Longitudinal Analysis of Collaboration in 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
- Operational health
- Survivability
  - Bus #

Open source is great
but not all the time

- Not everything works smoothly all the time in open source projects
- **Problems**:
  - Uncertainty
  - “Industry afraid of "fly-by-night" open source packages & vendors

What is this about?

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

- What triggers forking?
- Can we do something to predict when the community started to disintegrate?

Big Picture

- Developing complex software systems is complex
- Software developers interact
  - May 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

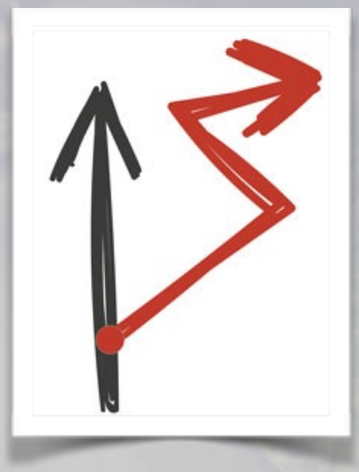
} people suffer

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

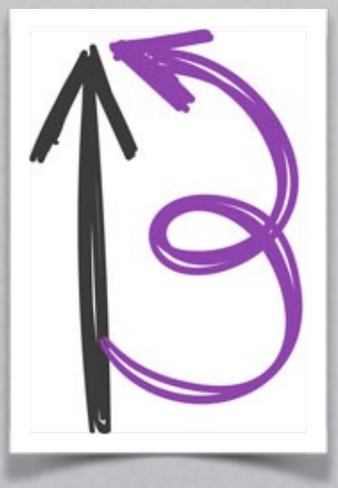
Forking Categories

Communities dissolve or evolve for a variety of different reasons:

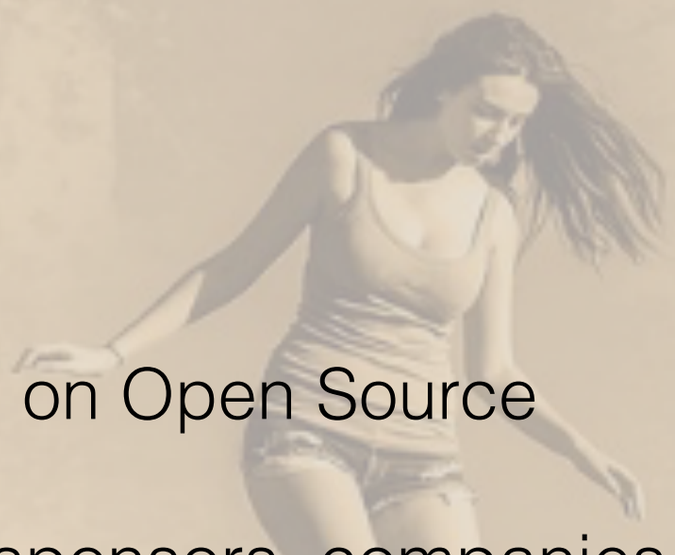
- **Conflict-driven**
  - Destructive
  - Lead to resentment
  - Become competitors; “you’re either with us, or against us”



- **Non-conflict-driven**
  - Developers interested in exploring different functionalities
  - 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?



- 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

“[…] this seems to be a back-room affair that is still playing out. […] 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.”

“[…] 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!”

Social Health is Important

- Is project likely to fork?
- When?
- Can we intervene?
- Objectively view whether this is a community you want to get involved and invest in.

Related Work

- Identifying knowledge brokers [47]
- Static Communication patterns [24]
- Sustainability [31]
- Visual exploration of collaboration [52]
- Post-forking porting of new features or bug fixes [7]

Related Work Limitations

- Previous research limited to one or two snapshots
- How dynamics change over time?
- What happens before forking?
- Could we have known earlier?

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

Research Goals

- Can we identify if there is a problem?
- Can we do this early enough that we can actually do something about it?
- Can we do this using objectively automate-able processes without having to be intimately familiar with the projects?

Data Collection

The first challenge you face:

- Finding projects that have gone through a various types of forking
- Projects survived
- Project records survived

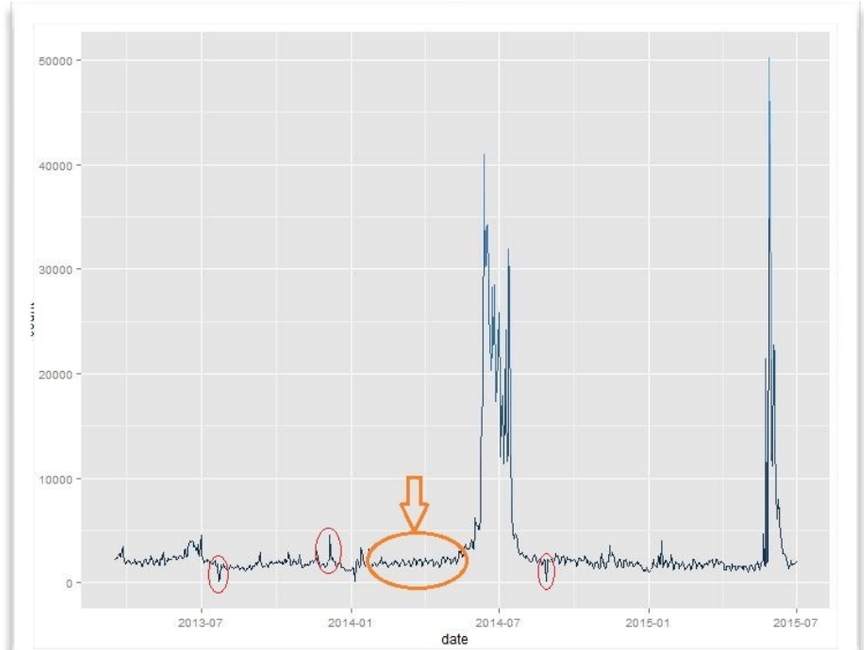
Data Collection Projects

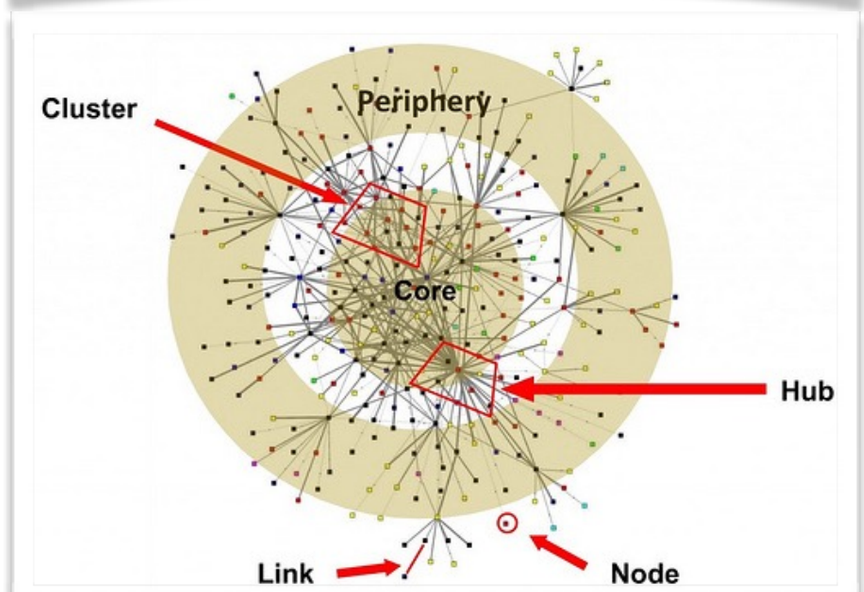
	Projects	Reason for forking	Year forked
Conflict-driven	Kamailio & OpenSIPS	Differences among developer team	2008
	ffmpeg & libav	Differences among developer team	2011
Non-conflict-driven Community-driven	Asterisk & Callweaver	More community-driven development	2007
	rdesktop & FreeRDP	More community-driven development	2010
	freeglut & OpenGLUT	More community-driven development	2004
Non-conflict-driven Technical	Amarok & Clementine Player	Technical (Addition of functionality)	2010
	Apache CouchDB & BigCouch	Technical (Addition of functionality)	2010
	Pidgin & Carrier	Technical (Addition of functionality)	2008
	MPlayer & MPlayerXP	Technical (Addition of functionality)	2005
Not forked	Ceph	Not forked	-
	Python	Not forked	-
	OpenStack Neutron	Not forked	-
	GlusterFS	Not forked	-

Methodology

Two analyses:

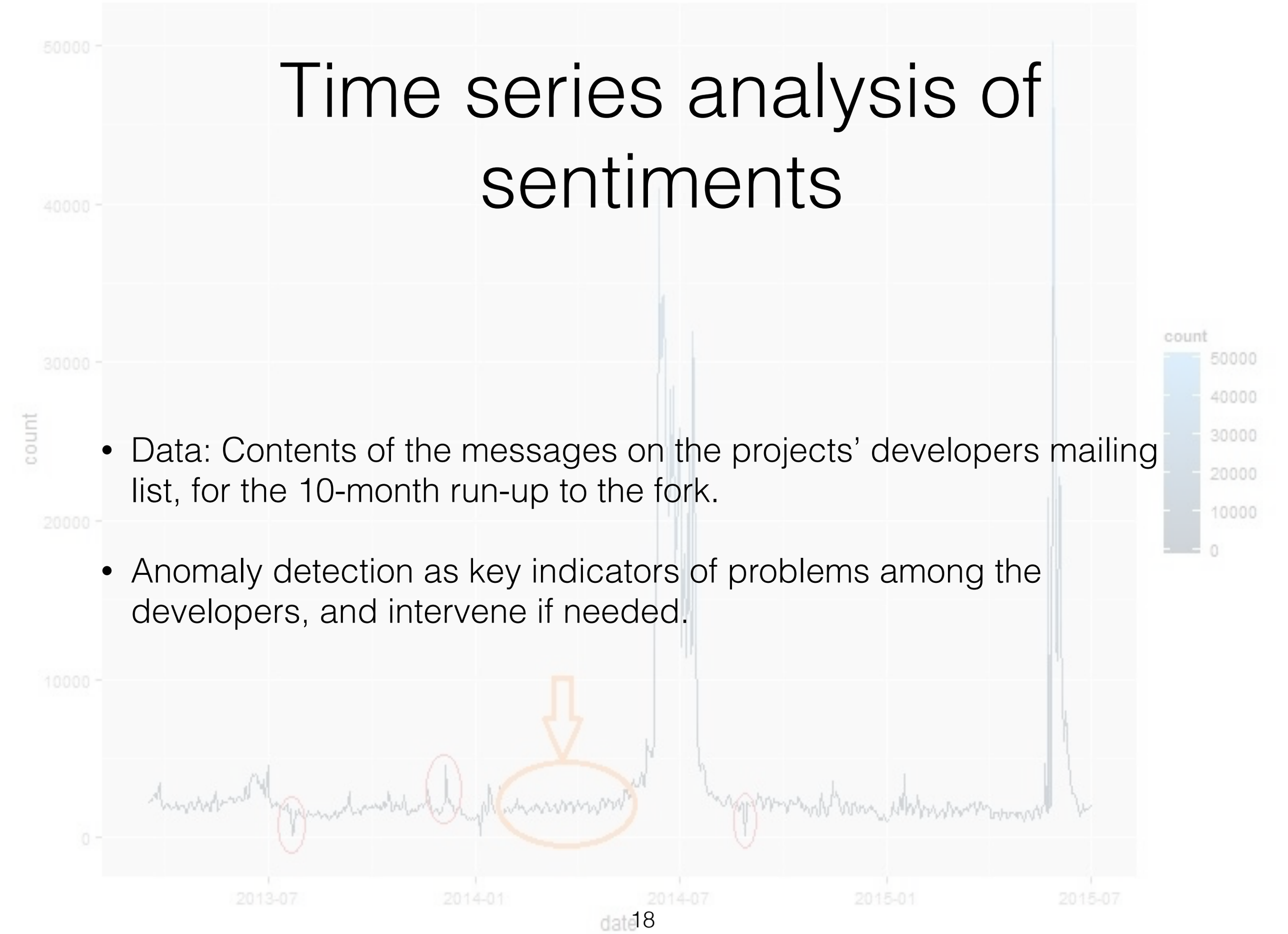
- Time series analysis of developers' communication sentiments
- Social network analysis of developers' interaction graphs





Time series analysis of sentiments

- Data: Contents of the messages on the projects' developers mailing list, for the 10-month run-up to the fork.
- Anomaly detection as key indicators of problems among the developers, and intervene if needed.



Sentiment Analysis

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

Used for

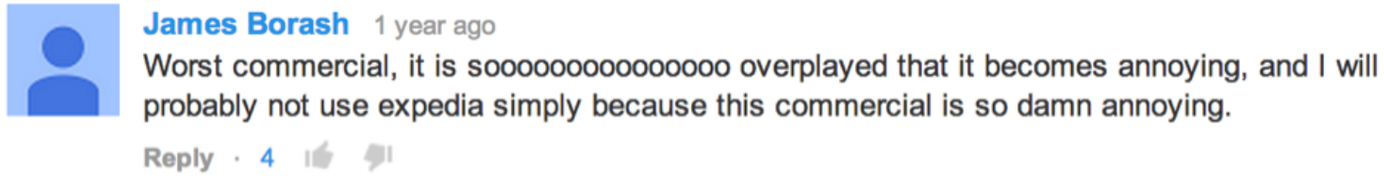
- “Flame” detection
- Bias identification in news sources
- Identifying (in)appropriate content for ad placement
- Targeting advertising/messages, gauging reactions, etc.

Sentiment Analysis

Examples

- The Obama administration used it to gauge public opinion to policy announcements and campaign messages ahead of 2012 election.

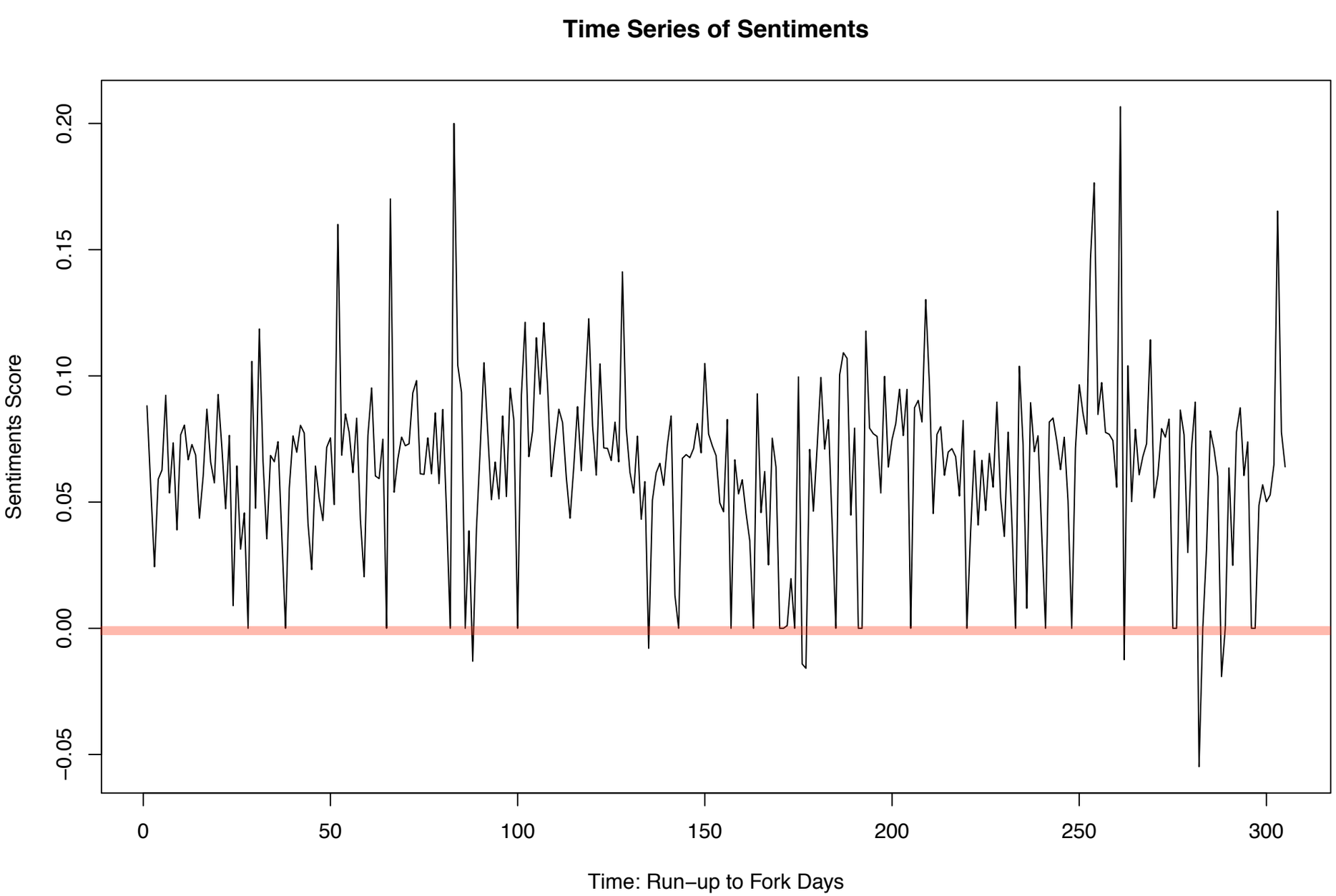
- Businesses like Expedia used it to understand consumer attitudes and react accordingly.



Sentiment Analysis

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

Time Series Analysis of Sentiments



Sentiment Analysis:
Time Series Analysis

Used to

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

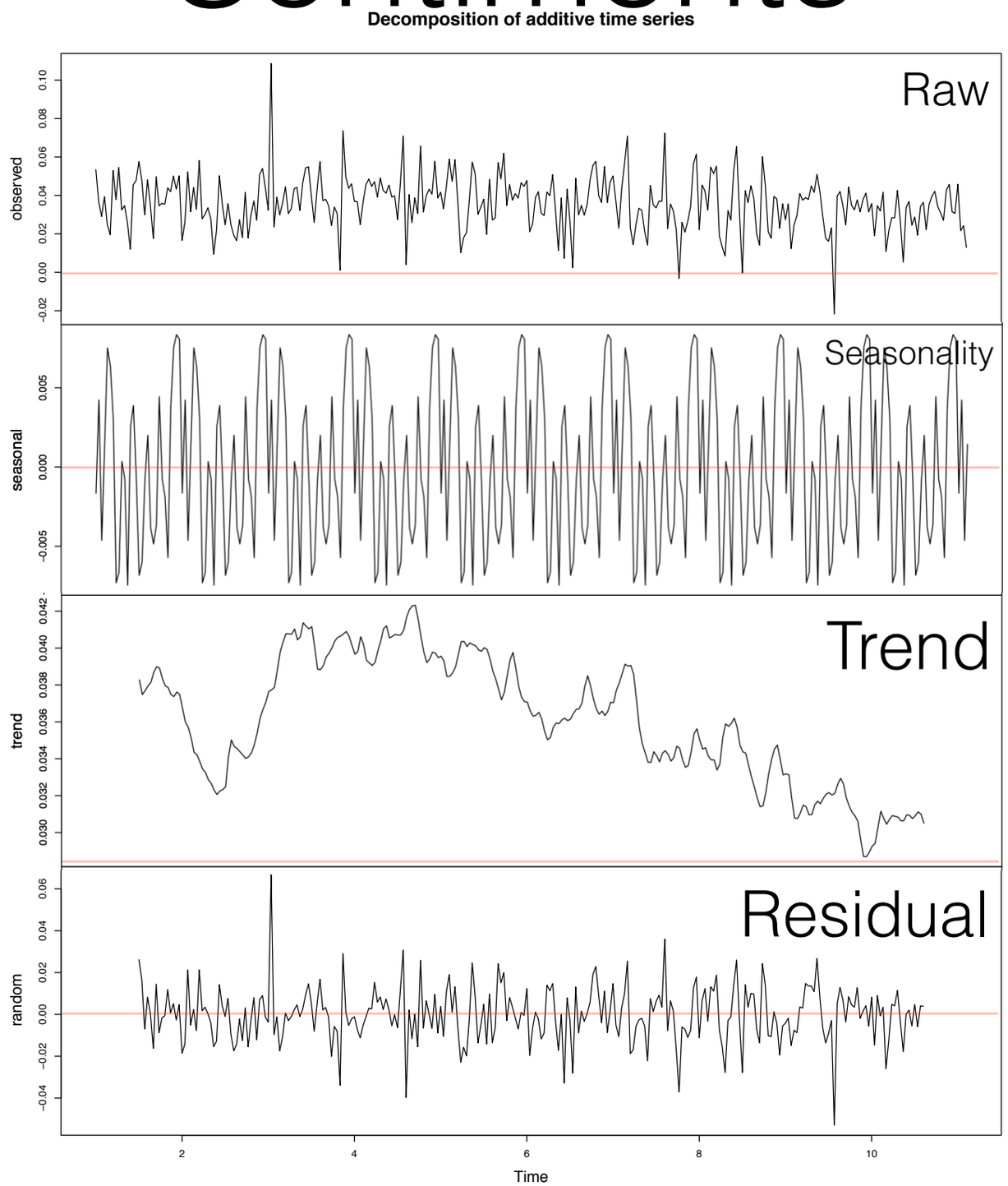
- **Forecasting** (predicting inference, a subset of statistical inference). assumes that present trends continue. This assumption cannot be checked empirically, but, when we identify the likely causes for a trend, we can justify the forecasting(extrapolating it) for a few time-steps at least

- **Anomaly detection**

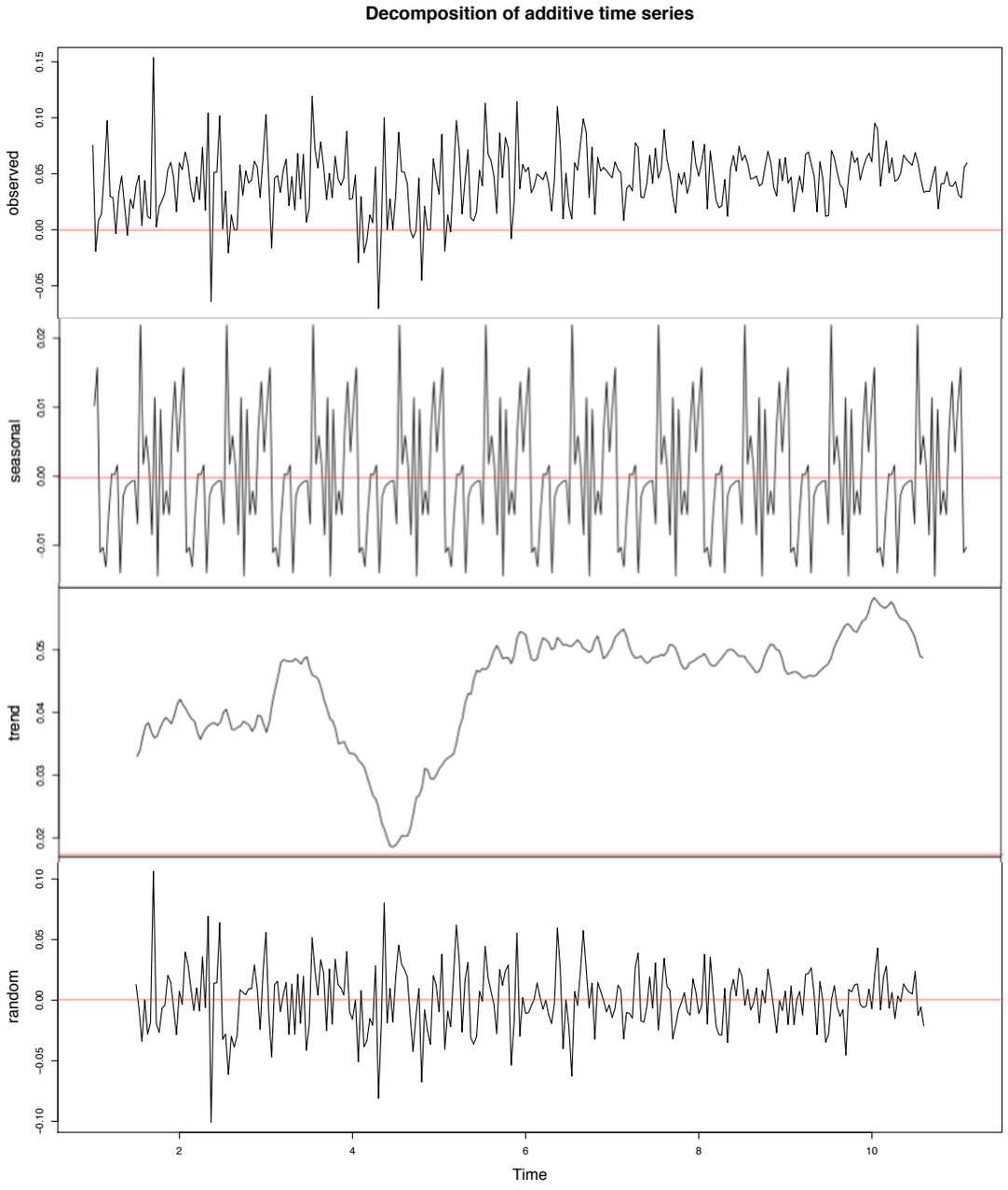
- Classification (assigning a time series pattern to a specific category: e.g. gesture recognition of hand movements in sign language videos)

- Query by content ~ content-based image retrieval

Time Series Analysis of Sentiments

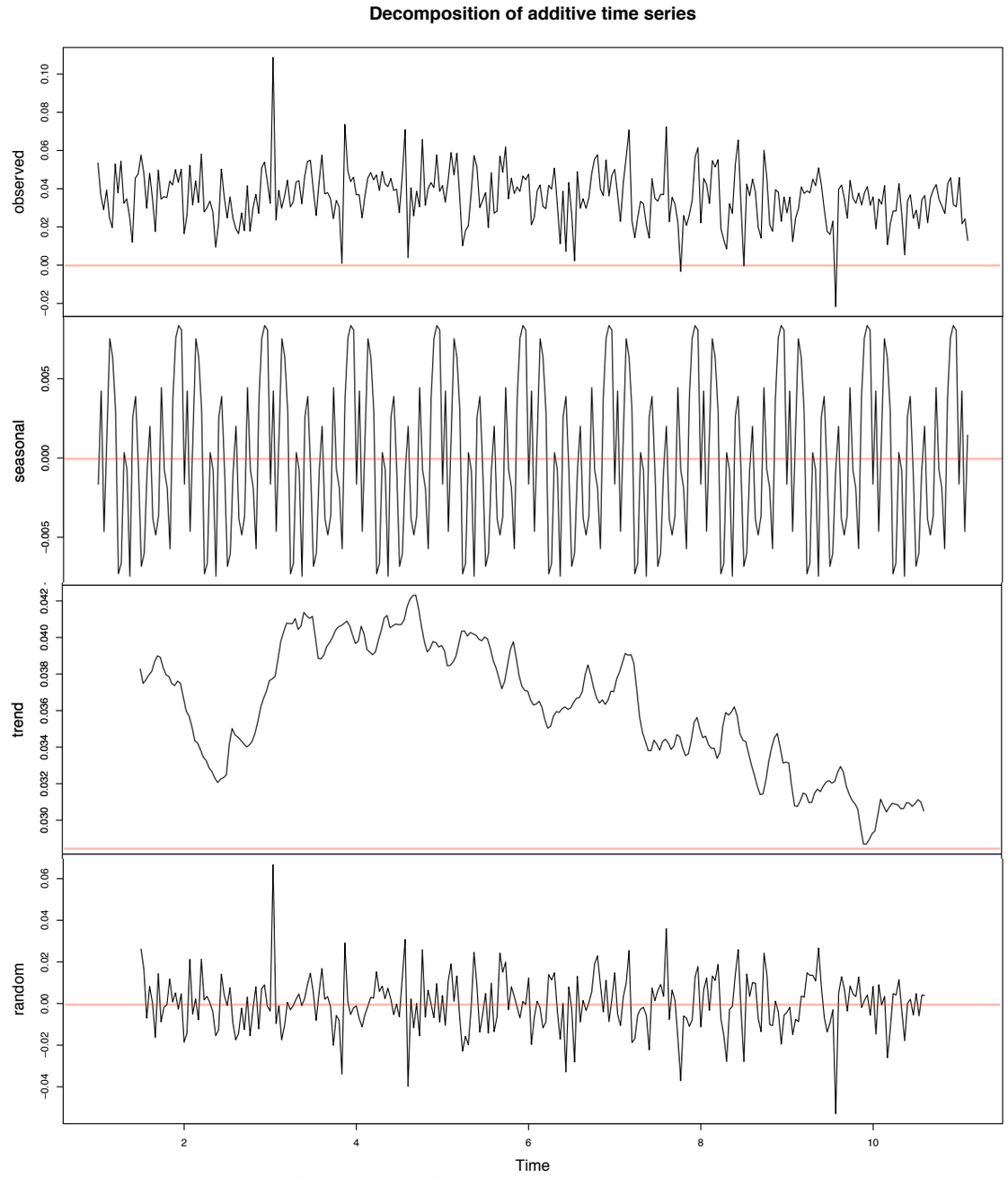


Time Series Analysis of Sentiments



Non-conflict-driven

vs.



Conflict-driven

Time Series Analysis of Sentiments: Anomaly Detection

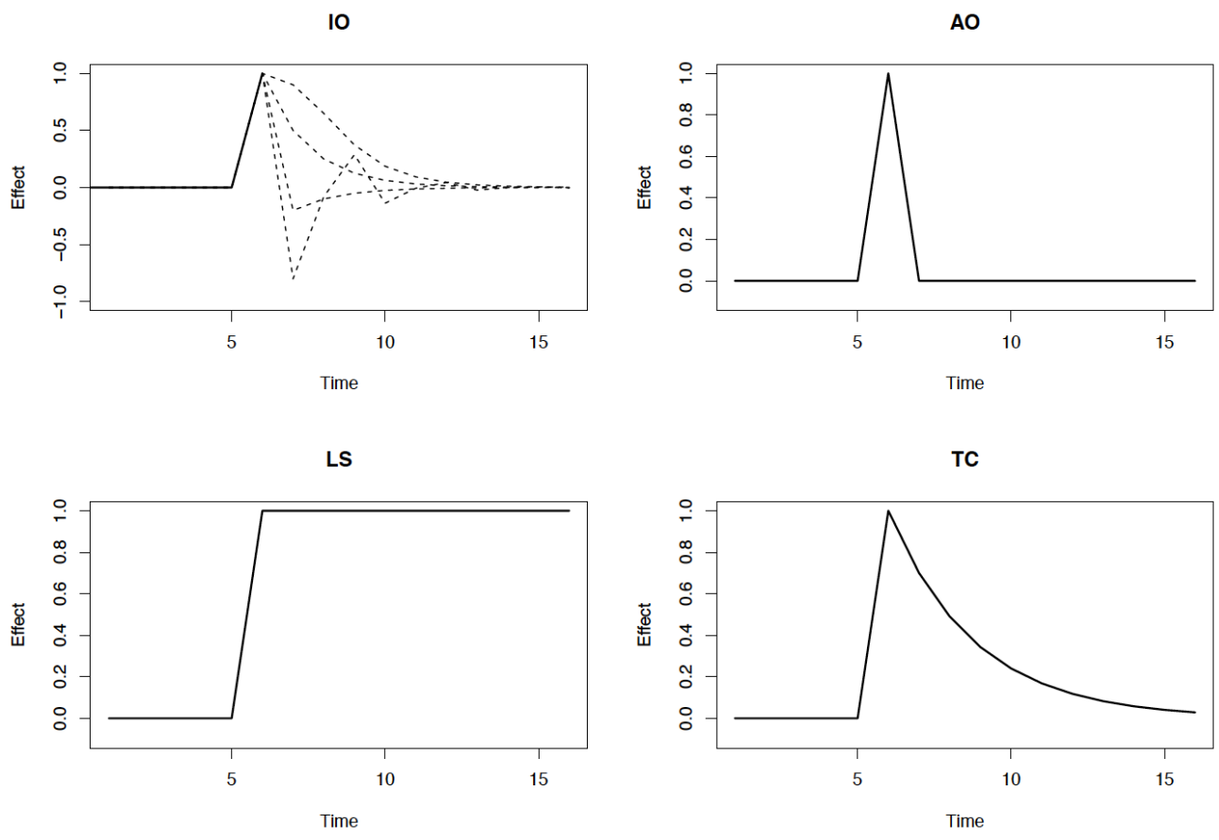
Observed series:

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

Four models for outlier effect:

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

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



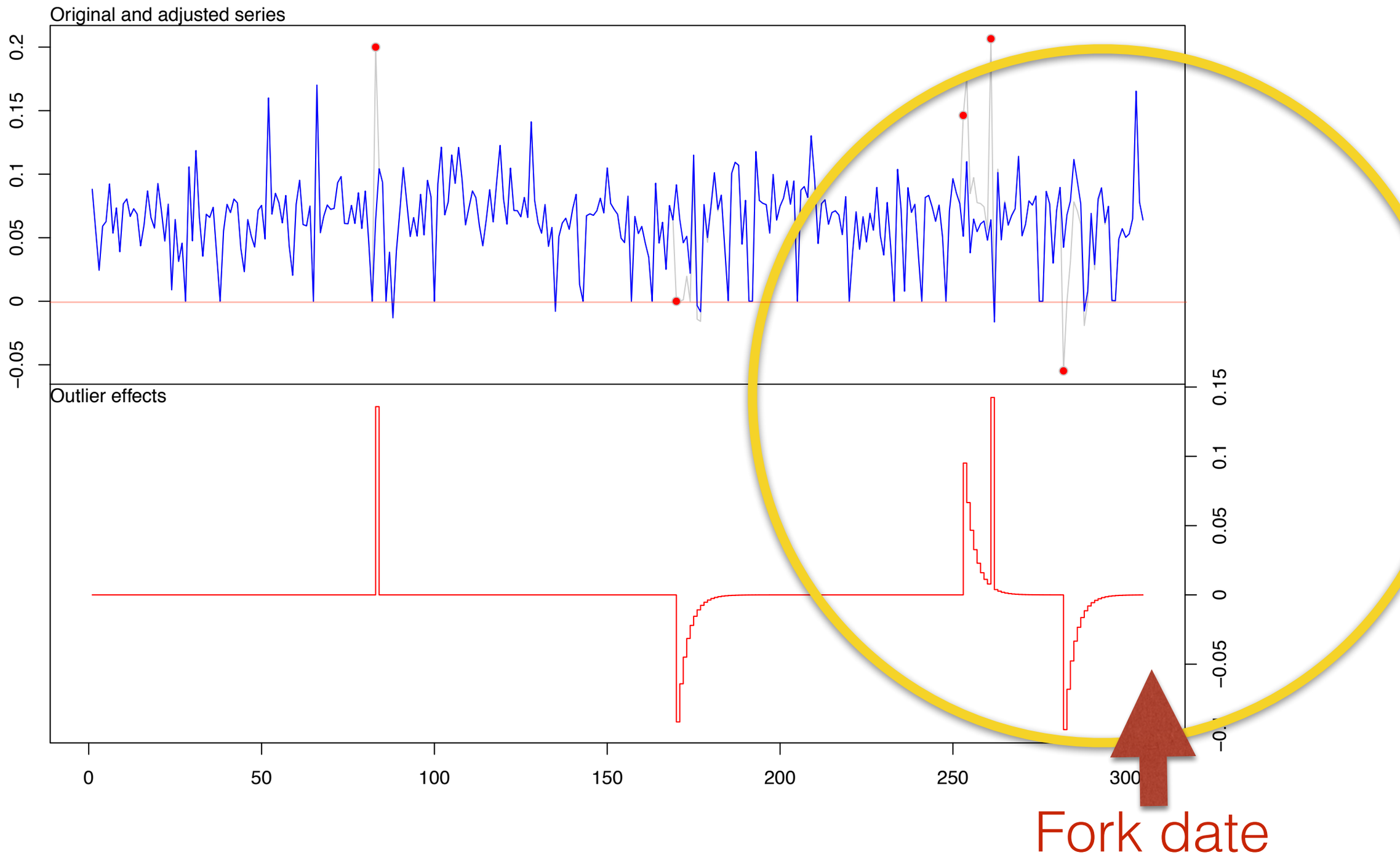
Example of negative sentiments

“[…] 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.”

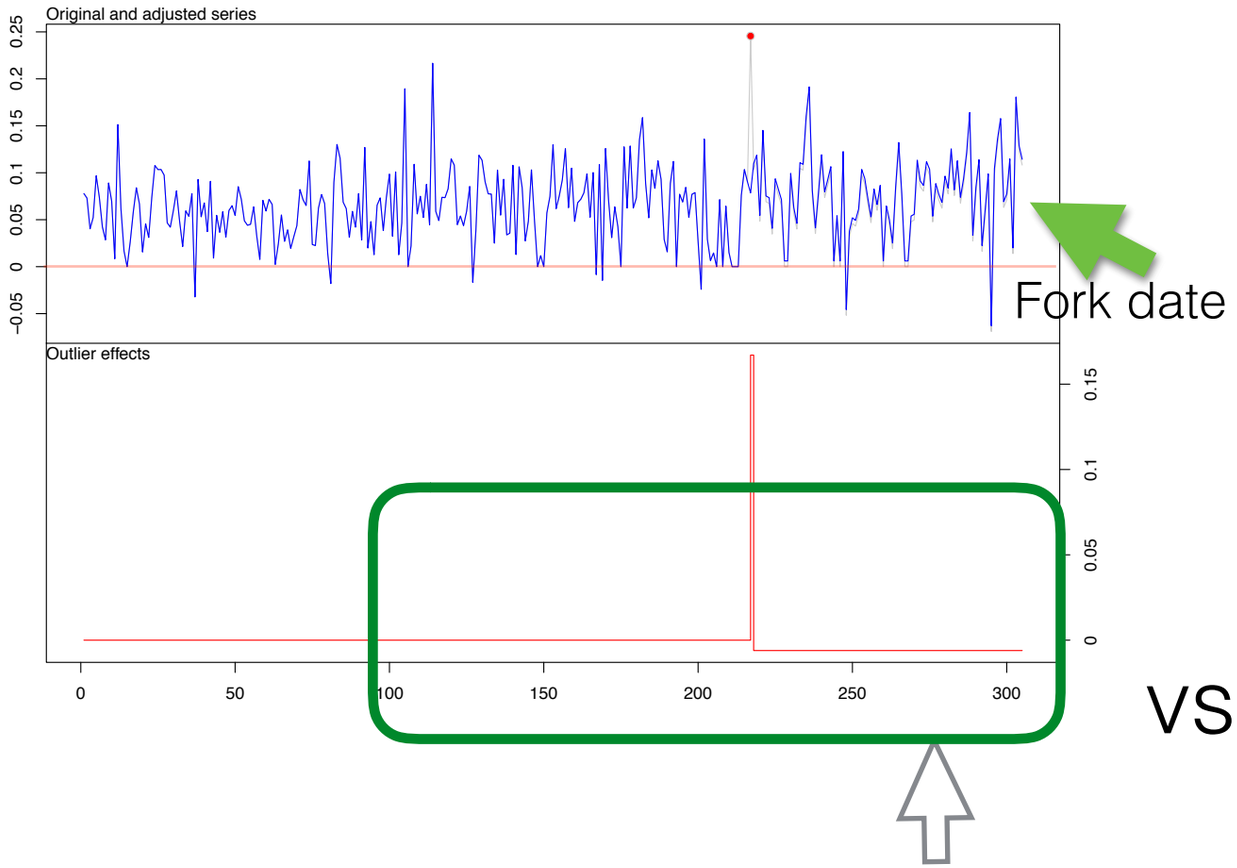
Example of negative sentiments

“thanks for the short announcement of the existence of this fork. 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.”

Time Series Analysis of Sentiments: Anomaly Detection



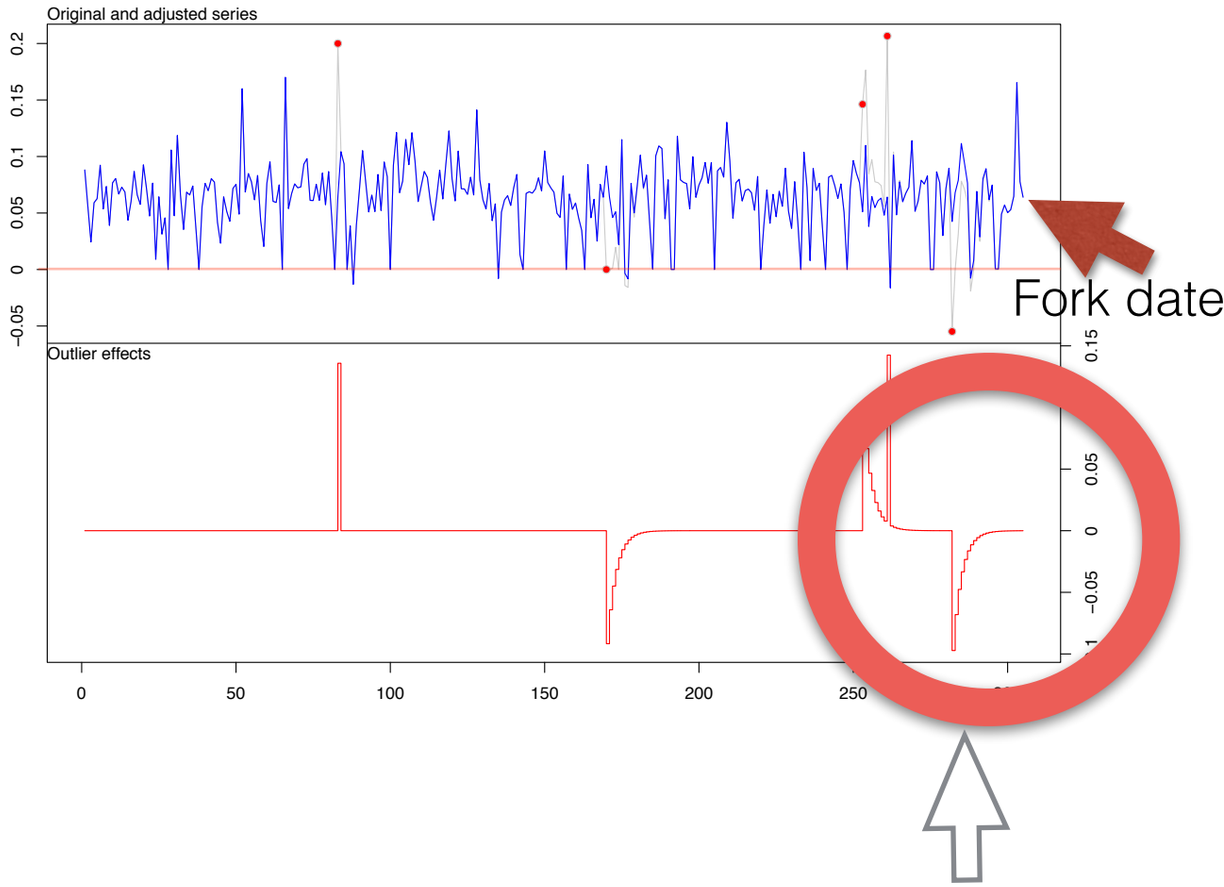
Time Series Analysis of Sentiments: Anomaly Detection



Normal, no negative outliers

Non-conflict-driven

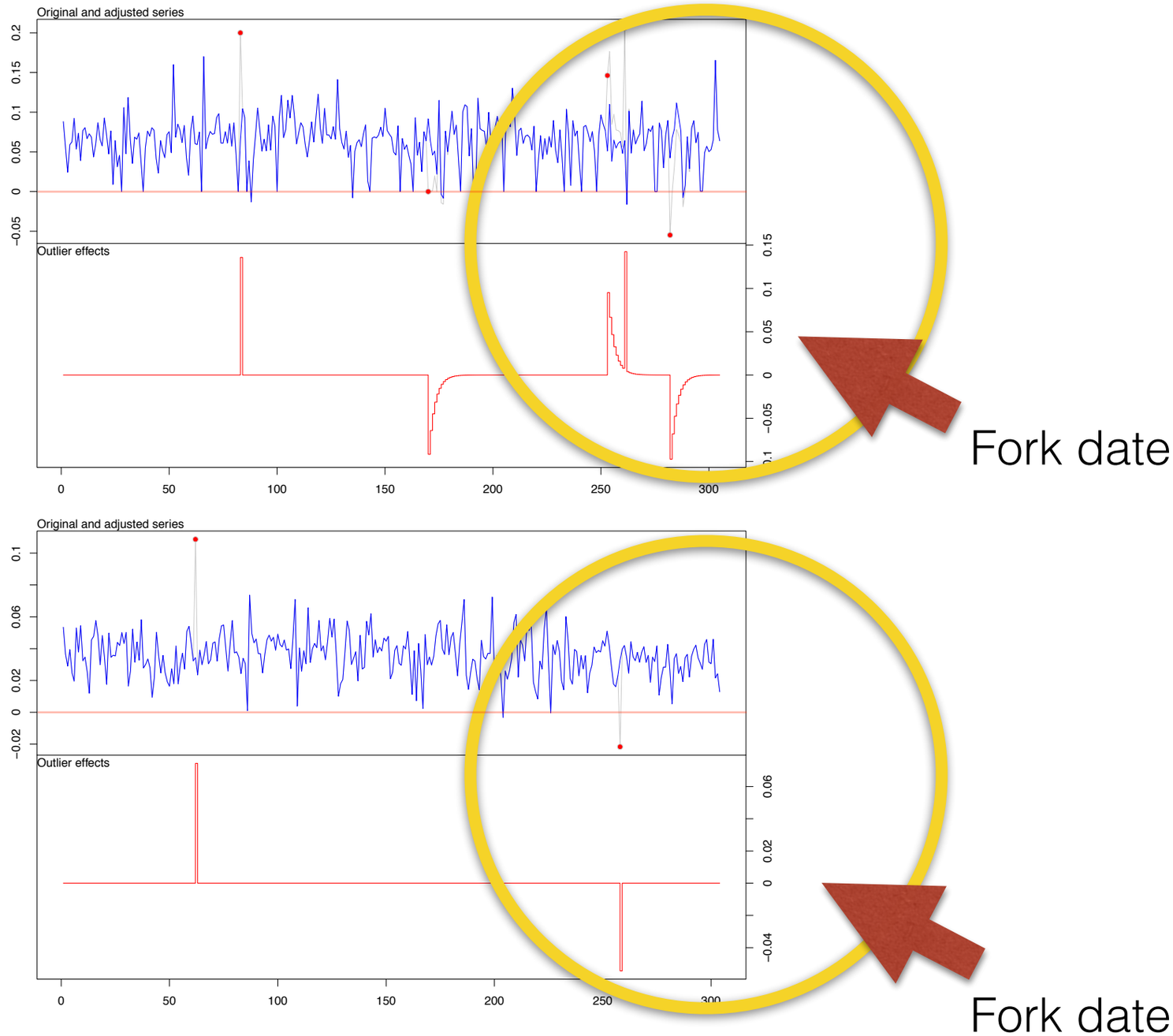
vs.



Anomaly, negative outliers

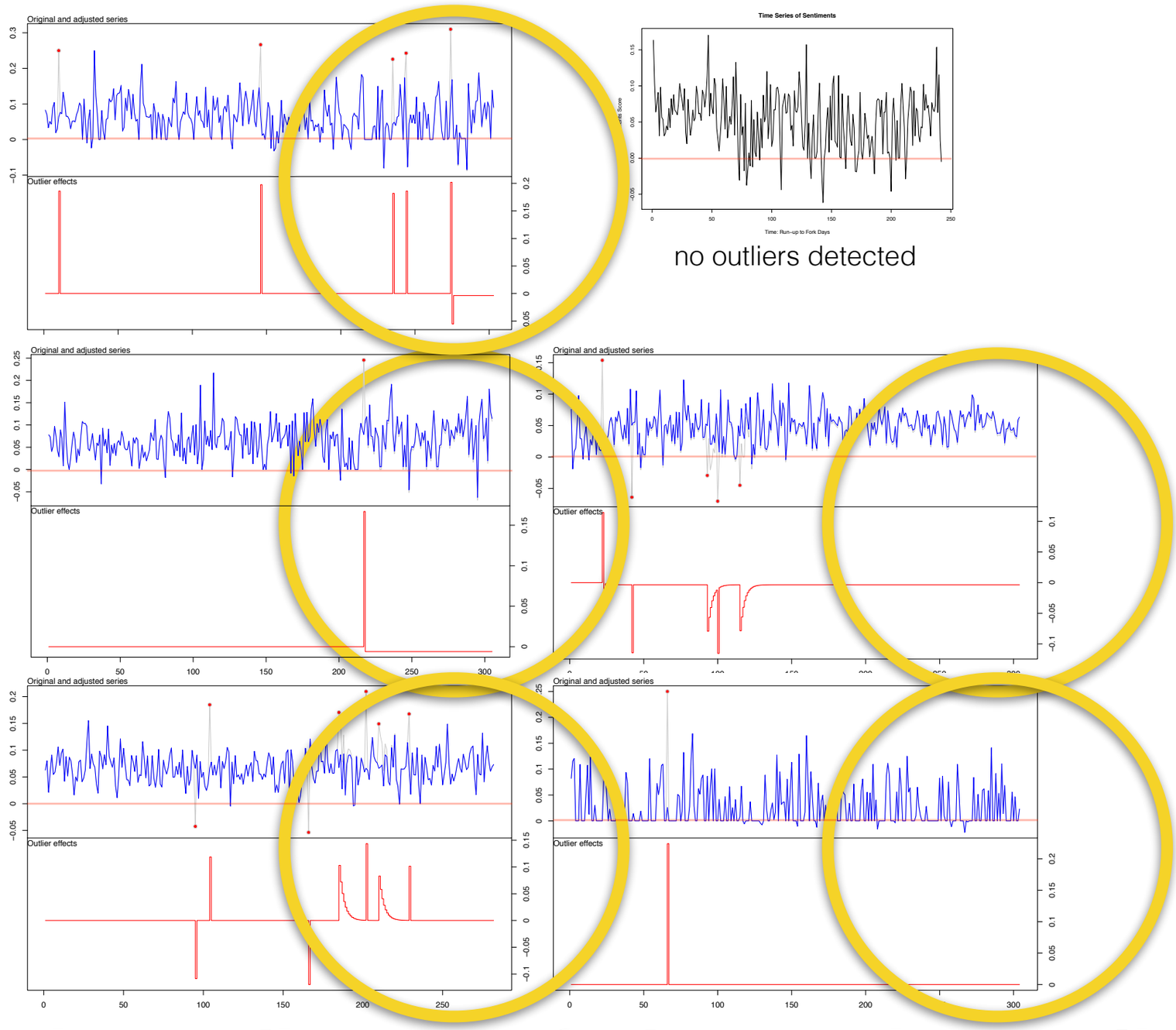
Conflict-driven

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

- We got close,
  - but can’t necessarily determine when one vs. other has occurred.

- So we turn to social network analysis, to figure out whether this is a storm in a teacup, or situation needing intervention.

- What do these networks look like, who’s involved, what are forces are shaping the community.

Social network analysis of developers' communication graphs

- Interactions modeled as graphs
  - Graphs change over time

Social network analysis of developers' communication graphs

- Mathematical models
- Lead us to estimates of the significance of several parameters' that affect this longitudinal change.

Results

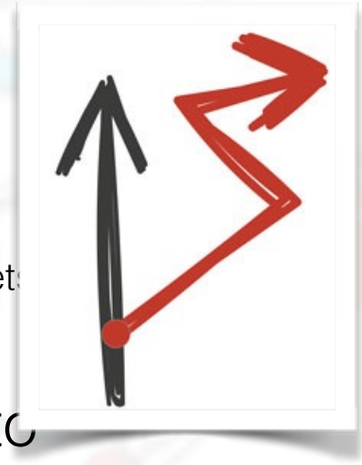
Social network analysis of open source developers' interaction graphs

	Negative three-cycles	positive transitive triplets	Out-out degree assortativity	Developer's source code activity
Conflict-driven 	Statistically Significant (*)	Statistically Significant (*)	Statistically Significant (*)	
Non-conflict-driven 				Statistically Significant (*)

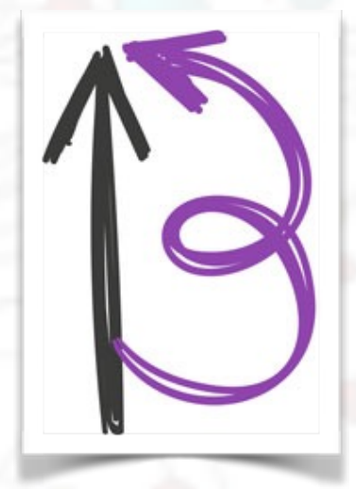
Results

Social network analysis of open source developers' interaction graphs

- Conflict-driven
  - Local hierarchy (Negative three-cycles (opposite of hierarchy) and positive transitive triplet
  - Tendency of heavy mailing-list poster developers to be tied to other heavy mailing-list poster developers (Out-out degree assortativity)



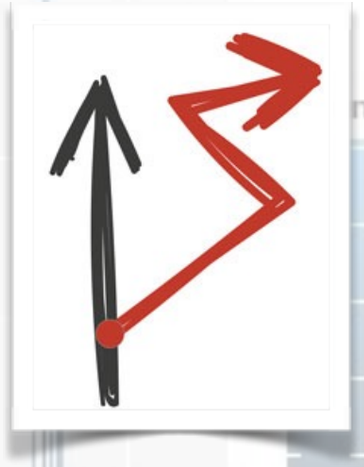
- Non-conflict-driven
  - Tendency of heavy code contributors to be tied to other heavy code contributors (Developer's source code activity)



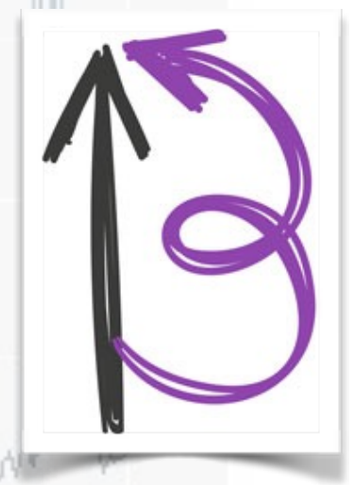
Results

Time series analysis of open source developers' communication sentiments

- Conflict-driven
  - Negative dips anomalies pre-fork

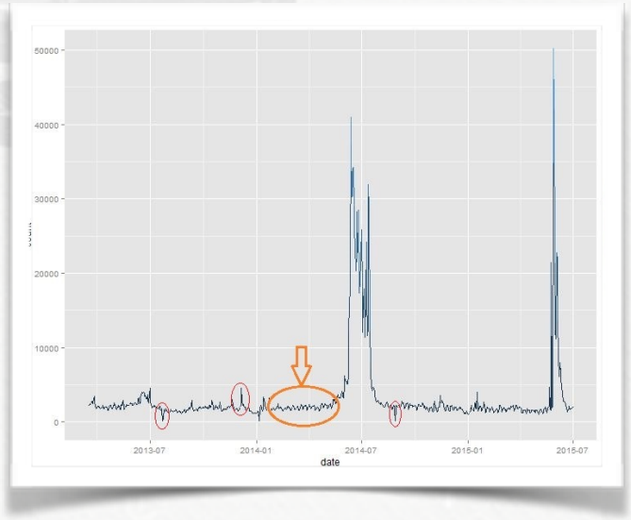


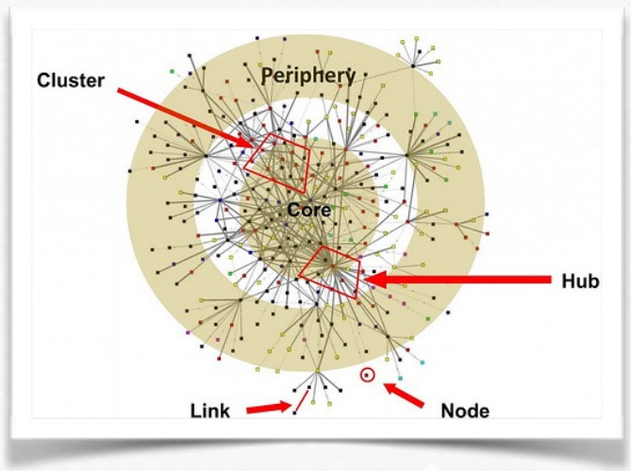
- Non-conflict-driven
  - Positive spikes anomalies pre-fork.



Summary

- Using the two techniques together, we believe, would be a good indicator of project health and project forking, however, this needs to be validated using a bigger dataset, when more data becomes attainable.



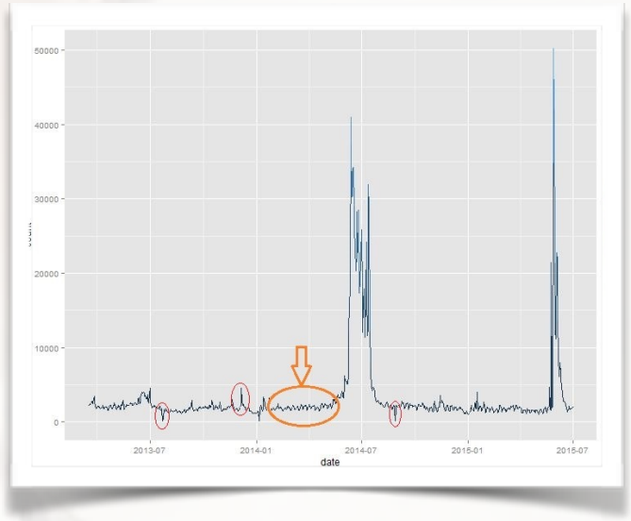


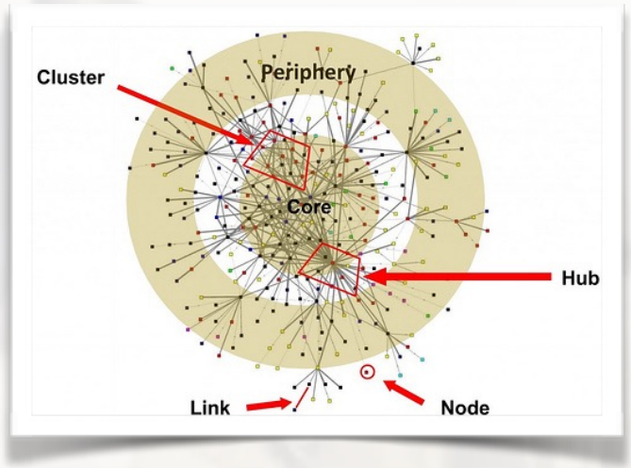
Summary

- 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 add-on)

Future Work

- More projects needed
- Apply into project management systems, e.g. Github integrate, to see if reflection changes community dynamics, or help make smarter decisions
- Does more transparency result in project self-regulation?





Acknowledgements

Prof. Carlos Jensen

Prof. Drew Gerkey

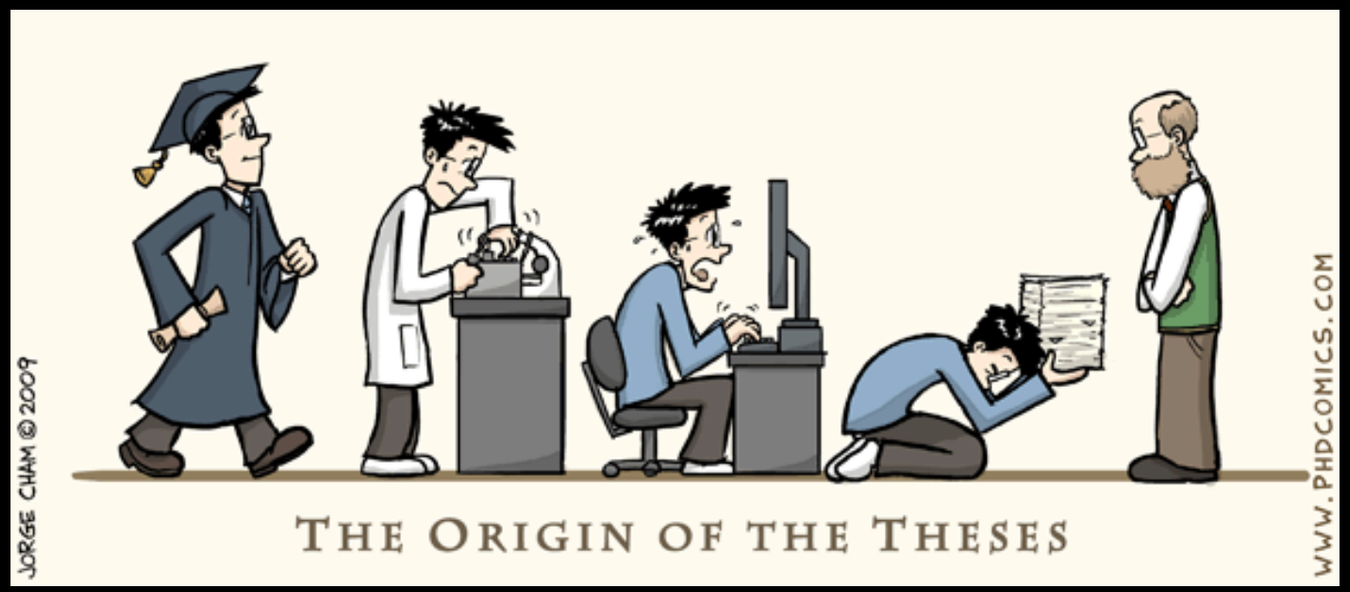
Prof. Ron Metoyer

Prof. Chris Scaffidi

Derric Jacobs

Prof. Cindy Grimm

Prof. Yelda Turkan


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

Back up slides follow

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.

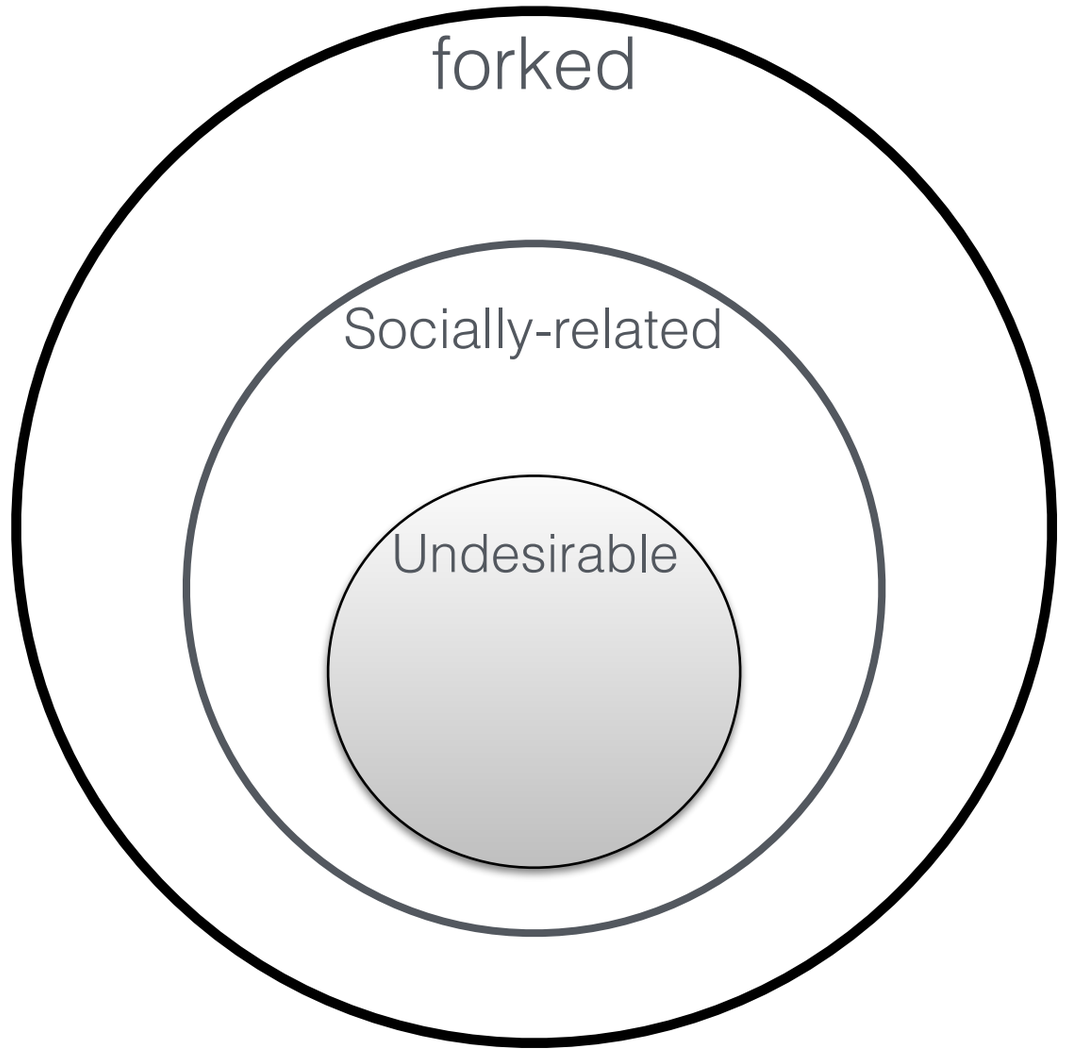
Why [some] Corporations Support Open Source?

- 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: ACF

Sentiment Analysis:
Time Series Analysis

Patterns

1. Trend: a non-periodic systematic change in a TS: a long-term change in the ‘mean’.

    - Can be modeled simply by a linear increase or decrease. (only if it's deterministic ~ it's non-stochastic).

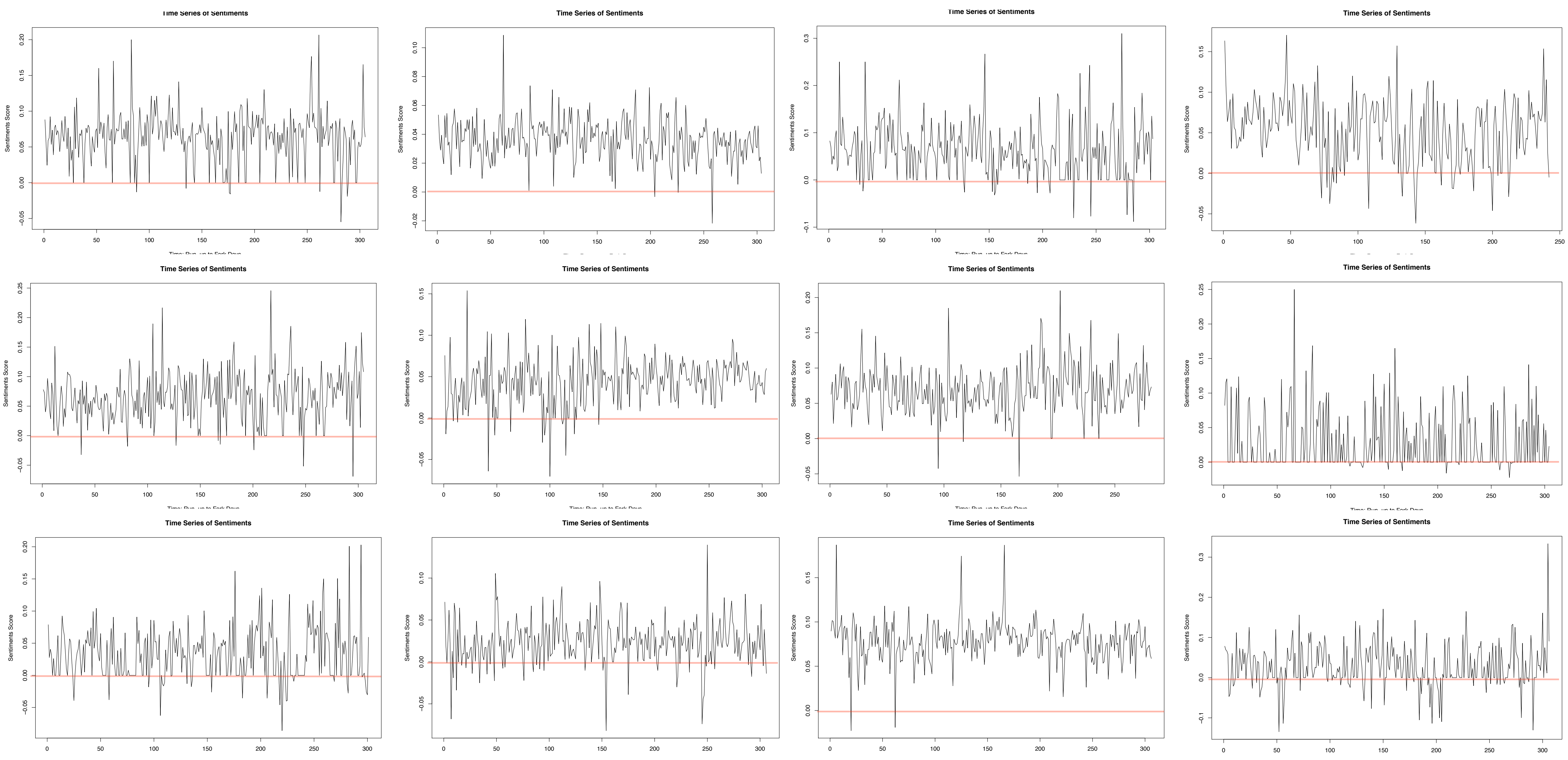
    - Stochastic trend: seems to change direction at unpredictable times rather than displaying a consistent pattern (e.g. like the air passenger series); inexplicable changes in direction

2. Seasonal variation: a repeating pattern within a fixed period (e.g. each year)

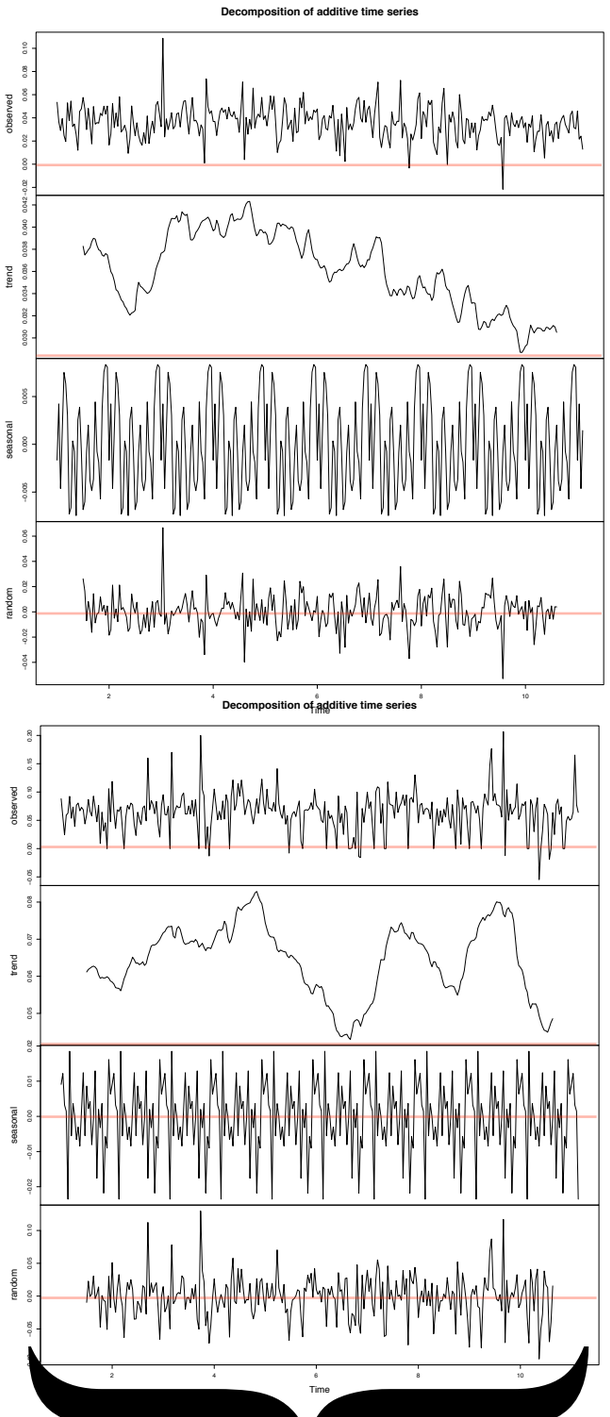
3. Cycles: a non-fixed-period cycle (without a fixed period). Example: El-Nino

4. Residual: Most interesting component for analysis.

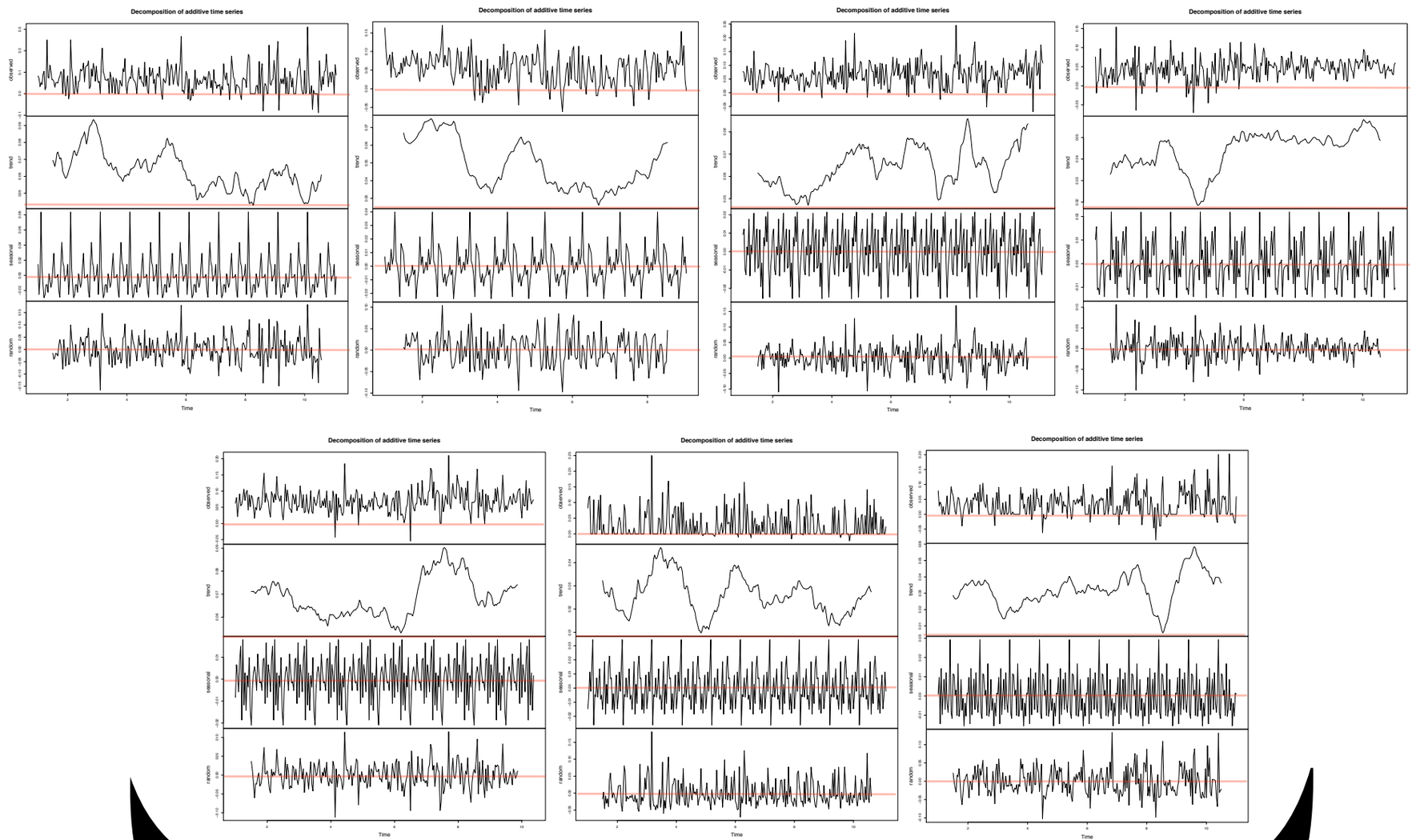
Time Series of Sentiments



Time Series Analysis of Sentiments

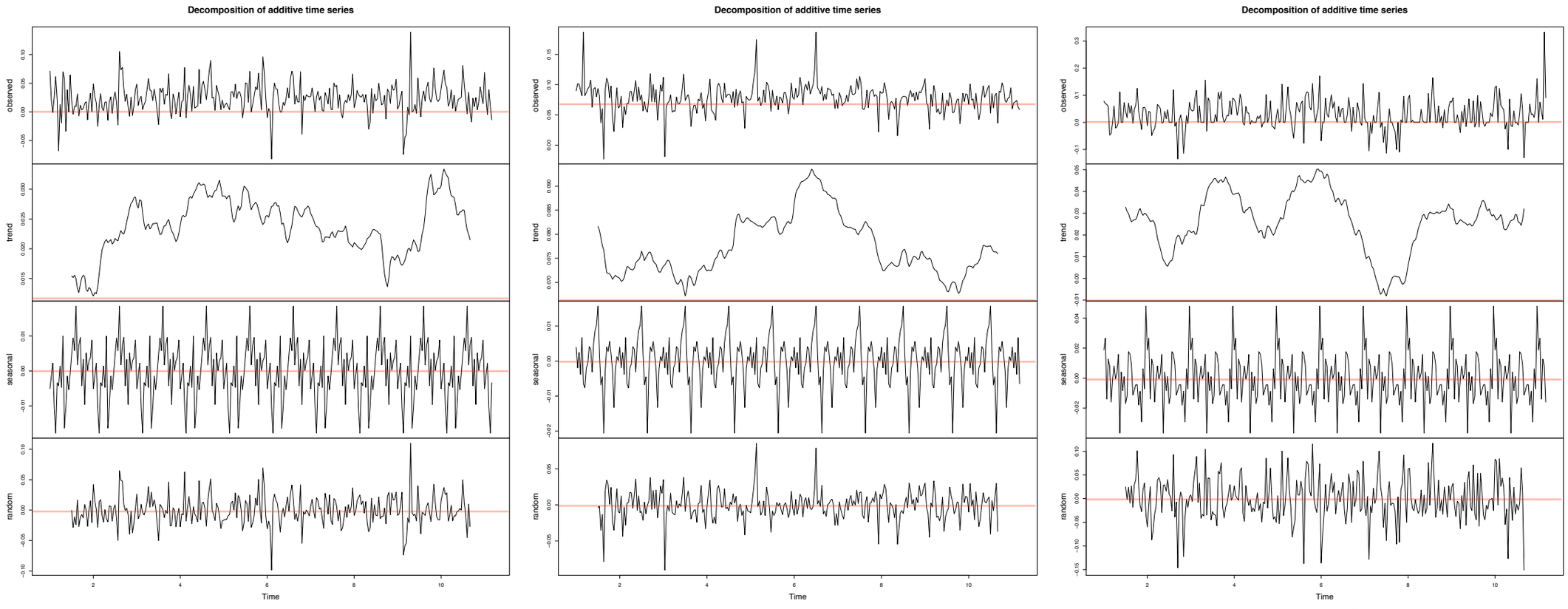


Conflict-driven



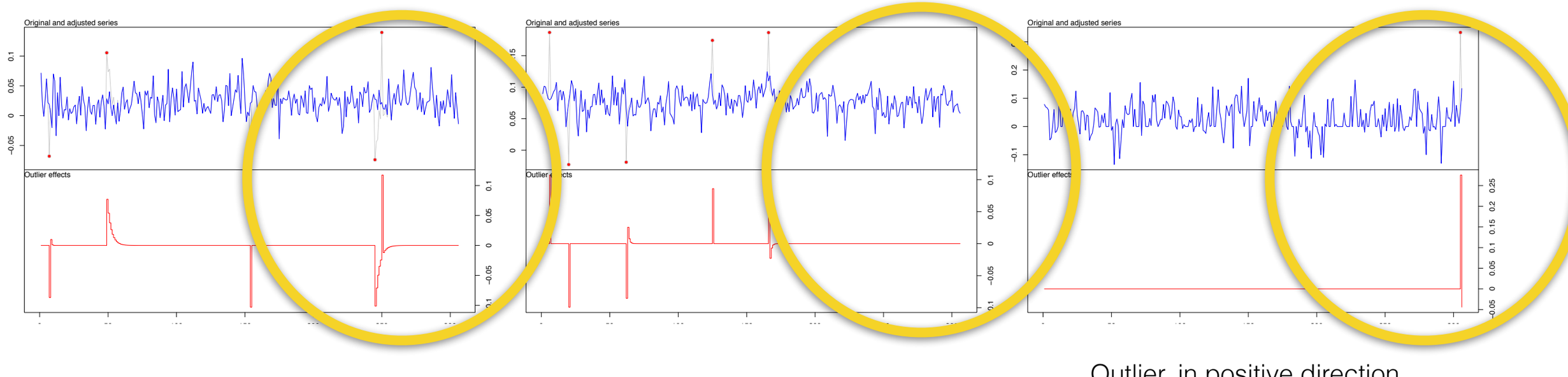
Non-conflict-driven

Time Series Analysis of Sentiments



No Fork

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

- 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

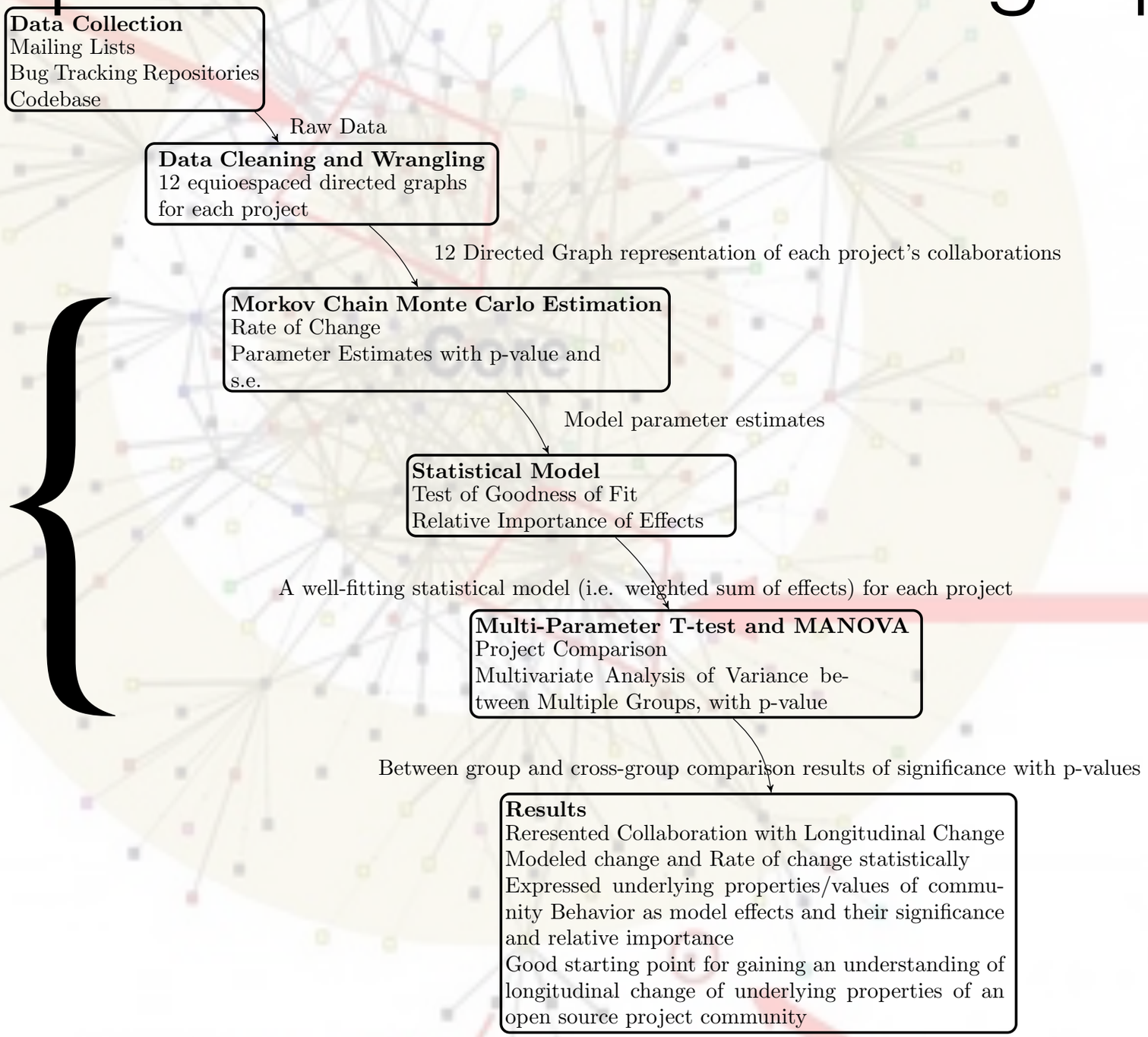
Social network analysis of developers' communication graphs

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.

Social network analysis of developers' communication graphs

Statistical Modeling



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:

TABLE

- 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.

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.

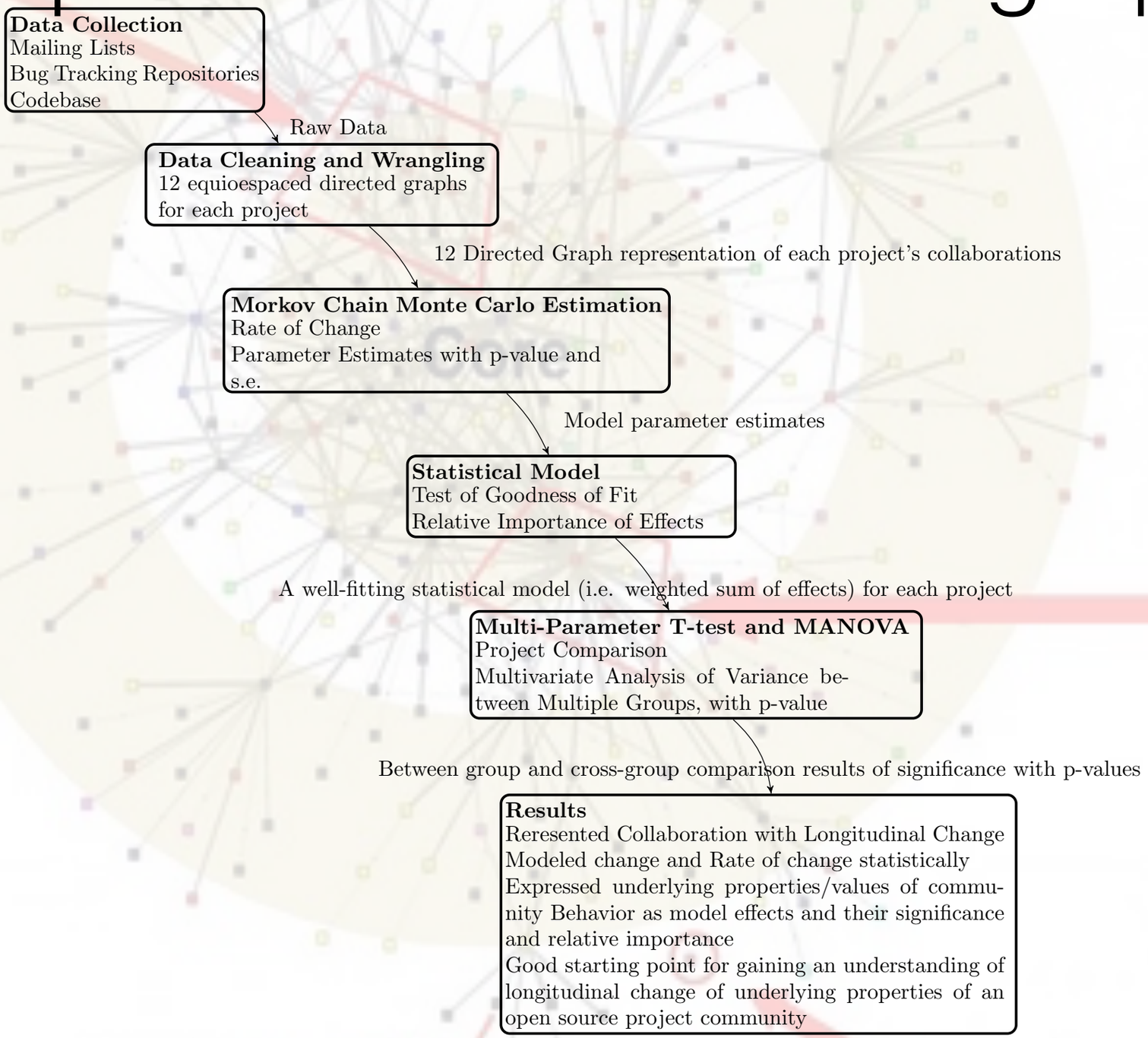
Here's why it matters:

- 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.

Social network analysis of developers' communication graphs



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)

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