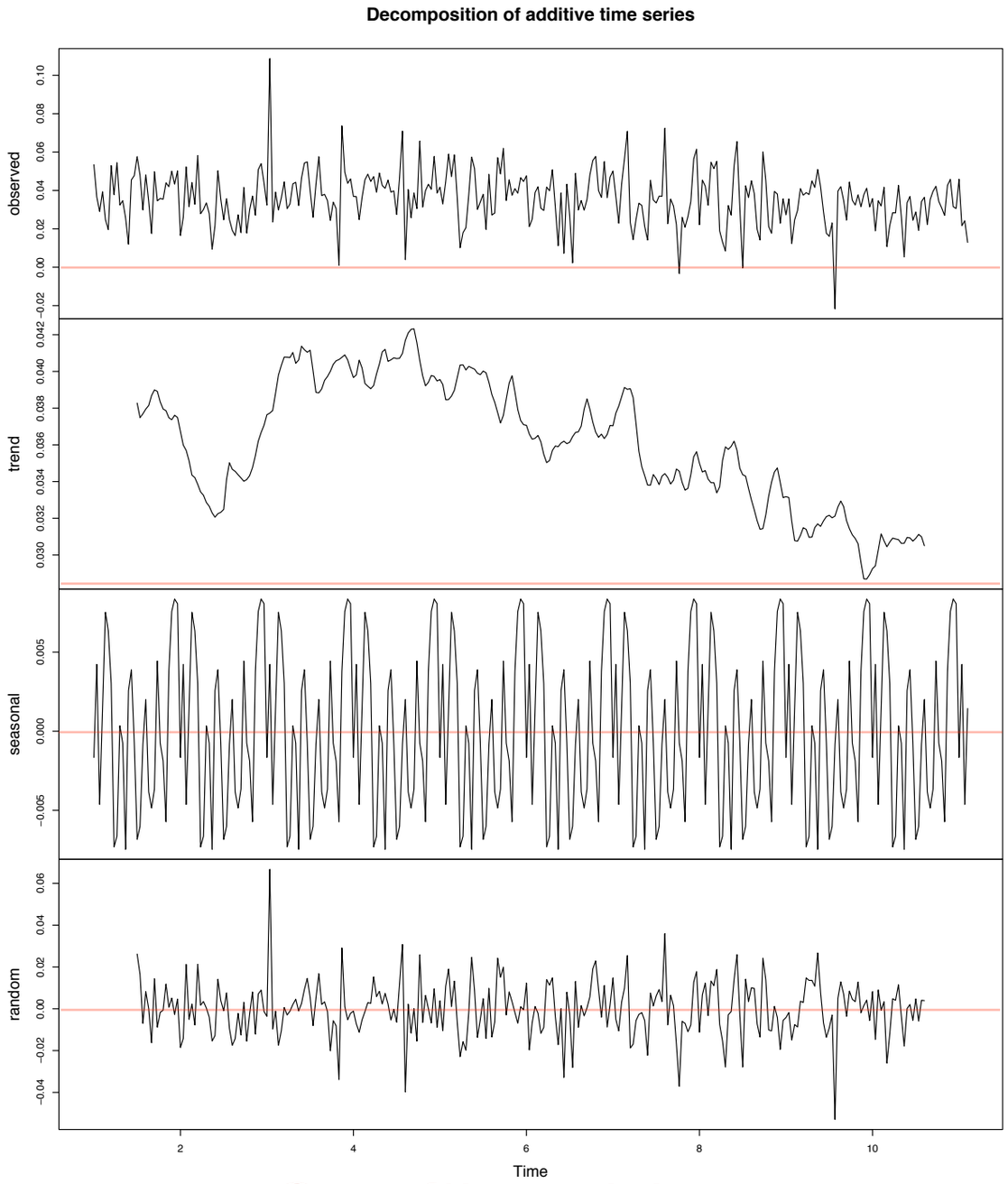


Time Series Analysis of Sentiments



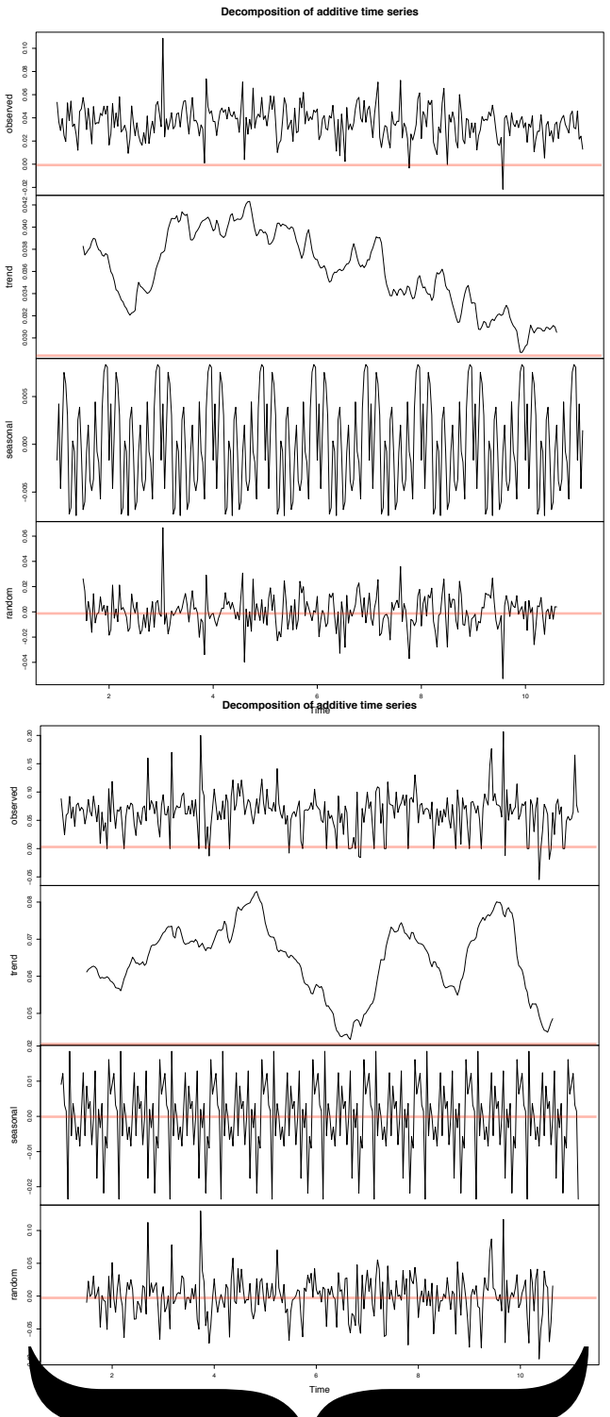
Non-conflict-driven

vs.

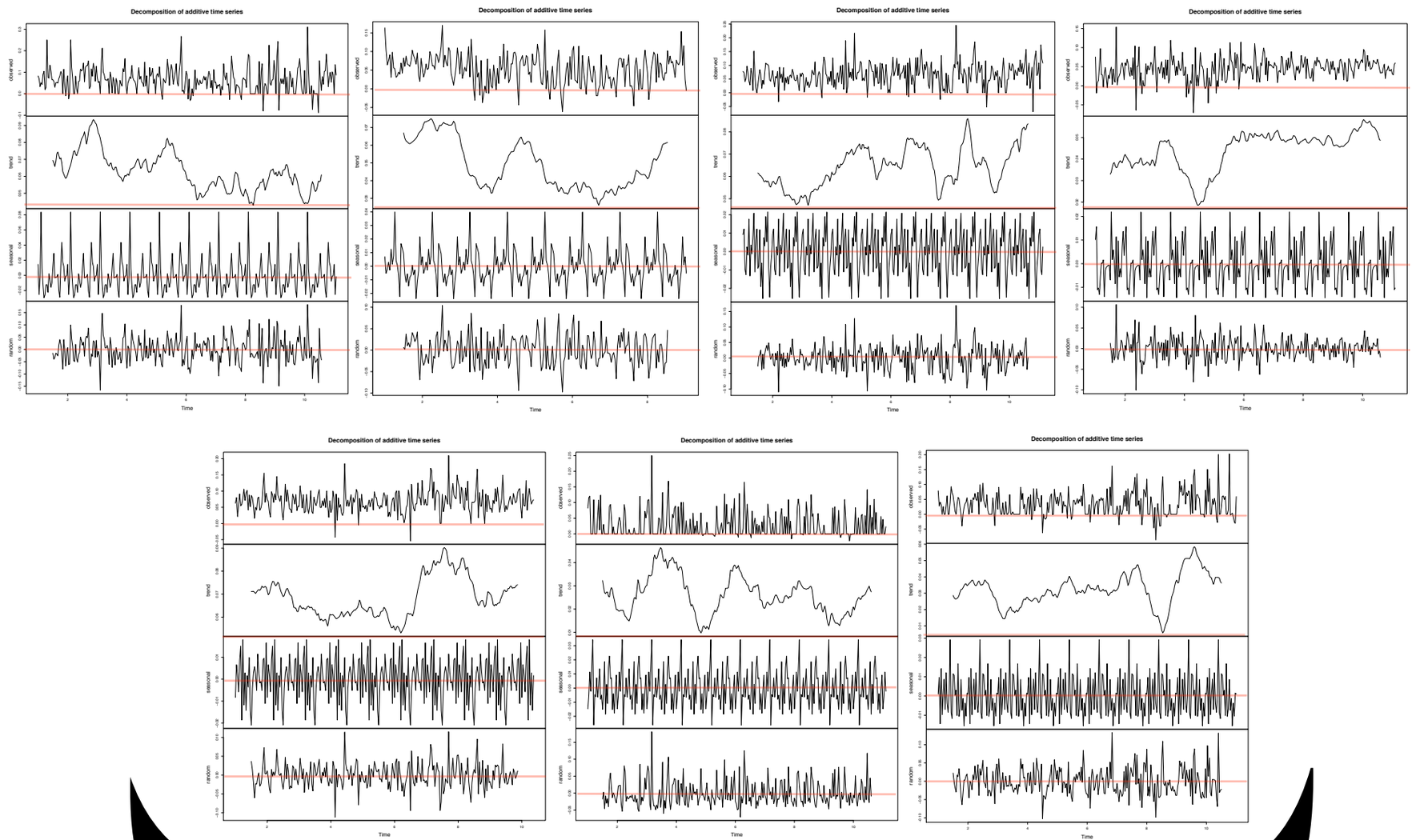


Conflict-driven

Time Series Analysis of Sentiments

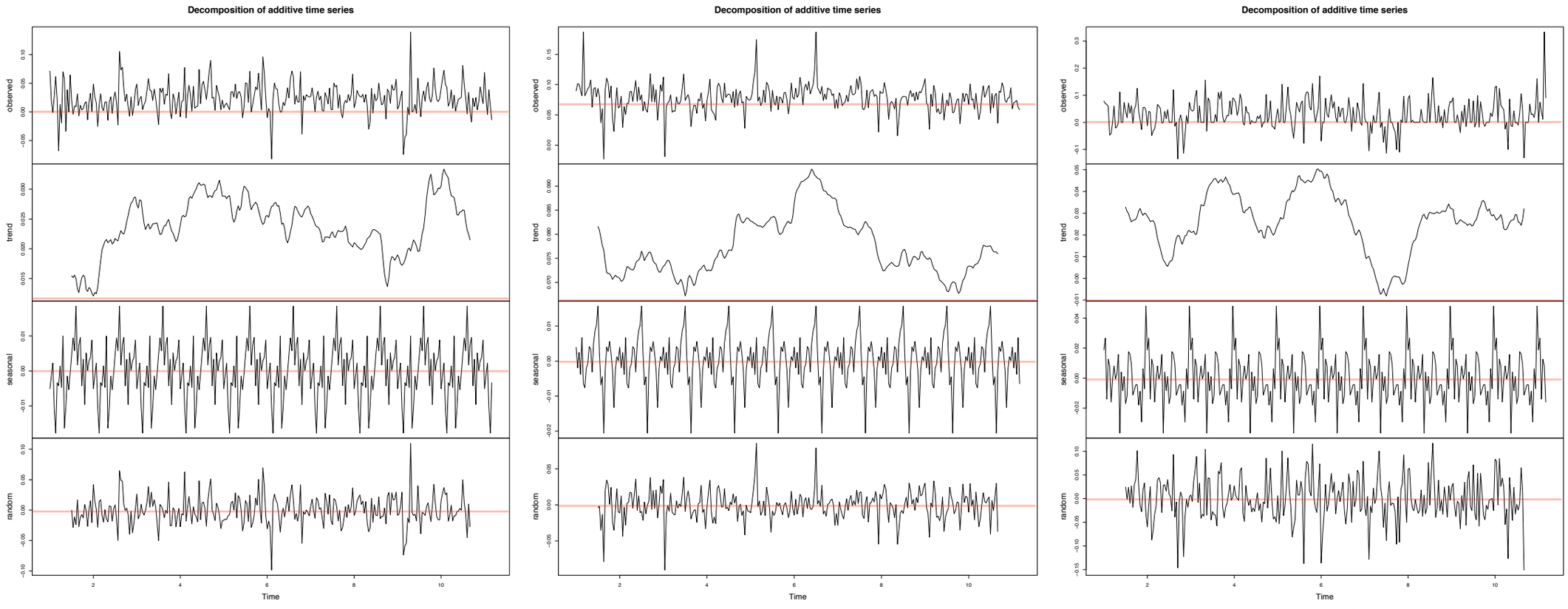


Conflict-driven



Non-conflict-driven

Time Series Analysis of Sentiments



No Fork

Time Series Analysis of Sentiments: Anomaly Detection

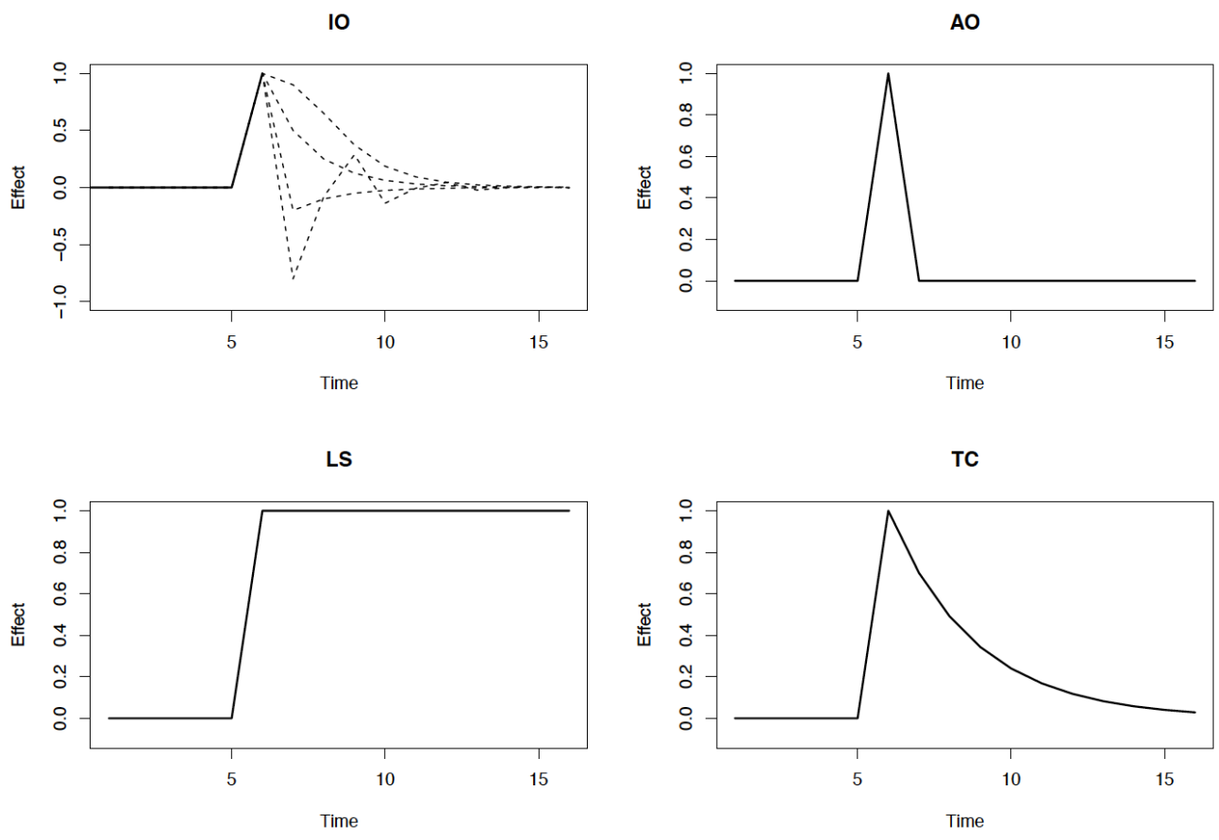
Observed series:

$$X_t^* = X_t + \text{ outlier effect}$$

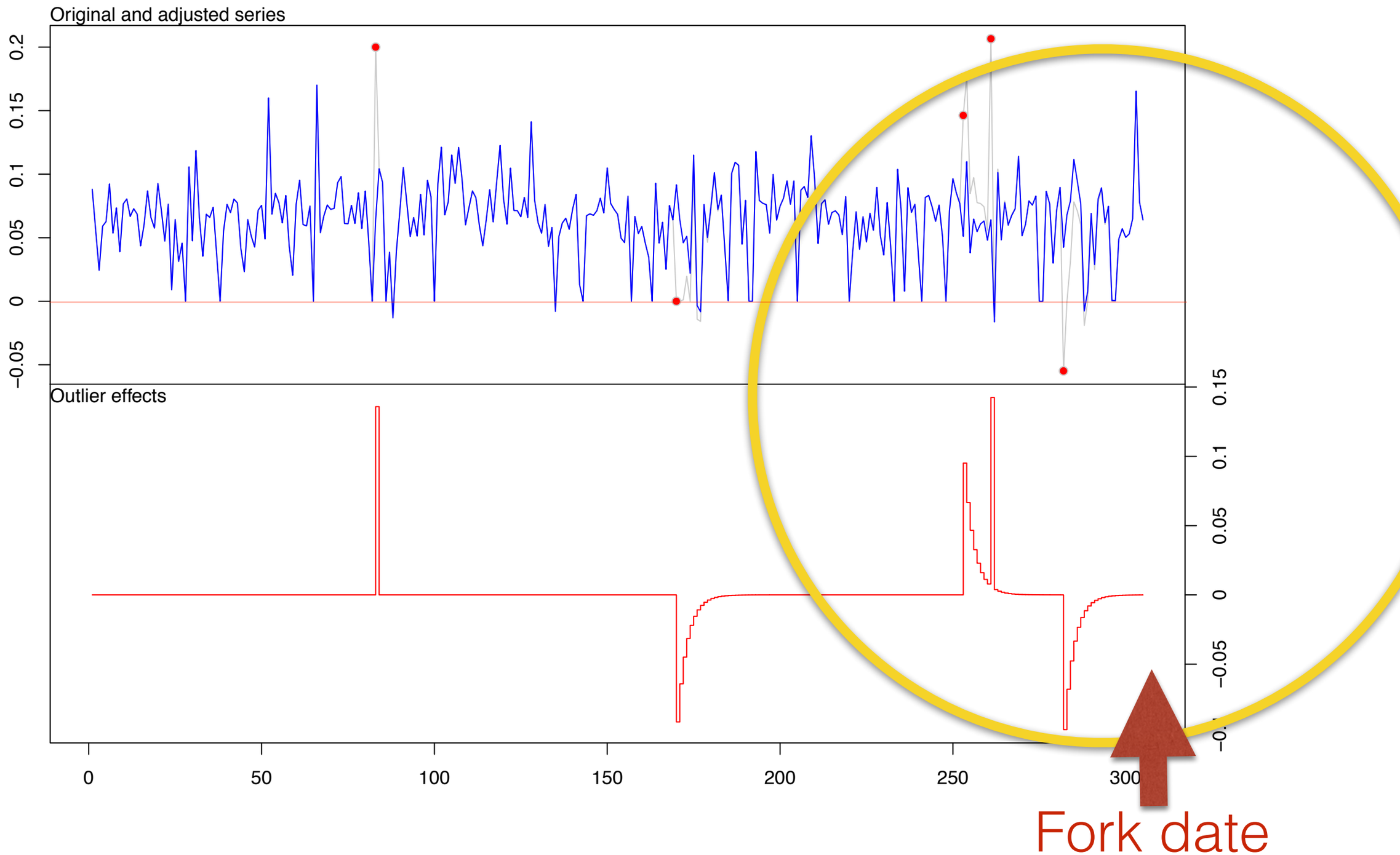
Four models for outlier effect:

- Innovational outlier (IO)
- Additive outlier (AO)
- Level shift (LS)
- Temporary change (TC)

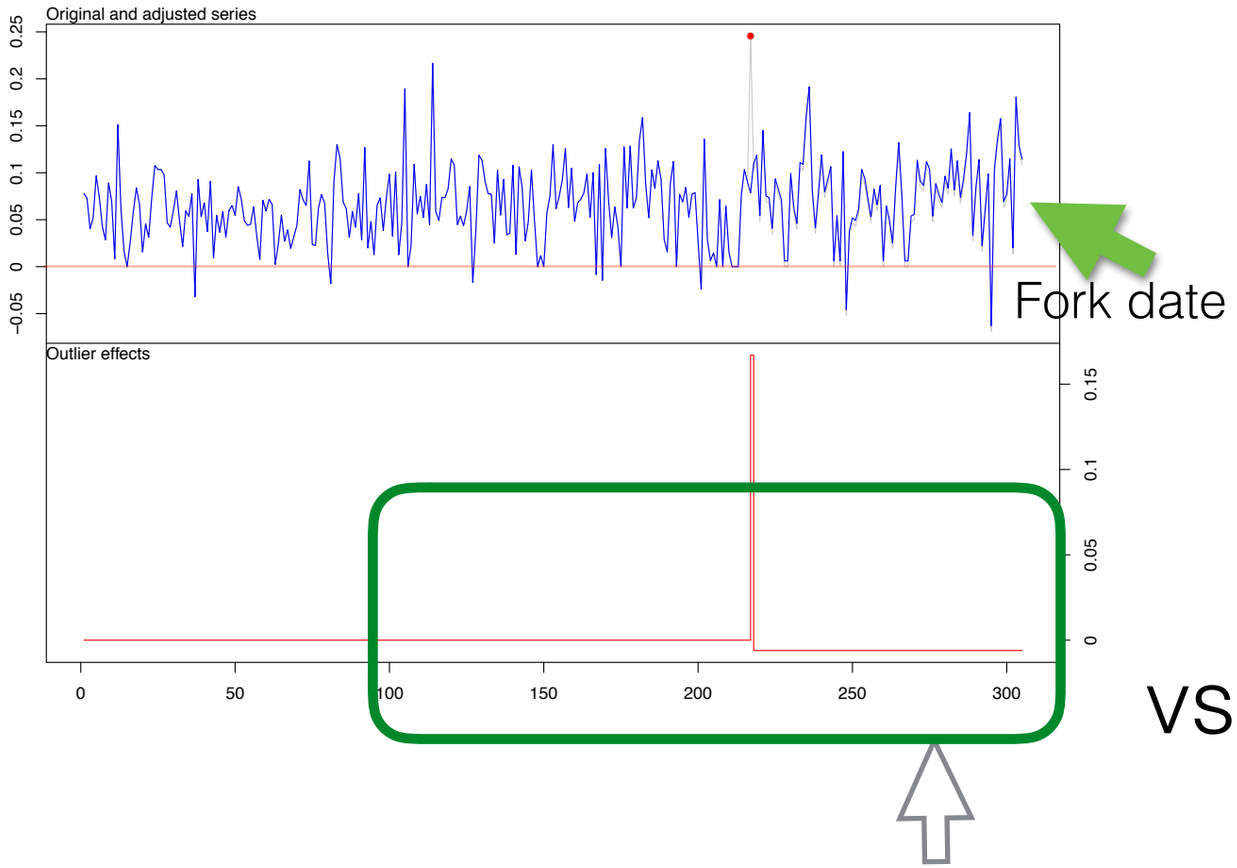
$$\begin{aligned}
\text{AO:} \quad & X_t^* = X_t + \omega I_t(t_1) \\
\text{LS:} \quad & X_t^* = X_t + \frac{1}{1-B}\omega I_t(t_1) \\
\text{TC:} \quad & X_t^* = X_t + \frac{1}{(1-\delta B)}\omega I_t(t_1) \\
\text{IO:} \quad & X_t^* = X_t + \frac{\theta(B)}{\alpha(B)\phi(B)}\omega I_t(t_1) \\
& = \frac{\theta(B)}{\alpha(B)\phi(B)}\left[Z_t + \omega I_t(t_1)\right]
\end{aligned}$$



Time Series Analysis of Sentiments: Anomaly Detection



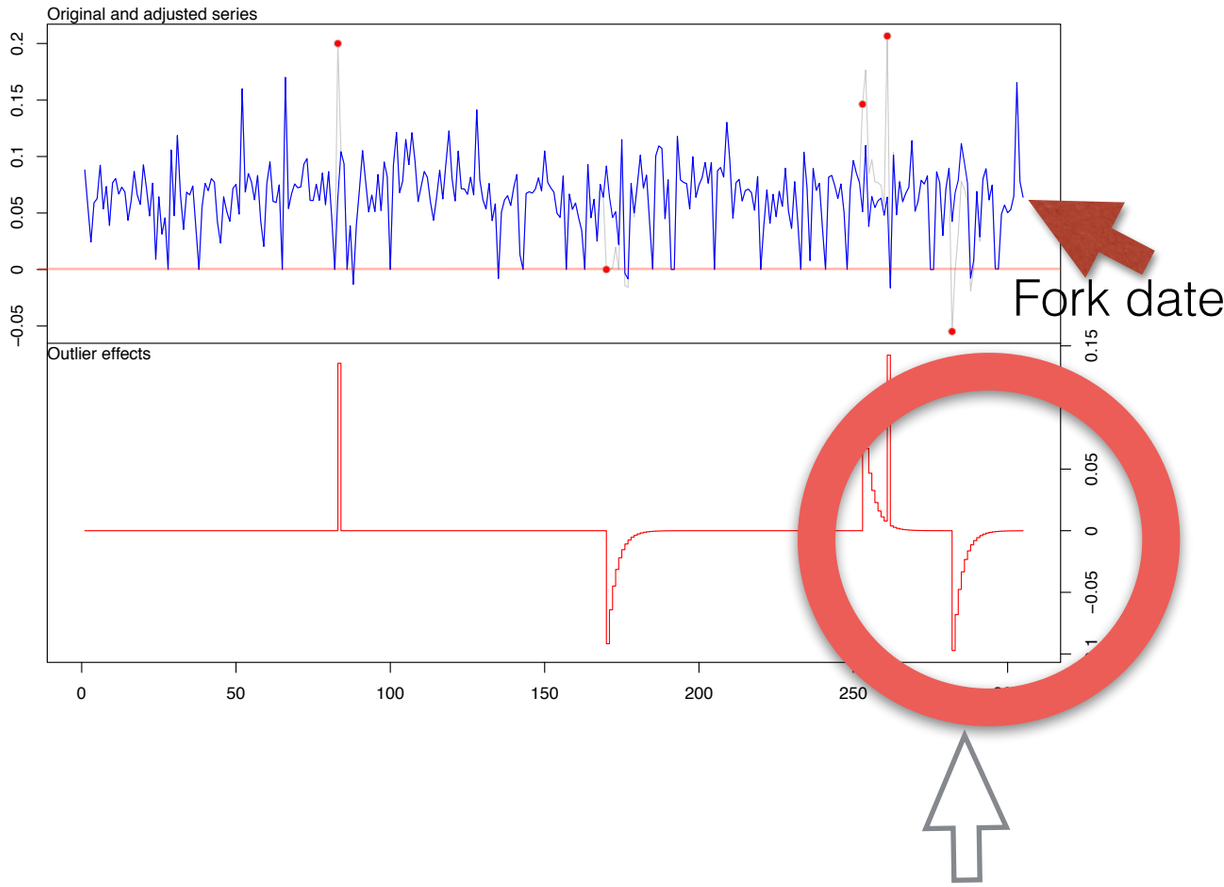
Time Series Analysis of Sentiments: Anomaly Detection



Normal, no negative outliers

Non-conflict-driven

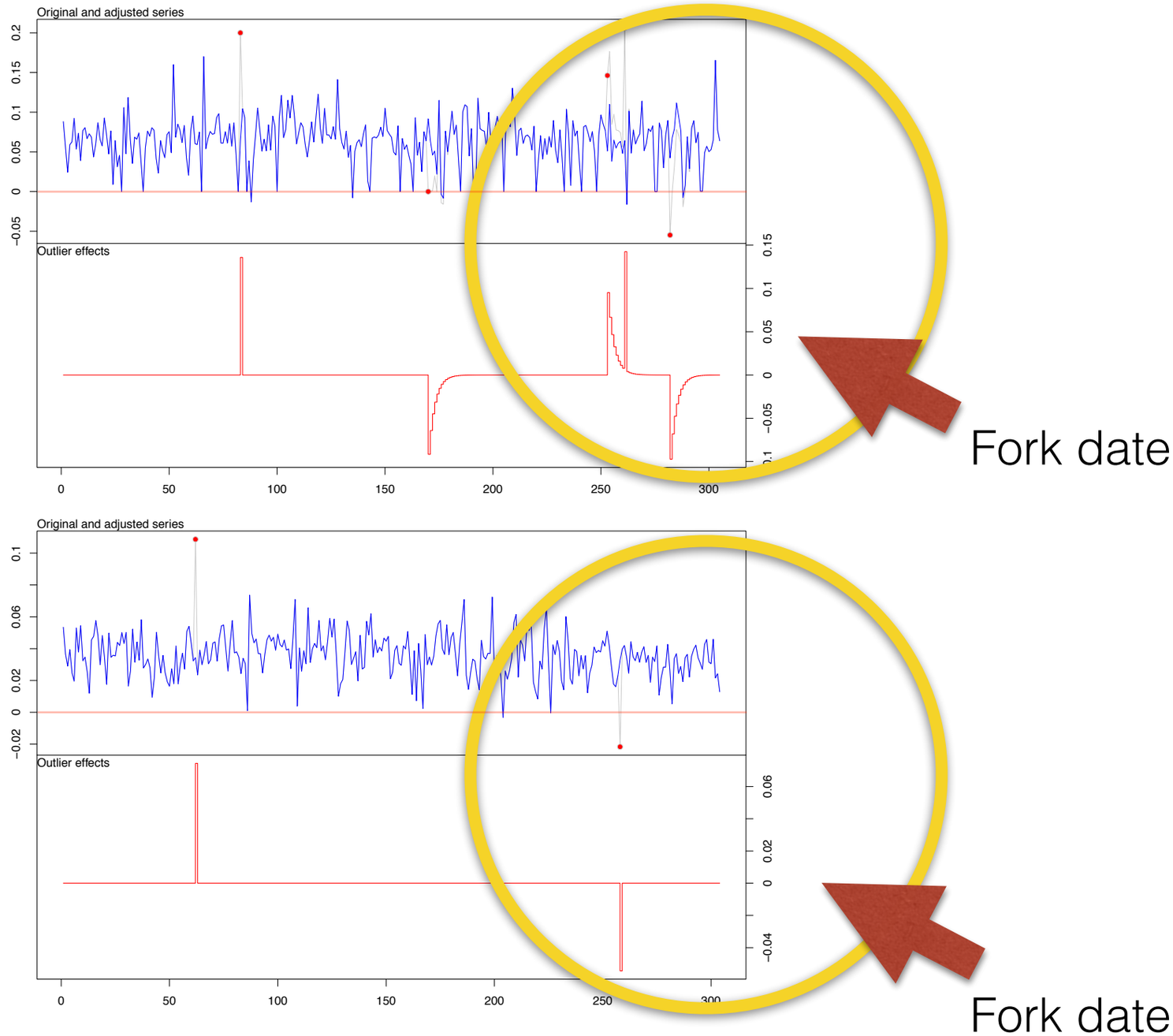
vs.



Anomaly, negative outliers

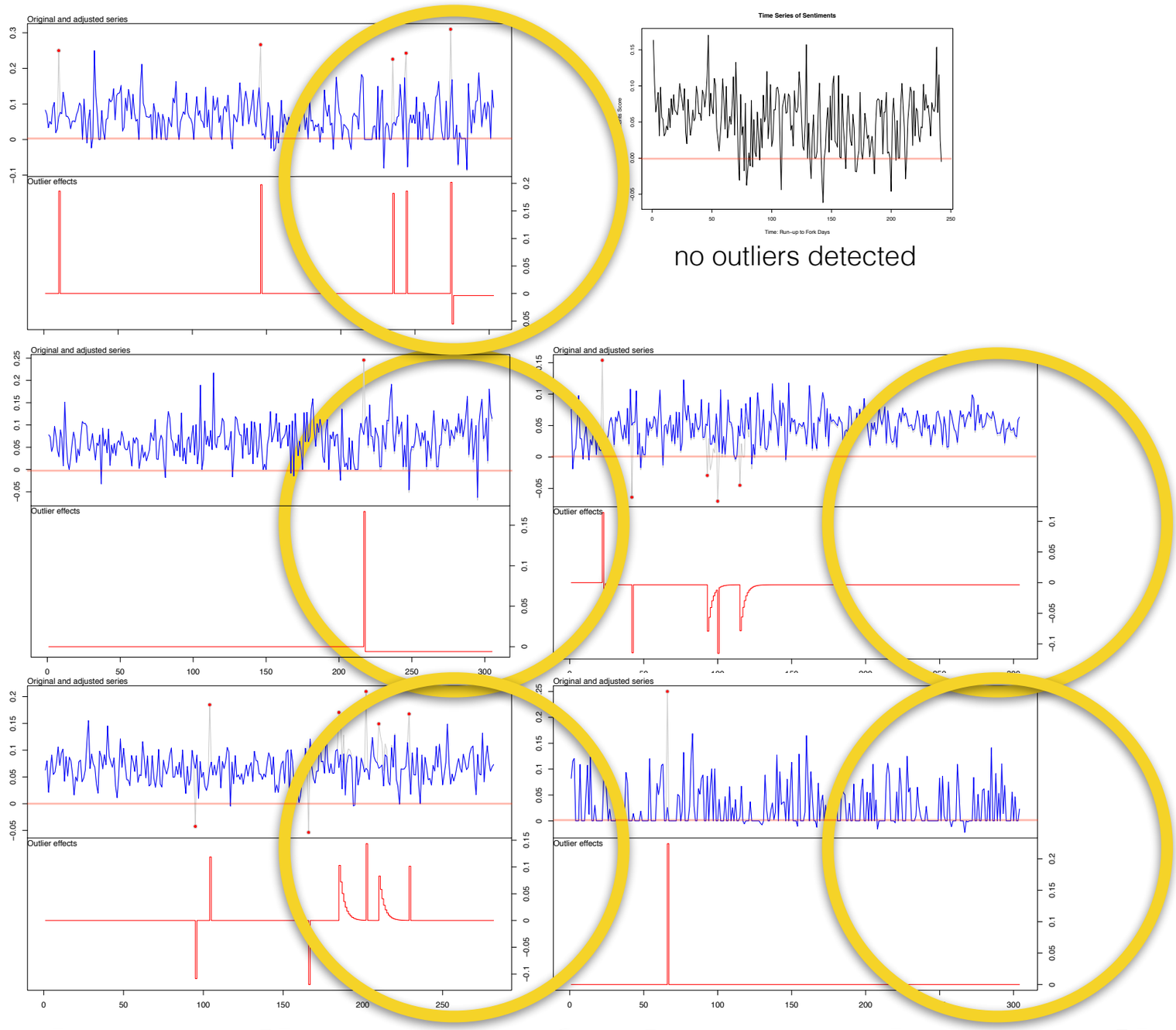
Conflict-driven

Time Series Analysis of Sentiments: Anomaly Detection: Conflict-driven



Pre-fork negative outlier anomaly

Time Series Analysis of Sentiments: Anomaly Detection: Non-conflict-driven



no outliers detected

No immediate pre-fork negative outlier

Results

Time series analysis of open source developers' communication sentiments

Here's my data:

- The contents of the messages sent and received on the projects developers mailing list, for the time period of 10-month run-up to the fork.

Here's what I found:

- Conflict-driven forks exhibited anomalies; time series analysis of sentiments shows the anomalies occur before the fork event
- Non-conflict-driven forks did not suffer from pre-fork anomalies

Here's why it matters:

- The objective was to be able to intervene, or evaluate the health.
- Anomaly detection of the time series analysis may be used by the project stakeholders/investors as key indicators left in the record, that can reliably be used to predict, or to identify problems among the developers, and intervene if need be.

Here's the implications of what I found

- It's not great, but a good way of caring about an open source project that you rely on.

Discussion

Time series analysis of open source developers' communication sentiments

- Dips don't always mean a fork happen for sure, because people are not robots, people have social intelligence, they can self-correct.
  - But, it shows that the community is going through a rough patch, and sometimes things may go completely out of control.
  - When we detect that we're heading towards a rough patch, an intervention, or a cooling-off period is probably a good idea. That's how this could be used to inform decisions.

- Same about periodicity; is it a community that went through one rough patch in 15 years, or is it a community that seems to have drama pop up every other week? If it pops up every other week, then chances are sooner or later, the dishes are gonna fly, and someone is gonna pack up their suitcase and move out.

Person A: Forked!
Person B: Really??!?!

'henning.westerholt@1und1.de'
'2008-08-04 11:09:15'
'[**Kamailio**-Devel] [Devel] SF.net SVN: opensips:[4568] tags/1.4.0/'

'Hi Dan, Hi Bogdan,

thanks for the short **announcement of the existence of this fork - opensips.org**. I was not aware of this, and i''m really disappointed. I think this is not the way to deal with this conflict. **I had appreciated to know a little bit earlier of this plans, now i understand the events of the past months on this list** and also the board somewhat better.

Henning'

Can we save troubled projects?

Person A: Forked!
Person B: Really??!?!

'jason.penton@smilecoms.com'
'2008-08-04 11:51:25'
'[Kamailio-Devel] [Devel] SF.net SVN: opensips:[4568] tags/1.4.0/'

'..... so now we have SER, Kamailio and opensips

Surely, **this is starting to confuse users and also dilute the number of developers** working to ONE common goal. **Is this really what we as developers and users want?** Is this really what the majority codebase of the product needs? **I''d be far happier having all the developers of these 3 projects working on ONE project!**

just my 0.02 worth

Cheers

Jason

On Mon, Aug 4, 2008 at 11:45 AM, Julien BLACHE <jb at jblache.org> wrote:

> Henning Westerholt <henning.westerholt at 1und1.de> wrote:

Could we know earlier?

Person A: Forked!
Person B: Really??!?!

'abalashov@evaristesys.com'
'2008-08-04 18:56:57'
'[Kamailio-Devel] [Kamailio-Users] 1.4.0 release plans,\n project matters'

'I would strongly concur with Darren here on all counts.

I don"t, of course, have any sort of inside perspective as I am not involved in development, so I can"t presume to judge the merits or veracity of the various justifications given for this adventure. Apart from Bogdan-Andrei"s brief treatment of the subject, explanations haven"t been particularly forthcoming to the community; this seems to be a back-room affair that is still playing out.

There are, certainly, times in the life of an open-source project where a code fork may be justifiable; irreconcilable differences over development methodologies, project management practices, and overall design philosophy and direction/roadmap may figure into these. Also, what Bogdan-Andrei has put forth concerning his freedom to pursue this undertaking in light of prevailing and officially codified open-source precepts is factually valid and logically ostensible.

Now, that having been said, from the perspective of a user contemplating the adoption of OpenSER in a serious industrial application, or worse, a businessperson who has sunk substantial investment into deployment of OpenSER, this is really, really bad.

Much of the value of open-source projects comes from the fact that despite the range of political characteristics potentially colouring the development and its management (from profoundly hierarchical, as in the case of the Linux kernel, to rather loose coupled and organic, as seems

to be the case with OpenSER), comes from certain rational expectations that one holds regarding their security, stability, consistency and continuity.

Industry is just as afraid of "fly-by-night" open source packages as it is of fly-by-night vendors that may not be around tomorrow.

Generally, when one chooses to invest in a rather serious and evolving open-source solution like OpenSER, one expects the following things:
1. The project will be around next year, more or less in its present state. […]

Industry/sponsors afraid of uncertainty?

Person A: Forked!
Person B: Really??!?!

'abalashov@evaristesys.com'

'2008-08-05 02:34:48'

'[Kamailio-Devel] [Kamailio-Users] 1.4.0 release plans,\n project matters'

'Bogdan,

Thank you for your reply.

You do raise a lot of good and constructive points. The overarching theme of my response to them is that because I am not privy to the interior of the project narrative, I do not have the knowledge to critically evaluate or authenticate a great deal of what you say. If the situation is as you suggest, then perhaps, indeed, you are doing a good and necessary thing. I can only give the perspective of an outsider.

That said, I offer a few inline replies, from that very vantage point:

Bogdan-Andrei Iancu wrote:

> to be honest I preferred not to put details about the arguments as I did > want people to think I''m pointing burning fingers or I''m blaming > something or somebody. Of the arguments are to be found on the > discussions from the board list, but this is not public.I can certainly understand appreciate the need to avoid public accusations or anything that could be construed as undermining anyone''s personal integrity.

The flip side to this conservative approach to publicizing internal affairs is that to the rest of us--those who are not project luminaries--it leaves the situation looking like some absurd, internecine feud replete with tribalism, factionalism, and egotism.

Of course, I think I--and the rest of us--know better than to suppose that this is actually the case, and know from passive observation of your erudition and epistolary character in the community that there are very likely real and substantive reasons for this move. However, it is not necessarily taken that way by some of our superiors and clients; these are hard-nosed, often unsophisticated people who are not going to be preoccupied with the sociological nuances and personal traits of this particular open-source ecosystem.So, managing the PR on these types of changes and finding that balance can often be the hardest challenge of all. Personally, I''d favour a little more transparency to this issue, even at the risk of some fingers and being pointed and some feelings being hurt. I''d much rather the key

Could we know earlier?

Methodology

Social network analysis of open source developers' interaction graphs

An overview of

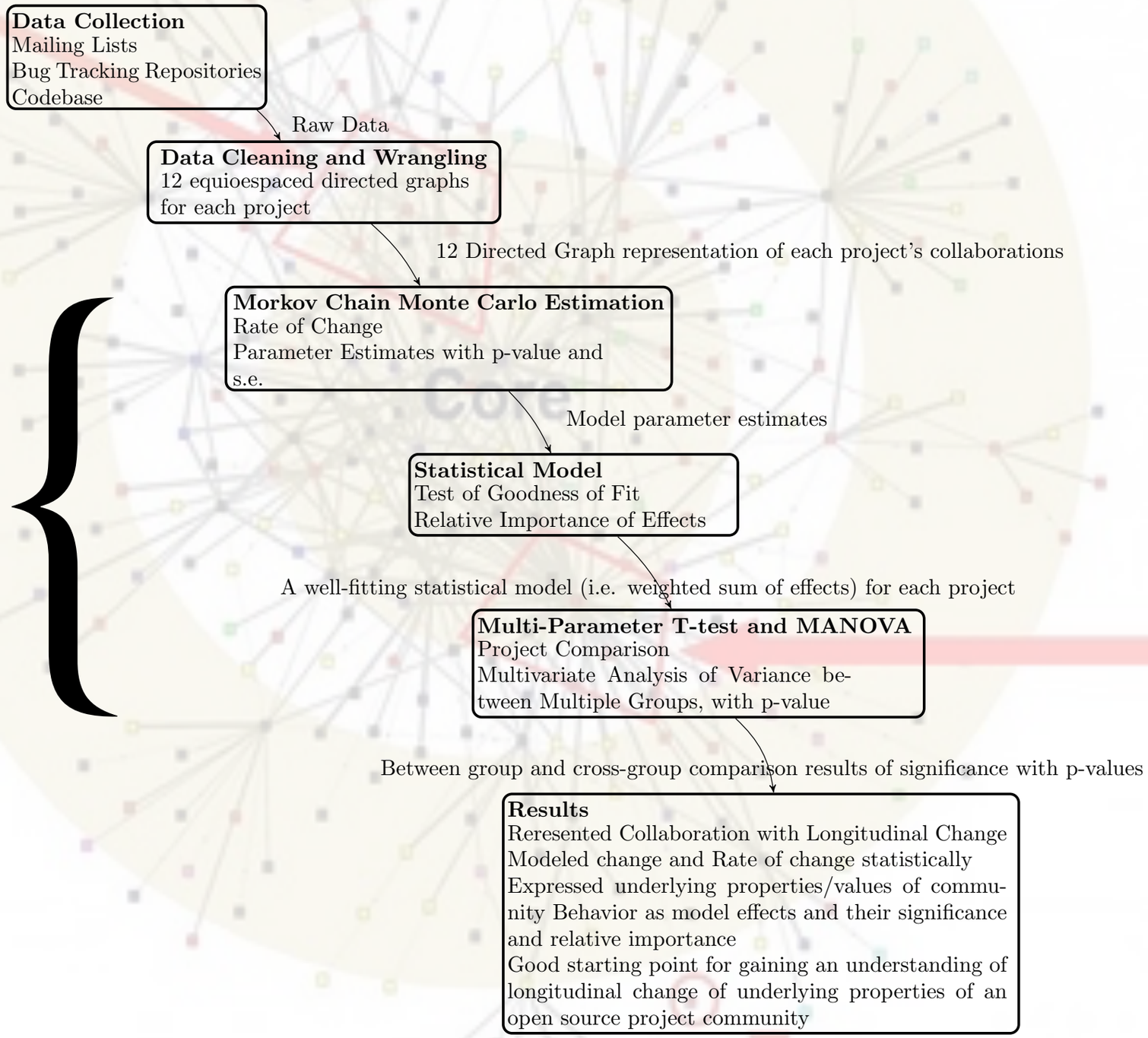
- What the mathematical models are:
  - A developer-oriented approach to statistically model the changes a community goes through in the run-up to a fork. The model represented tie formation, tie breakage, and tie maintenance between developers. It uses stochastic estimation methods to estimate several model parameters that capture the variance in the changes the community goes through.
- What they lead us to
  - Estimates of the significance of several parameters on this longitudinal change.

Communication

- Interactions can be modeled as graphs
- Graphs change through time

Methodology

Statistical Modeling



Statistical Model

- Longitudinal
- Developer-oriented:
  - Developers choose whom to talk to
  - Likelihood of forming ties, maintaining ties, …
- Stochastic
  - Developers behave to optimize an objective function; included our covariates
  - Assume observed graphs as outcomes of a continuous-time Markov process
- Estimated

Statistical Model

Longitudinal change = a series of mini-steps

To model graph evolution from $X(t_1)$ to $X(t_2)$,
and so on, we treat the dynamics as the results of
a series of small atomic changes (mini-steps)

Results

Social network analysis of open source developers' interaction graphs

Here's my data:

- The interaction graphs of the developers that sent and received messages on the projects developers mailing list, for the time period of 10-month run-up to the fork event. And the source code contribution levels of the developers

Here's what I found:

- In conflict-driven forks, (1) the developers maintained a preference for interacting with developers who had similar out-degrees, in contrast to the non-conflict-driven forks, where the developers did not require similar out-degrees. (2) In conflict-driven forks, the interactions were reciprocal, in contrast to the non-conflict-driven forks, where the interactions did not need to be reciprocal to happen. (3) In conflict-driven forks, the more senior developers preferred to interact with other more senior developers, in contrast to the non-conflict-driven forks.
- In non-conflict-driven forks, (1) the interactions did not necessarily need to be reciprocal, in contrast to the conflict-driven forks, where the interactions needed to be reciprocal to continue to happen. (2) In non-conflict-driven forks, the developers with high source code contribution levels interacted more with other high source code contributors. (3) In non-conflict-driven forks, high levels of contribution to the source code brings you connections more rapidly, while high levels of contributions to the mailing list is not suggestive of this. This can be interpreted as a sign of meritocracy based on code, rather than talk, which captures a healthy dynamic in these project.

Here's why it matters:

- Estimating the influence of several factors on how developers interact provides indicators of healthy dynamics, such as meritocracy in the project.

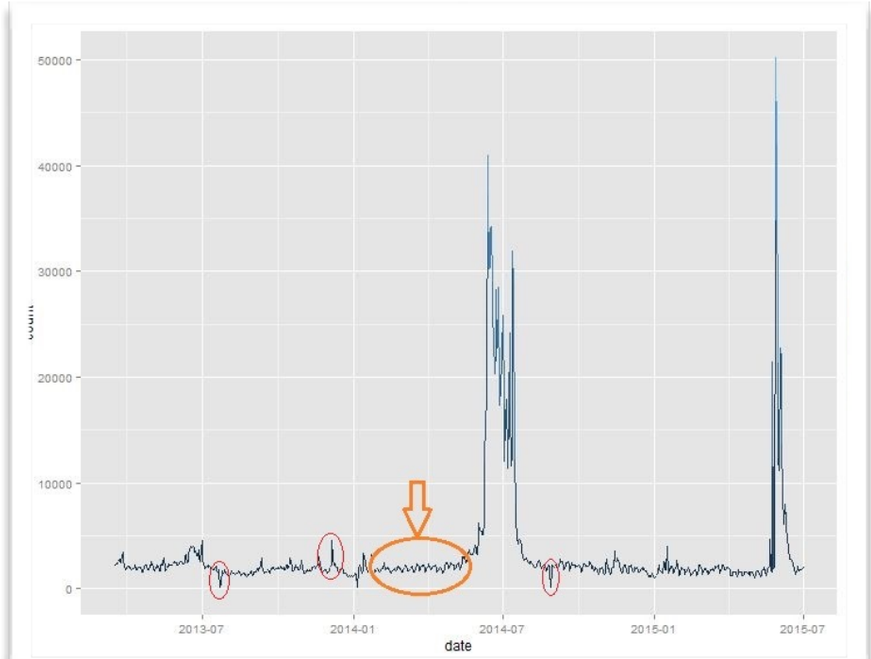
Here's the implications of what I found:

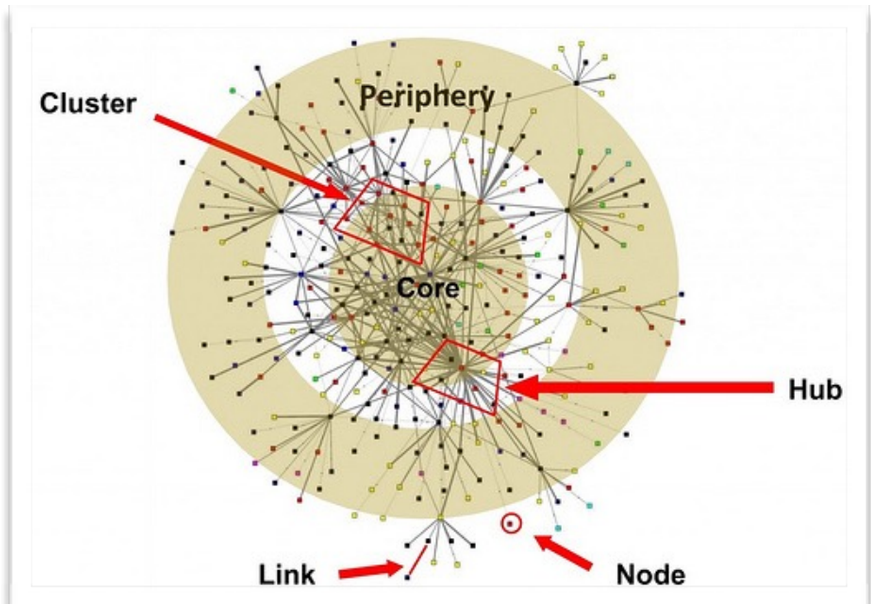
- It's not a great way of measuring up a project, but it's not a bad way of finding out who in the project deserves credit, who is mostly talk and no action, and whether the project is a meritocracy.

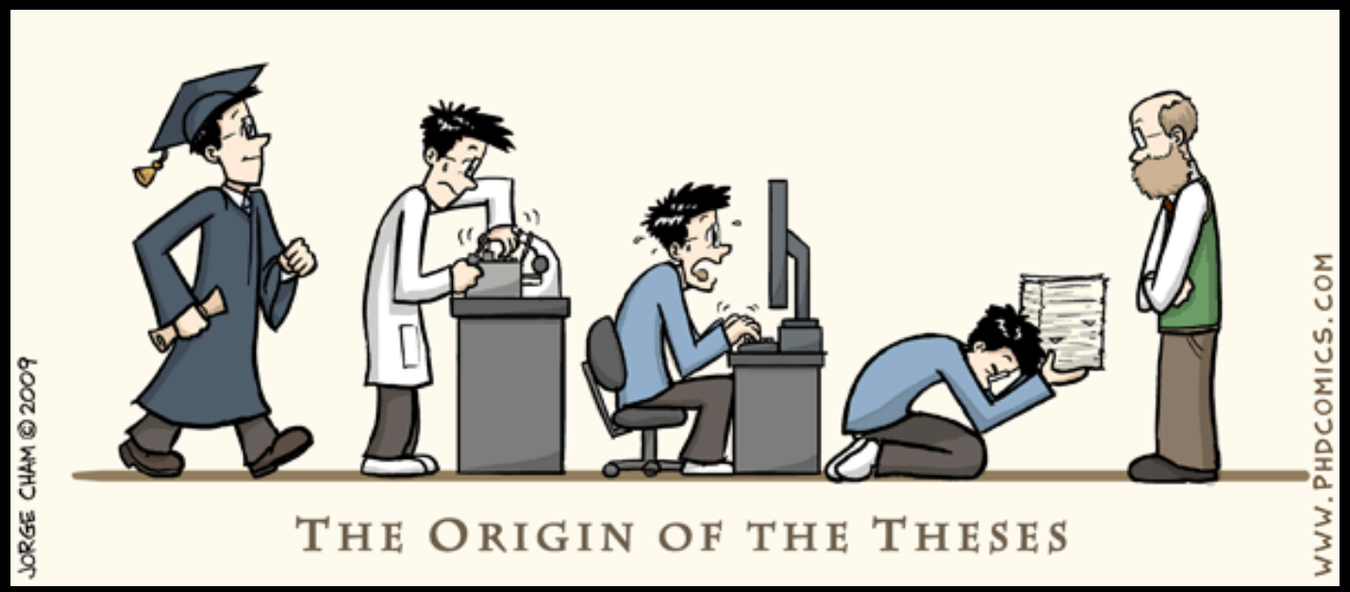
Summary

- Represented longitudinal dynamics in conflict-driven vs. non-conflict-driven forks
- Suggested indicators of unhealthy dynamics to suggest intervention
- Detected pre-fork anomalies suggestive of negative communication sentiments
- Expressed underlying properties/values of community behavior as model effects and their significance
- Good starting point for gaining an understanding of longitudinal change of underlying properties of an open source project community. (e.g. Github plug-in indicator)

Future Work


50000
40000
30000
20000
10000
0
2013-07
2014-01
2014-07
2015-01
2015-07
date


Cluster
Periphery
Core
Hub
Link
Node


JORGE CHAM © 2009
WWW.PHDCOMICS.COM
THE ORIGIN OF THE THESES

Developer Interactions

Occurs through mailing lists, IRC lines, comments, bug tracking system: This is where people talk, this is where communities form. Some of these are more important than others, for our analysis.

- We primarily focus on mailing lists; Mailing List is the official record; this is where the important stuff gets discussed. This is the biggest public forum. You could have minor blow-ups in other parts, and that's okay, if it's important, it will migrate to the mailing list. Will be captured in the mailing list.

- IRC: Not everyone participates, there's no record, it's ephemeral
- Git logs: not everyone reads them, they are typically focused on what the code is at that point
- Bug tracking system: not everyone reads them, they are typically focused on what the code is at that point

Economics that Drive Open Source

- A company often needs a particular piece of software to be maintained and developed, and yet does not need a monopoly on that software.
  - Example: everyone needs a web server software, but no one needs to own. Why not share the burden of maintenance?

Business Models

- Support-ware
  - Software is free, support is not
- Product-ware
  - Software is free, the device it runs on is not
- Cloud-ware
  - Software is free, but only runs on cloud. Pay us for use/resources
- Project-ware
  - Base software is free, pay us to customize it
- Hybrid-ware
  - Proprietary add-ons or premium features
- Dual License
  - Commercial us vs. non-profit & education uses
- Consortium-ware
  - Everyone commits resources for common good.
    Services (per use)
- Ad-Ware
  - All hail Google the great and powerful
- Donation-ware
  - Click to contribute

Governance Structures

- Benevolent Dictators
  - Final decision-making authority rests with one person, who, by virtue of personality and experience, is expected to use it wisely. Reluctance to make decisions by fiat is a trait shared by almost all successful benevolent dictators; and one of the reasons they manage to keep the role.

- Consensus-based Democracy
  - As projects get older, they move away from benevolent dictatorship, toward group-based governance.
  - Consensus simply means an agreement that everyone is willing to live with. A group has reached consensus on a given question when someone proposes that consensus has been reached and no one contradicts the assertion.

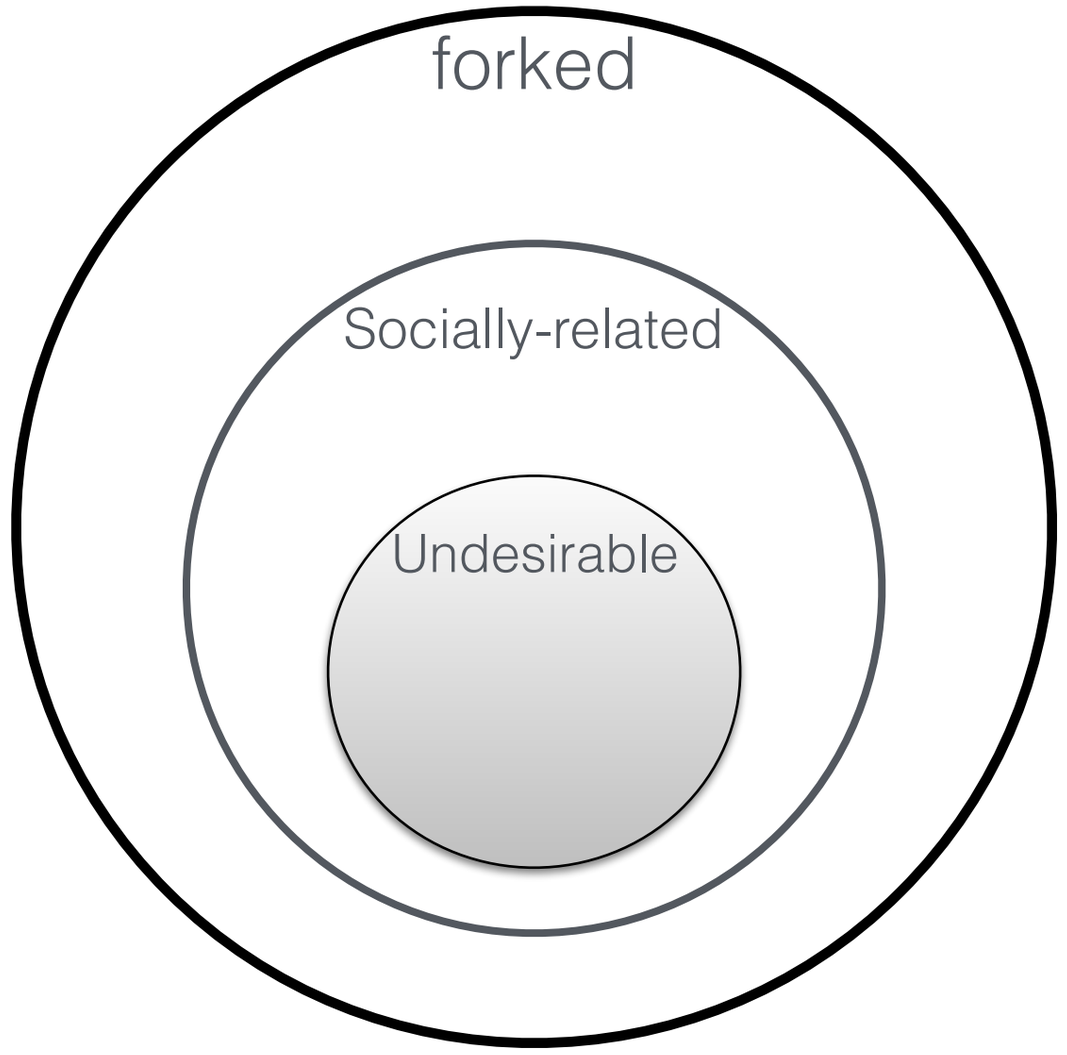
Why [some] Corporations Support Open Source?

- Sharing the burden
- Ensuring maintenance of infrastructure
  - when a company sells services which depend on FOSS
- Establishing a standard
  - releasing an open source implementation of that standard
- Creating an ecosystem
  - in which they are more likely to flourish
- Supporting hardware sales
  - having high-quality free software to run on hardware is important to customers
- Undermining a competitor
  - Eating away at a competitor's market share
- Marketing
- Proprietary relicensing

Undesirable forks = ?

- **Undesirable forks:** Perceived as bad forks, that affects the developer community and users negatively, and would've been avoided if possible.

- **Socially-related forks**: Could have left traces in the developers' interactions data.

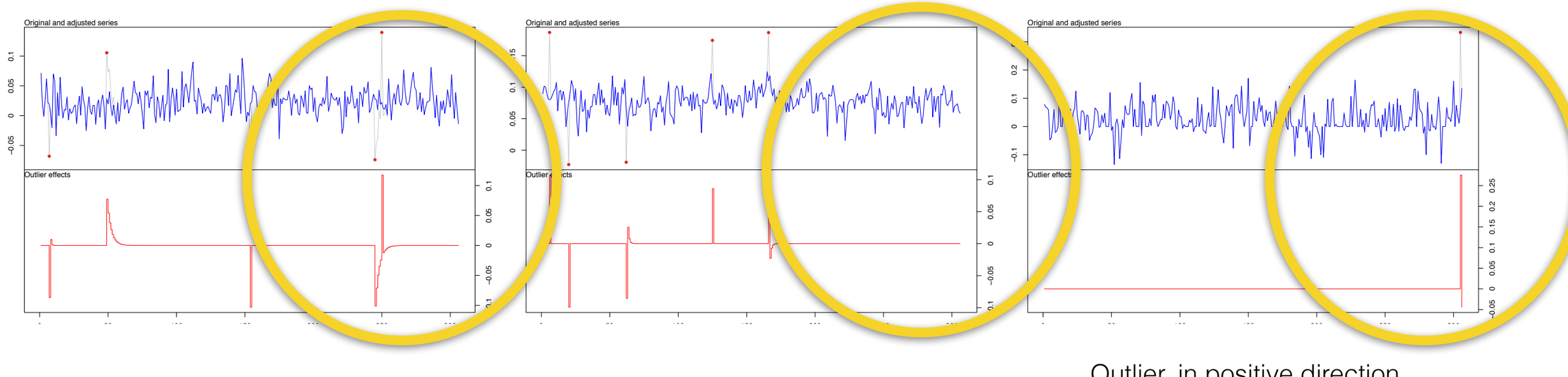


Why Projects Fork?

Socially-related { H.F. / U.F. }

Reason for forking	Example forks
Technical (Addition of functionality)	Amarok & Clementine Player
More community-driven development	Asterisk & Callweaver
Differences among developer team	Kamailio & OpenSIPS
Discontinuation of the original project	Apache web server
Commercial strategy forks	LibreOffice & OpenOffice.org
Experimental	GCC & EGCS
Legal issues	X.Org & XFree

Time Series Analysis of Sentiments: Anomaly Detection: No.F.



Outlier, in positive direction,
in contrast to negative outlier in conflict-driven.

No fork

Statistical Model
Mathematical notation

- Data: M repeated observations on a graph with g developers

- Representation: Directed graphs with adjacency matrices $X(t_m) = (X_{ij}(t_m))$ for $m = 1, \ldots, M$
  - where i and j range from $1, ..., g$

- X_{ij} shows whether at time t, there exists a tie from i to j (value 1) or not (value 0).

Statistical Model
Mini-steps

- Developer whose tie is changing, has control over the change

- The graph changes one tie at a time (**1 mini-step**)

- The moment of change & the kind of change **depends on observed covariates** and the **graph structure**

- The **moment of change** is stochastically determined by the *rate function*

- The particular **kind of change** is determined by the objective function

Statistical Model
Rate Function

The **rate function** $\lambda_i(x)$ for developer *i* is the rate at which developer *i*'s outgoing connection changes occur.

$$\lambda_i(x) = \lim_{dt \to 0} \frac{1}{dt} \mathrm{P}\big(X_{ij}(t+dt) \neq X_{ij}(t) \quad for \quad some \quad j \in \{i, ..., g\} | X(t) = x\big)$$

It models *how frequently* the developers make mini-steps.

Statistical Model

Objective function

The **objective function** $f_i(s)$ for developer i is the value attached to the network configuration x.

$$f_i(\beta, x) = \sum_{k=1}^{L} \beta_k s_{ik}(x)$$

- Functions $s_{ik}(x)$ are the effects we define. Parameters $\beta = (\beta_1, ..., \beta_L)$ is to be estimated.

- Idea: Given the opportunity to make a change in his out-going tie variables $(X_{i1},...,X_{ig})$, developer i selects the change that gives the greatest increase in the objective function.

Statistical Model

Simulation and Estimation

- Using Method of Moments (MoM)
  - Equate the observed sample statistics to the theoretical population statistics and solve the equation to find the coefficients
- Markov Chain Monte Carlo (MCMC) estimation
  - Simulate the network evolution, and estimate the model based on the simulations

We fitted/have a model