Innovation Dynamics of Open Source Software

Masters Thesis

Of

Philippe Pautex

born on the 12th of March 1988 in Geneva

Supervisors:

Prof. A. Krause (D-INF)
Dr. T. Maillart (D-GESS/D-MTEC)

ETH Zürich
Eidgenössische Technische Hochschule Zürich
Faculty of Computer Science

March 19, 2012
Contents

1 Introduction 3

2 Background 4

3 Data 7
  3.1 Data Sets .................................................. 8
  3.2 Data Collection ........................................... 9

4 Preliminary Results 10
  4.1 Data Exploration .......................................... 10
  4.2 Self-excited Hawkes Conditional Poisson Process ........... 12

5 Method 13
  5.1 Hidden Markov Model ........................................ 13
    5.1.1 Motivation .............................................. 13
    5.1.2 Model ................................................... 14
    5.1.3 Assumptions ............................................ 15
  5.2 Filtering .................................................. 15
    5.2.1 Autoregressive Model .................................. 16
    5.2.2 Kalman Filter .......................................... 16
    5.2.3 Kalman-Lévy Filter ..................................... 17
    5.2.4 Particle Filter ......................................... 18
  5.3 Extensions ................................................ 19
    5.3.1 Multiple Model Kalman Filter ......................... 19
    5.3.2 Auto Regressive Model At File/Developer Level .... 21

6 Results 23
  6.1 Filtering ................................................ 25
    6.1.1 Auto-Regressive Model ................................ 25
    6.1.2 Kalman Filter ........................................... 26
    6.1.3 Kalman-Lévy ............................................ 27
    6.1.4 Particle-Filter ......................................... 27
    6.1.5 Filtering Overview ..................................... 29
  6.2 Extensions ................................................ 30
    6.2.1 Multiple Model Adaptive Estimator ................... 30
    6.2.2 Developer level predictions ........................... 32
    6.2.3 Files - Java Classes Level ............................ 34
    6.2.4 Logistic Regression .................................... 35

7 Discussion 35
  7.1 Filtering ................................................ 35
    7.1.1 Auto-Regressive Model ............................... 35
    7.1.2 Kalman Filter .......................................... 35
    7.1.3 Particle Filter ......................................... 36
    7.1.4 Filtering overview ..................................... 36
7.2 Extensions ............................................................ 37
  7.2.1 Multiple Model Adaptive Estimator ......................... 37
  7.2.2 Developers level predictions ................................. 37
  7.2.3 Files level Predictions ....................................... 38

8 Further Developments .................................................. 38

9 Conclusion ........................................................................ 39

A Demonstrations .............................................................. 44

B Methods Details ............................................................. 44
  B.1 Kalman Filter .......................................................... 44
  B.2 Kalman Lévy Filter ................................................... 45
  B.3 Fine Level Predictions ............................................... 46

C Projects Details .............................................................. 46
Abstract

In the world of open source, developers and their contribution dynamics are key for software production, including bug fixing or innovative software pieces. To better understand these innovation dynamics, we aim at developing tools to track, measure and predict the future activity of developers given the complex and modular nature of source code. The class of conditional point process models can be used to represent human activity. We adapt this model to a hidden Markov model.

First a method is proposed to filter out exogenous shocks that are not related to the innovation process (code evolution), by considering the characteristics of the projects. In the second application, we use a multiple model estimator; instead of having one memory kernel applied all over the development process, multiples parameters sets are tested in parallel to provide optimal results with different regimes of collective dynamics. Finally we attempt to predict the behaviors at the developer and file levels.

This research is a first attempt to systematically model the evolution of open source development, which often involves many heterogeneous developers with various incentives, skills and duties. Hopefully, this work will open perspective towards systematic large scale forecasting of open source software development and more generally of human collective action.

1 Introduction

Open source software production has grown exponentially over the last decades up to presumably more than one billion of lines of codes [1]. Not only it has grown fast, but in a very chaotic manner, with much self-organization [2]. People and even researchers tend to believe that social experience (i.e. various social networks) is the consequence of the Internet. Actually, collective action (i.e. working together for a common purpose) is less a consequence than a cause of today’s Internet: the open source culture dates back to the sixties and to the emergence of the hacker culture as an efficient way to achieve robust and innovative programs through peer-review and software reuse [3]. However, open source developers usually make their code available online for reuse or modification by anyone. For that, systems to track changes over time have been developed (control version software) [4]. While control version systems are powerful tools to follow the evolution of open source software projects, production is such that only a small portion of projects can be thoroughly inspected. Therefore, the expression "Given enough eyeballs, all bugs shallow” [2] certainly applies only to projects with the broadest exposure (e.g. Mozilla, Apache httpd, Eclipse). Today’s "bazaar” restricts perspectives at the meso- and coarse-grained levels (over large amount of projects) and also at the macro levels. This blindness prevents researchers of various fields ranging from computer science to economics, to clearly understand the mechanisms underlying this phenomenal collective production of software as a public good [5, 6].

Understanding the developers contribution dynamics is key for the improvement of software production and is, in various contexts, involving "bottom-up” organization [7, 8]. Past research has shown that many complex systems, involving human collective action could be modeled a class of point processes, usually called Hawkes self-excited
processes [9]. Using this framework, we aim at developing tools to predict the activity of developers given the complex and modular nature of source code. This document is organized as follow: we first review research efforts already taken to tackle the open source software contribution and modularity problems (section 2), we then present the data employed for testing our model (section 3), we present preliminary results (section 4), and we introduce a new method for modeling self-excited processes applied to open source software development as a hidden Markov process, and we present five models: autoregressive model, Kalman filter, Levy Kalman filter, and particle filter, as well as some extensions (section 5). We present the results and a benchmark of performance of each method (section 6), we discuss the results (section 7), present possible future developments (section 8), and we finally conclude (section 9).

2 Background

Version control systems as an innovation sensing device
In computer science, management and network analysis notably, a lot of research has been done for analyzing the behavior of developers in open source software. In computer science, the impact of open source on reliability [10], or the implication in software engineering [11] have extensively been studied. In management, version control data are also of interest for industrial organizational science [12]. Open source software is also a large complex adaptive system that has been studied for its dynamics and evolution. First T. Maillart et al [13] conducted empirical tests of Zipf’s law in Linux Distribution, analyzing the in-directed links growth between the different release of debian distribution. Then the thesis of Markus Geipel [14], gives a good overview about the Open Source what are the licensing implications and the innovation model. Then experiments are conducted for understanding the activity of developer and users in community forums and in the code development. Finally an analysis on the modularity is done on a set of 100 open source projects based notably on the work of MacCornack et al [15].

Exogenous versus endogenous shocks in human dynamics
In the endogenous versus exogenous model the shocks are divided in two categories, on one hand the exogenous shocks that come from outside the model and cannot be explained by it and on the other hand the endogenous shocks are fully explained by the model. It is an important concept for understanding the relative effect of cross-excitation and external shocks. The endogenous exogenous theory has been validated first on books sales in amazon [16] and later on the views of youtube video [17]. Finally T. Maillart et al. [18] analyzed the waiting time before people answering emails starting with queuing theory to show heavy-tails dynamics in human behaviors. The self-excited conditional Poisson process is conveniently used for modeling complex triggering effects; it states that each agent is subjected to endogenous shocks which are triggered with a given memory function, and exogenous shocks occurring as a renewable process [19]. The conditional intensity rate $\lambda(t|H_t)$ is the expected number of event during a unit of time, meaning $\lambda(t|H_t) * dt = E[N(dt)|H_t]$, and has been developed in [20] in the following way.

$$\lambda(t|H_t) = V(t) + \sum_{t_i \leq t} \nu_i \phi(t - t_i)$$ (1)
where $\nu_i$ is the number of potential persons who will be influenced directly over all future times after $t_i$ by person $i$ who acted at previous time $t_i$, $H$ the history of the previous events, $\phi(t)$ is the memory kernel and finally $V(t)$ the rate of exogenous activity.

For $\langle \nu \rangle > 1$ the process is supercritical with exponential growth [21]. At criticality ($\langle \nu \rangle = 1$), four possible regimes are obtained [22], and presented below and in Figure 1 in the context of YouTube videos [17]:

- **Exogenous sub-critical.** Activity spreading are relatively small, exogenous events does cascade only during the few next generations.
- **Exogenous critical.** Spreading is propagated through many generations, triggered user trigger the next generation.
- **Endogenous critical.** The burst of activity are not due to exogenous events but are fueled by endogenous shocks.
- **Endogenous sub-critical.** The response is largely driven by fluctuations and no clean peak of activity.

![Figure 1: A schematic view of the four categories of collective dynamics: Endogenous sub-critical (Upper Left), Endogenous critical (Upper Right), Exogenous sub-critical (Lower Left), and Exogenous-critical (Lower Right). (reproduced from Crane and Sornette, PNAS (2008) [17]).](image)

Two memory kernel functions are commonly used: (i) a power-law $\phi_{pow}$ and (ii) an exponential memory kernel $\phi_{exp}$,

$$\phi_{pow}(t - t_i) \sim a \ast \frac{1}{(t - t_i + c)^{1+\theta}} ,$$

with $0 < \theta < 1$, and $a = n \ast \theta \ast c^\theta$ \hspace{1cm} (2)

$$\phi_{exp}(t - t_i) \sim a \ast e^{-b\ast(t-t_i)} ,$$

with $a = n \ast b$ \hspace{1cm} (3)
where $n$ is the branching ratio, that is the number of daughter any event trigger in average. It is a key parameter for understanding the behavior of a system notably its stability. One of the most recent works in this field has been written in 2012 by V. Filimonov and D. Sornette[23], who introduced a new measure of activity that provide a direct access to the level of endogeneity. They tried to quantify the interplay between exogeneity and endogeneity. It gives new methods to characterize the robustness of systems, by analyzing the evolution of the branching ratio. The optimal branching ratio is found by log-likelihood maximization[20]

$$
LogL(\theta) = \sum_{i=1}^{N} \log \lambda(t_i, \psi) - \int_{0}^{T} \lambda(t; \psi)dt
$$

(4)

And the goodness of fit is tested by studying the residuals that compute the transformed time $\tau(i)$ that must be as near as possible as the cumulative number of events at time $i$. Therefore the graph with the transformed time in abscissa and the cumulative number of events in ordinate should be a line with slope of one and origin at zero.

$$
\tau(i) = \int_{0}^{t_i} \lambda(t)dt
$$

(5)

The key issue of this model is the parameterization, and the computation time for each step. Indeed the whole history must be considered at each new event and the log-likelihood is computationally expensive to compute. However it has some key advantages mainly that it handles very well sparse data, and works well in the analysis of the whole time series.

Endogenous vs exogenous on Open Source Software

Preliminary results show that[24] open source activity can be modeled as a Self-Excited Hawkes process using the model described in [20]. Indeed some activity, the innovation, is generated by new needs of the community or new ideas to improve the software. Innovation is considered as exogenous activities and can therefore hardly be predicted. These exogenous shocks will generate endogenous activity so that the new feature to be used in the best way all over the project. If a significant part of the exogenous shocks are due to innovation, in the sense of bug correction, performance improvement or features adding, some of these shocks are not part of any innovation but are fake shocks, which can be considered as outliers of the process. These shocks concern a majority of the files active at the time of the shock and we assume that the modification involves parts of the code that won’t be executed, such as licensing information in the header of files, or the files and folder hierarchy in the control version system (CVS).
Two interesting observation can be done based on Figure (2). First we can observe various regimes during the development time of junit. End of 2004 we can observe a clear Exogenous Sub-Critical regime with one main peak followed by short triggering. At the opposite around the middle of 2009 we can observe Critical Endogenous activity with each event triggering more events. Finally we can see that 3 shocks are much larger than the others and do not trigger later events with an amplitude that can be compared to the amplitude of these main shocks. Therefore we can assume that this 3 shock can be considered as fake exogenous shocks that should be reduced for a better prediction.

3 Data

Maven
Maven is one of the main software management system as of 2012, primarily for Java projects. It is mainly used to ensure the inter-compatibility between projects allowing automating project’s building. Maven is based on Project Object Model (POM) files, a file that contains all the information about a project given the artifact (name of the project), the group of the project and its version. It also specifies a very precise hierarchy for the files. In this data collection step the main role of Maven is to be a repository that contains a large number of Open Source Projects. Indeed, almost all projects are stored in a central repository named central maven, this repository keeps track of every project’s POM and in most of the cases for Java projects the corresponding jar files. More than a software building platform, Maven is a large complex network of more than 37’000 unique projects [25] with dependencies between all of them. In addition to the link between classes that can be found within the projects that are studied in this project, Maven provides a formally organized and codified complex network with package dependencies.
Repository Manager
A repository Manager is a layer between the user and the repositories, it usually allows simpler management between local and remote repository by caching the required files. Nexus Sonatype \(^1\) is one of the most commonly used with apache archiva\(^2\) and has been selected for this project because its rest API is very complete. In our case it is not used as a caching system but only to easily get the information from the repository. Indeed a search engine is included and the rest API as well as the structure of the web services make the overall research easier.

Software Control Management Revision Tools
In addition to Maven that allows building a project with dependencies to other projects, the big majority of the projects have a revision control system. Within Maven the revision controls systems are called Software Control Management Revision Tools (SCM). These SCM have two main goals, the first one is to keep a last version of the project online so that all developers are working on the last version. The second objective of SCM is to keep a history of all the modifications so that it makes easy to find an older version of the project. The first step for the developer is to checkout the project, meaning download the last version of the project then do the modification and finally when the developer has finished to change a part of the code he commits it to the SCM. If a lot of SCM are implemented (fully or partially) in Maven only 4 are used in the majority of the projects. Concurrent Versions System (CVS) [26] was developed in 1986 and is now followed by Apache Subversion (SVN) that is currently the most used. In parallel of these two open software, GitHub a commercial revision control manager is also commonly used mainly for its free accounts for open source projects. Finally mercurial has been designed for scalability and decentralization.

3.1 Data Sets
SCM Logs
For this project, two datasets are used. The first one is from small projects from Maven, and data are collected as described in 3.2, the second one is composed of 151 bigger projects and has been collected by Giacomo Ghezzi from the Department of Informatics at the University of Zürich. For the Maven projects 8 fields have been collected from the SCM’s information. The data of the larger projects have been arranged to follow the same structure without the information about the type of modification nor the number of lines added or deleted. Moreover the file name is a unique id for each file and therefore cannot be mapped to the Java class.

1. An Id to identify unique commits. \(^3\)
2. Name of developer
3. Date of commit in human readable format

\(^1\)http://www.sonatype.org/nexus/
\(^2\)http://archiva.apache.org/
\(^3\)Only for Maven projects
4. Date as UTC milliseconds from the Java epoch (January 1, 1970)
5. Type of Commit (M for Modification, A for file added, and R for file removed)³
6. Path to the file modified ⁴
7. Number of lines added³
8. Number of lines deleted³

3.2 Data Collection

The data collection step has been done in Java and the source code for logs crawling is available in GitHub ⁵.

Figure 3: Architecture for Central Maven Crawling, Nexus is used as repository manager and the process is split in two steps first getting all the search results then get the poms. Two Nexus can be used the one in cloud is the one online that is effectively used in this project. It can however also be installed locally and is represented in the box in this schema.

POM Files crawling

The first step is to crawl all the POM files available on maven central using Nexus Sonatype as the intermediary Rest API. Getting all the POMS files has to be done in two steps: the first is to index all the artifacts names and group ids that uniquely identify each projects and store them in an XML File (Search Results). Then all the POM Files are collected given their artifact and group ids. From the POM Files two data-sets are created the first one containing all the dependencies between projects and the second one contains a modified file with all the informations from all the POMs files for a given

⁴For Larger Projects only an ID
⁵https://imthephil@github.com/imthephil/Maven-Logs-Crawler.git
letter. The dependencies between projects are not used in this project but kept for further research. After the POM crawling, all the files were analyzed to only focus on the most used SCM, the process of getting the full logs from the SCM can be time consuming and in the scope of this project getting all the logs was not necessary therefore only SVN and GitHub were used. The extraction of the logs from SVN was done using SVNkit \(^6\). For Github the log files were downloaded using git and then the log files were reformatted. For this research all the logs were collected in November 2012, however we could imagine for further work collecting the logs in real time, meaning predict future changes whenever someone commits changes in a monitored project.

Table 1: Software Configuration Management Source Control Tool

<table>
<thead>
<tr>
<th>SCM Software</th>
<th># Of artifacts</th>
<th>% of projects</th>
<th>% of projects with scm</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO SCM</td>
<td>55291</td>
<td>79.12%</td>
<td></td>
</tr>
<tr>
<td>SVN</td>
<td>11475</td>
<td>16.42%</td>
<td>78.7%</td>
</tr>
<tr>
<td>GIT</td>
<td>1599</td>
<td>2.3%</td>
<td>10.1%</td>
</tr>
<tr>
<td>CVS</td>
<td>1006</td>
<td>1.4%</td>
<td>6.9%</td>
</tr>
<tr>
<td>HG</td>
<td>393</td>
<td>0.6%</td>
<td>2.7%</td>
</tr>
</tbody>
</table>

Inter-Classes Dependencies
For the second part of the project the dependencies between Java classes of given projects are required. To execute this task the first step was to download all the jar files of the project, it would give a snapshot of the project. The second step was to look for the creation time of the project, since the time on maven was the time when the jar was uploaded and not necessarily created the date of a given version was the date of creation of the MANIFEST file. Finally the dependencies between the classes were analyzed using Dependency Finder \(^7\) then the distance analysis was done using jGraphT \(^8\) and the visualization using iGraph \(^9\) for R.

4 Preliminary Results

4.1 Data Exploration

Maven
As results of the data collection, we got 311 well formatted logs in GitHub and 1107 from SVN. Some basic statistics have been done on these project for a better understanding of our data.

\(^6\)http://svnkit.com/
\(^7\)http://depfind.sourceforge.net/
\(^8\)http://www.jgrapht.org/
\(^9\)http://igraph.sourceforge.net/
We can observe that a majority of projects are extremely small with less than 5 developers, it might be an issue for this research, and therefore another set with larger projects is used in parallel for validation.

Figure 5: Start and end dates of projects. We can see that a main part of projects are still in development and a big part of them have been developed only recently.
Both Maven, SVN and GitHub are relatively new platforms. Therefore creating a new application for getting data from these growing platform is great opportunity for this project but also for further research.

**Large Projects**
The larger projects dataset contains 151 projects. This set contains significantly larger projects in the number of commits, number of developers and number of files. To conserve a fair computation time and a representative set of projects the overall research only the 100 bigger projects of the two dataset are kept.

<table>
<thead>
<tr>
<th>Mean of top 100 projects</th>
<th>Nb Commits</th>
<th>Nb Devs</th>
<th>Nb Files</th>
<th>Median Commit per dev</th>
<th>Max Commit Rate</th>
<th>nb weeks</th>
<th>mean activity</th>
<th>median activity</th>
<th>peaks</th>
<th>Sparsity</th>
<th>Skewness</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maven</td>
<td>10546</td>
<td>13.12</td>
<td>3677</td>
<td>1985.77</td>
<td>0.67</td>
<td>189.41</td>
<td>101.63</td>
<td>27.36</td>
<td>1.29</td>
<td>0.41</td>
<td>5.41</td>
<td>276.59</td>
</tr>
<tr>
<td>Large</td>
<td>176184</td>
<td>57.78</td>
<td>97945</td>
<td>17171.52</td>
<td>0.64</td>
<td>278.21</td>
<td>841.73</td>
<td>195.03</td>
<td>1.34</td>
<td>0.14</td>
<td>5.98</td>
<td>2334.08</td>
</tr>
</tbody>
</table>

Figure 6: Size Comparison between Maven projects and the Large Projects dataset.

### 4.2 Self-excited Hawkes Conditional Poisson Process

Using these data the first step was to try the commonly used the self-excited Hawkes conditional Poisson process to get a feeling about the data and the possibilities. The Pt-Process package for R [27] developed for earthquake predictions fulfills our needs for the log-likelihood fitting. The model is tested as described in the Background section with both power-law and exponential memory kernel, providing close results. The following figure shows the effective event number in function of the transformed time.

**Figure 7: Transformed Time junit**
As we can see in Figure (7) results are very promising, and the process is well predicted by the point-process model. Therefore it gives a strong incentive to use this model and to try to optimize it both for the computation time performance and for the precision of predictions. Due to the computation time for the log-likelihood estimation, the process is optimized by parallelization using the snow package for R[28] and dynamic programming. It reduces the computation time from more than 3 hours to 1 min on the same computer for Eclipse logs analysis. However, even with this significant improve of performance the computations are too slow for large scale studies, indeed the log-likelihood has to be compared with various parameters and on many projects. We find that this method might be inappropriate for large-scale studies, as we wish to develop.

Moreover, one of the final goals would be online finding of the parameters, meaning that data arrives at the rate they are created and parameters updated by integration of new data. Using the log-likelihood online computation of the branching ratio is impossible, it requires the entire dataset and maximize the log-likelihood for a unique branching ratio. A new method, that would allow the branching ratio to evolve over time has to be used (MMAE Developed in section 5.3.1).

The last reason for developing a new model is the lack of flexibility of the current one, the new model proposed in the following section allows more complex Markov models that allows filtering, fine level prediction in a simpler approach than the multi-variate model. Of course, the new model by binning the data suffers from the simplifications and cannot be used in all situations.

5 Method
5.1 Hidden Markov Model
5.1.1 Motivation
The Hawkes self-excited conditional Poisson point process is usually calibrated by log-likelihood. However, it is computationally slow, does not allow online parametrization, and the exogenous shocks are always considered without noise. We propose a new method that uses the endogenous exogenous model but not as a point-process but rather as an intensity predictor. Moving apart from the point-process by binning the activity in multiple bins allows us to take a new approach with a lot of new possibilities. Binning in weeks reduces the effect of periodicity and therefore simplifies the model by ignoring additional factor such as the day of the week, or the hour of the day.

In the model proposed, the current state is a combination of the previous state and the previous observed value. Applying the endo/exo model leads to a transformation for computing the next endogenous activity knowing the previous endogenous activity and the previous exogenous activity. This new model opens a large bunch of opportunities. First, it allows filtering for weighting the measurements errors based on various models for the exogenous shocks either as Gaussian Distributed (Kalman Filter), Heavy-Tailed Distributed (Kalman Lévy Filter), or variable rate (Particle Filter). Then, it allows inferring the branching ratio of the system at a given time, or to make precise predictions on the next developer who will commit or the next file committed.
5.1.2 Model

Two key features of the Hawkes self-exited conditional Poisson process are kept to create a new model for activity prediction instead of a point process. First the separation of endogenous and exogenous shocks and then the memory kernel.

It can be shown in demonstration (36) that an exponential memory kernel can be derived as a linear function of its value in the previous time bin. The exponential memory kernel is commonly[23] used for Hawkes self-excited because it is easier to compute and provide satisfying results. In our case the transition from one state to the next one cannot be linearized with the power law kernel as it is for the exponential memory kernel.

$$\phi_{exp}(k + dt - i) = e^{-b^*dt} * \phi_{exp}(k - i)$$

(6)

Where $k$ is the current time, and $i$ the time of the event. $\phi(k-i)$ is the endogenous activity at time $k$ triggered by event at time $i$. This linearization implies that the triggering of activity due to event $i$ at time $k + dt$ is a multiple of the triggering of event $i$ at time $k$.

Starting from this affirmation we can prove that the endogenous activity at a given time is a linear combination of the endogenous at the previous time step and the exogenous activity at this previous time step as shown in demonstration (37).

$$X_{endo}^k = \left( e^{-b^*dt} * X_{endo}^{k-1} \right) + \left( X_{ex}^{k-1} + X_{endo}^{k-1} \right) * a * e^{-b^*dt}$$

(7)

With $X_k$ the number of events between time $k$ and $k + dt$.

Figure 8: Hidden Markov Model of the process. Where $Z$ is the observation, and $X$ a vector of the combination of the endogenous and exogenous activity. We can observe that the endogenous activity is a function of the previous endogenous and the previous measurement as developed in 7.
5.1.3 Assumptions

There is a strong intuition that the exogenous shocks follow a power-law distribution after a given threshold as suggest many studies of empirical data [29]. The hypothesis is that below a given threshold the residuals are Gaussian Distributed and therefore can be considered as noise. The exogenous shocks are shocks with an amplitude larger than the threshold.

Based on this assumption and on the model described in Figure 8 we consider the exogenous shocks distributed with an heavy tail distribution and the measurements following a Poisson distribution of parameter $X_{endo} + X_{exo}$.

5.2 Filtering

In the first step we test this new model against the point process using a simple autoregressive model for the states transitions. It is assumed that all the measurements are corrects and therefore that all the exogenous shocks generate endogenous activity at the same level.

Then three filtering methods are compared, the goal of filtering is to find a value that is as near as possible as the true value of the system at a given time. At every step the prediction is done using the full history of the process that is summarized by the endogenous activity at the previous time plus the exogenous of the previous. This prediction is not an exact value but is within a state transition confidence. In parallel of this uncertainty in the prediction, the measurement is noisy, and therefore the true value of the system is a combination of the prediction value and the measurement. In open Source Development we do not claim that the measurement are noisy in the same sense as for object tracking for instance because the CVS provide exact values of development time, however all the information about the development are not in the CVS. Some information is missing; for instance the discussions on the community forums or external work on the project. Moreover some of the changes in the code are not real innovation but fake shocks that would not trigger any endogenous activity. Therefore the goal for using filtering methods in the development predictions is to reduce the impact of the fake exogenous shocks and to reduce the measurement noise.

The Kalman Filter is used when both the state variation and the measurements noise are Gaussian distributed. The Kalman Filter returns good approximation of heavy tailed filtering if the exponent is near to 2 [30]. However, if the exponent is larger than 2 the deviation is getting worst and the Kalman-Lévy Filter must be used. The Kalman-Lévy Filter, the second filtering method tested, is a modified Kalman Filter that eliminate the Gaussian noise assumption to allow heavy-tailed distributed shocks.

Finally the last filtering method used is the Particle-Filter that allows to have a heavy tailed distribution of the exogenous shocks and to generate an effective distribution of the exogenous events that vary over time. Instead of the assumption that the branching ratio change over time [23] that is developed in the multiple model adaptive estimator 5.3.1 the particle filter keeps the regime fixed but with an evolving rate of exogenous shocks.
5.2.1 Autoregressive Model

The auto-regressive model is a random process that is used for prediction of linear transitions models. In this project it is used first to measure the impact of data binning and then to provide a benchmark for the following filtering methods. The following notation is used in order to keep coherence all over the project.

\[
F = \begin{pmatrix} e^{-bsd_t} \ast (a + 1) & e^{-bsd_t} \ast a \\ 0 & 0 \end{pmatrix} 
\]

\[H = \begin{pmatrix} 1 & 1 \end{pmatrix} \]

\[X_{\text{prediction}} = F \ast X_{\text{update}} \]

\[\text{Residual} = Z - H \ast X_{\text{pred}} \]

\[X_{\text{update}} = \begin{pmatrix} X_{\text{end pred}} \\ \text{Residual} \end{pmatrix} \]

Where \(F\) is the state transition matrix and \(H\) the model observation matrix. The exogenous shocks are fixed to zero, meaning that the residuals are are equals to the exogenous contributions.

5.2.2 Kalman Filter

In 1960 R.E. Kalman [31] published a revolutionary recursive solution for discrete data filtering. The Kalman Filter has been used in many fields mainly in objects tracking and autonomous navigations. It allows making a precise prediction of the exact position of an object considering that the measurements are noisy. The Kalman Filter selects a value between the observed value and the model prediction that allows to avoid totally wrong prediction in case of unreliable measurements. The Kalman Filter in its classical definition has two key limitations. The first one is that the state transition function, given by the F matrix (8), must be linear, which is the case in our system and the second one is that the noise is Gaussian Distributed.

As presented in the assumptions in section (5.1.3), there is a strong belief that the exogenous shocks are heavy-tailed distributed. Since our goal is to reduce the amplitude of fake exogenous shocks, the measurement variance can be selected either too large so that the tail is well distributed but the small residuals don’t have enough weight, on the other hand selecting a variance too little ignore the tail. It has been observed that the larger events are usually not followed by high activity and can therefore be considered as fake exogenous events. Selecting a small variance reduces the impact of larger shocks and therefore following our assumption should reduce the impact of shocks that are not due to innovation and do not trigger later activity.

The Kalman filter is usually described in the following way using notation from [32]
\[ X_k = FX_{k-1} + BU_{k-1} + \omega_{k-1} \]  
\[ Z_k = HX_k + \nu_{k-1} \]  
\[ \omega \sim N(0, Q) \]  
\[ \nu \sim N(0, R) \]  
\[ Q = GG^T \cdot \sigma_{exo} \]  
\[ R = J \cdot J^T \cdot \sigma_{measure} \]

where \( X_k \) is the state of the system at time \( k \), \( U \) a control variable and \( \omega \) the state transition error. \( Z_k \) is the measurement at time \( k \) and \( \nu \) the measurement error. The state transition matrix \( F \) as well as the observation model matrix \( H \) are the same as for the autoregressive model. \( G = [0; 1] \) because the error in the state prediction are only due to exogenous shocks, \( J = 1 \), and \( U = [0; \mu] \) to add a constant amplitude to the exogenous shocks.

The selection of the variances \( \sigma_{exo} \) and \( \sigma_{measure} \) is an important step for the implementation of the Kalman Filter in general, in our case we focus on the state variance and fix the measure variance to a relatively small value. Indeed in our model since only one variable is noisy both are directly correlated.

The details of the algorithm are developed B.1. It is a simple model that is computationally efficient and have been used in many fields. The model has been used as a basis for many further development as the Kalman-Lévy Filter or the Multiple Model Adaptive estimator developed later in the project. However starting with the assumption made in 5.1.3 that the exogenous shocks are heavy tail distributed it doesn’t fully fit our requirements.

### 5.2.3 Kalman-Lévy Filter

The two limitation of the Kalman Filter is that the states transition must be linear and the noise must be white and Gaussian. The linearity issue can be solved by using Extended Kalman Filter or Unscented Kalman Filter[33], but these methods won’t be discussed since our system is indeed linear. However the distribution of the noise is an important limitation in our predictions and therefore two methods will be used to bypass this requirement, namely the Kalman-Lévy Filter[30] or the Particle Filter[34].

The Kalman-Lévy Filter is a method that provides a generalization to the case where error in predictions and noise in measurement have heavy tail distributions. It can be shown that the normal Gaussian filter recovers power law with \( \mu = 2 \), however the results are strongly deviating when \( \mu \) is far from 2. The key concept of this method is the ”Tail-Covariance” matrix \( B \) that generalizes the standard covariance matrix to the case of power-law.

The initial model is the same as the Kalman filter at the exception that the noise, and the predictions error are not distributed according to a Gaussian distribution but to a power law.

\[ x_k = Fx_{k-1} + \eta \]  
\[ z_k = Hx_k + \epsilon \]
With $\eta$ and $\epsilon$ are assumed to follow a power-law distribution of exponent $\mu$.

$$\eta = G_\eta * \omega_\eta$$
$$\epsilon = G_\epsilon * \omega_\epsilon$$
$$C_\eta = I * \text{var}_\eta$$

with $G_\eta = Q$ and $G_\eta = R$ with $Q$, and $R$ covariance matrix from equations (15) and (16). $B$ is the "Tail-Covariance" Matrix and can be decomposed as $B = G^{[\frac{\mu}{2}]} * C * G^T^{[\frac{\mu}{2}]}$ with $G^{[\beta]}_{i,j} = \text{sign}(G_{i,j}) * |G_{i,j}|^\beta$.

Again the details can be found in appendix B.2. The two free parameters are the variance of the process states, and the exponent $\mu$. For the same reason as for the Kalman filter the variance of measurement is fixed because strongly correlated to the variance of the process. Again cross-validation is used to determine the best parameters.

### 5.2.4 Particle Filter

Particle Filter[35] also called Sequential Monte Carlo Methods (SMC) is an estimation technique based on simulations. The idea is to represent the posterior density function by a sample of n particles and compute estimates based on the particles and weights. The weights are computed based on the estimated probability distribution function (pdf) and the particles are resampled if the weight are too different from each others. The resampling step redraws from the random distribution (Re-sampling Step). In our case the distribution for the particles is a power-law of exponent $\mu$ and the measurement noise follows a Poisson distribution.

Let $N_s$ be the number of particles, $\{x_{0,k}, i = 0...N_s\}$ the support points also called particles and $\{w_{i,k}, i = 0...N_s\}$ the weights associated to the support points. Weight are normalized such that $\sum_{i=1}^{N_s} w_k^i = 1$ and $\{x_j, j = 0...k\}$ are the true states of the system up to time $k$.

$$p(z_k|x_k^i) \sim \text{Poisson}(H * x_k^i)$$
$$w_k^i \propto w_{k-1}^i * p(z_k|x_k^i)$$

During the initialization a set of n particles are randomly chosen from a power law distribution. Then at each time-step a weight is given to particle according to the likelihood of the observed event knowing the value of the particle, then all the weights are normalized by the sum of all the weights.

If the sum of squared normalized weights is too big, meaning if few particles are responsible of explaining a big part of the measurement, a re-sampling of the useless particle is done. The re-sample is done as the initialization step with random value from a power law distribution.

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N_s}(w_k^i)^2}$$
The re-sampling is done if \( N_{\text{eff}} < N_t \), \( N_t \) fixed manually.

The particle filter is subject to an important trade-off that makes it computationally slow. To have significant result it must be insured that at least one particle explains the measurement. However since we are working with heavy tailed distribution the number of particle required to ensure extreme values is very large. On the other hand, particle filter is extremely slow to compute since it requires executing the evaluation for each particle. Therefore, the choice of the number of particles must be conducted carefully. Moreover, since the two parameters of the power-law distribution, the exponent and the scale parameter, are free and must be found using cross-validation this solution is extremely slow.

5.3 Extensions

5.3.1 Multiple Model Kalman Filter

The previous methods for the predictions have strong advantages and allow good repartition between the predictions and the measurements; however the parameters are found by cross-validation and therefore optimized for the whole process. We assume that the development process follow one unique regime all over its development duration. Since the goal of the project was to understand the behaviors of exogenous shocks and avoid having fake exogenous shocks with too high importance in the prediction, we focused on mono-regime prediction however multiple-regime could allow better predictions.

The Multiple-Model Adaptive Estimator\cite{36} has been developed in the seventies to allow online parameter estimation, considering that the system has one true parameter set. It uses a bank of Kalman Filter, at each time step every Kalman Filter from the bank computes its residuals and makes a prediction. Considering the residuals from all the previous time steps, the system computes a probability for each Kalman Filter to be the true one. Then the final prediction is the weighted sum of all the Kalman Filter, the weight being the probability of each Filter to be the true one. The MMAE is used for unknown mono-regime process however it can easily be modified to allow change of regime. Adding a prior probability to each Kalman Filter probability allows the system to change of regime from the given prior uniform probability.

There are two ways of using the multiple Kalman Filter, the first one and simplest one is to use a fixed number of \( J \) Kalman Filters meaning we provide different parameters to each of them but keep them fixed all over the process. The other method is to use a moving bank, meaning we change the parameters of the less likely Kalman filter to only use a bank of likely Kalman Filter. We will only focus on the fixed bank since our set of parameters is relatively small. Moreover since the data are extremely noisy we cannot expect a too high precision that can be provided by fine grain adjustment of the parameter set. The last reason for fixed bank is that it allows a better understanding of the data providing human readable results.
Figure 9: Schema for MMAE. The Kalman Filters are working in parallel and the final prediction is a weighted combination of all of them.

Lets call $\vec{a}$ a p-dimensional vector of unknown parameters in the system, for each Kalman filter we look at the conditional probability $p_j(k)$ that is the probability that $a_j$ is the good parameter set knowing $z$,

$$p_j(k) = P[a = a_j | Z(k) = z(k)]$$

(28)

It has been shown that $p_j(k)$ can be recursively evaluated from the following formula [36]

$$p_j(k) = \frac{p_j(k-1) * f(z(k) | a_j, Z_{0:k})}{\sum_{m=1}^{J} p_m(k-1) * f(z(k) | a_m, Z_{0:k})}$$

(29)

$$f(z(k) | a_j, Z_{0:k-1}) = \frac{1}{(2\pi)^{d/2} |S|^{-\frac{1}{2}} e^{-\frac{1}{2}y^T S^{-1} y}}$$

(30)

With $y$ the residual from equation (40) and $S$ the residual covariance from equation (41).

Knowing the hypothesis conditional probability $p_j(k)$ the final prediction can be make.

$$X_{MMAE}^{pred}(k+1) = \sum_{j=1}^{J} X_j^{pred}(k+1) * p_j(k)$$

(31)

This filter has been conceived to adapt to the good set of parameters meaning that the process won’t change of regime during its duration, however it can easily be adapted by adding a prior probability to $p_j(k)$ meaning that $P_j^{new}(k) = p_j^{prior} \cdot p_{prior} + p_j(k)(1 - p_{prior})$
with $p_j^{prior}$ are uniformly distributed over all the Kalman Filter and $p_{prior}$ a parameter with a small value. The key issue is that with the linearization of the process to Filter with different value of $n$ and $b$ can have very similar results it lead to too many change of regime. We might be afraid that the system would be too reactive to fake exogenous shocks.

5.3.2 Auto Regressive Model At File/Developer Level

After having a good prediction on how many commit will be done in the next time step, it is even more interesting to know which files and which developers will work in the next commit. Since the data added are orthogonal it is believed to provide more precise predictions. Moreover, it brings a considerable asset in management fields for a better understanding on interconnectivity between developers first and then between files. The main intuition is that the same file will be changed multiple consecutive times and that the same developer has period of work during which he is active. However, in addition of this first hypothesis we might observe that part of code that are dependent will be more often modified at the same time or in a near future.

The first step for this part is the analyze of the link between dependencies matrix and commit in the code. Then we apply the auto-regressive model at a more precise level to predict the activity on each file.

The model is be applied using the following method:

$$P_k = P(Z_k = 1|H_t)$$

$$P_k \sim \text{Logistic}(X_{k+1}^{endo})$$

(32)

$X_k$ an Hidden variable, vector of size $n$ (number of files/Developers). $X_{f,k}$ is the endogenous activity for a file $f$ at time $k$. 

![Figure 10: Schema for File/Developer predictions, With $p$ a vector of the probability of each file to be modified/Developer to commit and $Z$ the vector of modified files/Active Developers.](image)
Z_k A binary vector of size n: 1 if the file/dev has been modified at time k, 0 otherwise.

n the number of files/Developers.

X_{endo} and X_{endo+exo} of size (n x 1) one line for each file/developer

\[ X_{endo}^{k+1} = X_{endo}^k * e^{-b \cdot dt} + X_{exo+endo}^k * c \cdot A * a * e^{-b \cdot dt} \]  

(33)

The exogenous shocks are still fixed to zero as we can observe in the details B.3

**Choice of Matrix A** The key of this analysis is the Matrix A. The factor c is computed by cross-validation. The element A_{i,j} defines how an activity on file/developer i will generate an activity on file/developer j. First 3 kinds of A Matrix can be developed for both the developers or the files.

1. Files/Developer activate all files/developers equivalently.

   \[ A_{i,j} = 1 \quad \forall i, j \]

2. Files/Developers only activate themselves.

   \[ A_{i,j} = \begin{cases} 1, & \text{if } i \text{ is equal to } j. \\ 0, & \text{Otherwise.} \end{cases} \]

3. During the evolution of the project we change the matrix A, such that two files/developer having activity in a same bin are considered as close and therefore their mutual influence is increased in the A Matrix. This method can be compared as the mutual information of two elements at a given time.

At each time step k

\[ A^k_{i,j} = \begin{cases} (A^{k-1}_{i,j} + 1) \cdot \frac{k}{k+1}, & \text{If developer i and j worked at time } k. \\ 1, & \text{If } i \text{ is equal to } j. \\ (A^{k-1}_{i,j}) \cdot \frac{k}{k+1}, & \text{Otherwise.} \end{cases} \]

**Files - Java Classes Level** The first step described in section 3 is to extract the Dependencies Graph from the Jar files. This can be done using both directed and undirected graphs. The distance matrix D_{dir}/D_{undir} is constructed from this Basis having D_{ij} the number of hops between class i and class j. D_{dir} can be used in both direction since we would like to verify the way in which dependencies in the code generate spreading in the development.

Using this additional information we can have more information about the distance between two files and therefore use it as the A matrix.

1. Files generate activity to other files inversely proportionally to their distance in the undirected/directed class network. p a power that reduces the impact of other files
and increase the impact of one file to itself.

\[ A_{i,j} = (1/(D^\text{undir/dir}_{i,j} + 1))^p \]

2. File generate activity to other files directly linked to them and to themselves in a undirected/directed class network.

\[ A_{i,j} = \begin{cases} 
1, & \text{if } D^\text{undir/dir}_{i,j} \text{ is equal to 0.} \\
1/2^p, & \text{if } D^\text{undir/dir}_{i,j} \text{ is equal to 1.} \\
0, & \text{Otherwise.}
\end{cases} \]

**Receiver Operating Characteristic** Using this method for each time step and each file we find a value between 0 and 1, then we verify if the file was indeed modified or not. We call positive if the file has been modified.

Then we can fix a threshold from which the prediction will be positive. Meaning if the threshold is fixed at 0.5 all the predictions below 0.5 will be predicted as *This file won’t be modified* and if the prediction is higher than 0.5 as *This file will be modified*. Reducing this threshold increase the number of False Positive, and at the opposite increasing the threshold increases the number of False Negative or equivalently decrease the number of True Positive.

The TPR being the true positive rate \( \frac{\# \text{ of True Positive}}{\# \text{ of Positive}} \) and the FPR the false positive rate \( \frac{\# \text{ of False Positive}}{\# \text{ of Positive}} \)

Using this approach the threshold for positive prediction is changed and a ROC graph is produced. The baseline of a useless predictor is a diagonal line meaning the TPR is the same as the FPR for all thresholds.

The Area Under the Curve (AUC) is a commonly used metric to compare ROC graphs. Bigger the AUC is better the predictor is, if the AUC is 0.5 the predictor is a random predictor.

### 6 Results

This section presents the results for the methods described in the previous section. For the comparison of the methods the same procedure has to be applied all over the process. Therefore to keep consistency for the comparison of the methods the parameters \( n \) and \( b \), respectively the branching ratio and the scale parameter of the memory kernel are computed for minimization of the Root Mean Square Error (RMSE) in the autoregressive model by cross-validation. The Root-Mean-Square-Error (RMSE) is a goodness-of-fit metric that in our case gives less weight to the extreme residuals than log-likelihood.

\[
RMSE = \sqrt{\frac{\sum_{i=1}^{N}(X_i^{\text{pred}} - Z_i)^2}{N}}
\]  

(34)

Cross-validation is a commonly used method for parameters estimation and model
testing; it split the data sets in folds using the first folds for finding the best parameters and the last one for testing the model.

The methods are compared using an index of improvement that is computed in the following way. It represents the amount of improve the filter $F_2$ gave comparing to filter $F_1$.

$$I_{F_1,F_2} = \frac{RMSE_{F_1} - RMSE_{F_2}}{RMSE_{F_1}}$$

(35)

A large index means that the $F_2$ has fewer errors and therefore provide better results than $F_1$. At the opposite if the index is negative $F_1$ returns better results than $F_2$. For a better understanding of the characteristics that should be considered in the choice of a filter a set of 12 different projects’ characteristics have been selected. A linear regression on the improve index and the parameters is performed for comparison, if the reasons for the improve have to be studied more in detail.

1. **Number of Commits** Is a measure of the overall activity, composed by the total number of week of development and the average development by week. It’s a measure of the maturity of the project.

2. **Number of Developers** The number of developer can influence the way developer activate each others and it influence the way people work on the project, meaning that more developers requires more commits for the same amount of work.

3. **Number of Files** The number of file is notably important for the fake exogenous shocks in which it is believed that a significant part of the active files are modified. It also gives a measure of the complexity of the project.

4. **Median number of commit per developer** Minimum number of changes committed by the most 50% of the developers, it is a metric to understand how the amount of work is distributed among the developers.

5. **Contribution of most important developer** Mainly in the maven projects a significant part of the work is done by one developer, knowing the percentage of commit that is done by this main developer can like the median number of commit per developer give an insight on how is the work distributed among the developers.

6. **Number of weeks of development** Gives an insight about the maturity of the project.

7. **Mean Number of Commit per week** Gives an insight about the activity of the project.

8. **Median Number of Commit per week** A measure of the activity of the typical week, will be more resistant to the peak than the average.

9. **Peaks amplitude** $90^{th}$ percentile$-\frac{\text{Mean}}{\text{Mean}}$ That is a metric to compute the relative amplitude of the 10% higher peaks.

10. **Sparsity** $\frac{\text{Weeks without Development}}{\text{Total number of weeks}}$ The number of week without development is an interesting metric for predictions.
11. **skewness** is the lengths of the tail, meaning that bigger the skewness is more big events are frequent.

12. **standard deviation** A measure of volatility to fix how predictable the data is.

### 6.1 Filtering

#### 6.1.1 Auto-Regressive Model

For the larger data-set the point-process outperform the autoregressive model by 0.57% That is significantly different with a p-value of 0.028, however in the case of the Maven project the auto-regressive model outperform the point-process by 0.51% (p-value of 0.069). Since there is a big difference between the two data-sets, it will be interesting to understand the reason of this difference. (Figure 16)

![Figure 11: Data-set with large projects. Regression on the improvement between Point-Process and Auto-Regressive Model. The thin red bars are the 95% confidence interval for the slope. On the Y axis $I_{\text{Point,Auto}}$ means that for value <0 the point-process gives better results and at the opposite if >0 then the auto-regressive works better. We can observe that only the number of developer and the skewness of the data are within 10% significance correlated.](image)

We can observe in Figure (11) that only two of the parameters are significantly correlated with the RMSE improve. First bigger number of developer implies worst results of the Auto-regressive model. Then bigger skewness of the number of commit per bins implies better prediction with the auto-regressive model. From the maven projects the conclusions are equivalent.
6.1.2 Kalman Filter

For larger projects the Kalman Filter significantly outperform the autoregressive model by 0.25% however for smaller projects the Kalman provides significantly worst results by 0.06% (Figure 16) To understand this difference we will again have a look to the regressions.

Figure 12: Data-set with large projects. Regression on the improvement between Point-Process and Auto-Regressive Model. The thin red bars are the 95% confidence interval for the slope. On the Y axis $I_{Auto,Kalman}$ means that for value <0 the auto-regressive model gives better results and at the opposite if >0 then the Kalman Filter works better. We can observe that only the number of files and the sparsity are within 10% significance correlated.
Figure 13: Comparison of the predictions (On the left) and the residuals distributions of the Right for the first three method explained. We can observe as expected that the Kalman Filter will less over-predict after the shocks.

6.1.3 Kalman-Lévy

The Kalman-Lévy Filter outperforms significantly the Kalman Filter and shows no correlation between the improve comparing to the classical Kalman Filter. This can lead to the conclusion that the Kalman-Lévy Filter outperform the Kalman Filter without condition on the projects characteristics. However in computation time the Kalman-Lévy Filter requires to solve a non-linear equation and therefore can be more expensive.

6.1.4 Particle-Filter

For the particle filter the results are less obvious, for the larger projects it significantly increase the performance over the auto-regressive model but not over the Kalman Filter, for the smaller projects no significant conclusion holds.
On the particle Filter we can observe that it adapts to the rate of exogenous shocks for instance after the 20th week. The Kalman Lévy Filter will predict with the same trends, however the larger shocks have more impact than using the classical Kalman Filter.

Figure 14: Junit 4.8 Timeline prediction with Kalman-filter, Particle Filter and Kalman-Lévy Filter
Figure 15: Data-set with large projects. Regression on the improvement between Kalman Filter and the Particle Filter. The thin red bars are the 95% confidence interval for the slope. On the Y axis I(kalman,Particle Filter) means that for value <0 the Kalman filter gives better results and at the opposite if >0 then the Particle Filter works better. We can observe that only the number of developers, the activity of the main developer as well as the sparsity and the skewness are within 10% significance correlated.

6.1.5 Filtering Overview

<table>
<thead>
<tr>
<th>Larges</th>
<th>Auto-Regressive</th>
<th>Kalman Filter</th>
<th>Particle Filter</th>
<th>Kalman Lévy Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point-Process</strong></td>
<td>0.57 **</td>
<td>0.32</td>
<td>0.14</td>
<td>0.16</td>
</tr>
<tr>
<td><strong>Auto-Regressive</strong></td>
<td>-0.25 **</td>
<td>-0.46 **</td>
<td>-0.16 **</td>
<td></td>
</tr>
<tr>
<td><strong>Kalman Filter</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Particle Filter</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maven</th>
<th>Auto-Regressive</th>
<th>Kalman Filter</th>
<th>Particle Filter</th>
<th>Kalman Lévy Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point-Process</strong></td>
<td>-0.51 **</td>
<td>-0.45</td>
<td>0.42</td>
<td>-0.54</td>
</tr>
<tr>
<td><strong>Auto-Regressive</strong></td>
<td>0.06 **</td>
<td>0.92</td>
<td></td>
<td>-0.02</td>
</tr>
<tr>
<td><strong>Kalman Filter</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Particle Filter</strong></td>
<td></td>
<td></td>
<td>0.85</td>
<td>-7.69</td>
</tr>
</tbody>
</table>

Figure 16: Table for each RMSE of the residuals for each method.

The table in Figure (16) summarized all the results presented so far, if the results are fulfilling the presented hypothesis for the larger projects the results are usually not signifi-
cant for the projects automatically found in Maven that are usually smaller and involving fewer developers. However for the larger projects we can observe that the Kalman Lévy Filter is the best option for a broad scale of projects and the Particle Filter can be used in evolving projects with multiple generations of developers.

| 1 | # of Commits | 0.02 *** | 0.02 *** | 0.02 *** | 0.02 *** | 0.02 *** |
|   | 0.447 | 0.447 | 0.449 | 0.447 | 0.448 |
| 2 | # of Developers | -2.67 | -2.66 | -2.65 | -2.67 | -2.65 |
|   | 0.013 | 0.013 | 0.013 | 0.013 | 0.013 |
| 3 | # of Files | 0.02 *** | 0.02 *** | 0.02 *** | 0.02 *** | 0.02 *** |
|   | 0.349 | 0.351 | 0.354 | 0.351 | 0.352 |
| 4 | Median C./Developers | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
|   | 0.022 | 0.021 | 0.021 | 0.021 | 0.021 |
| 5 | Commit % main dev. | 416.35 | 385.73 | 364.32 | 386.99 | 363.25 |
|   | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| 6 | # of weeks | -5.89 ** | -5.93 ** | -5.91 ** | -5.93 ** | -5.90 ** |
|   | 0.041 | 0.041 | 0.041 | 0.041 | 0.041 |
| 7 | Mean activity | 2.41 *** | 2.44 *** | 2.44 *** | 2.44 *** | 2.43 *** |
|   | 0.637 | 0.644 | 0.646 | 0.644 | 0.646 |
| 8 | Median Activity | 4.11 *** | 4.17 *** | 4.17 *** | 4.17 *** | 4.17 *** |
|   | 0.445 | 0.453 | 0.456 | 0.453 | 0.456 |
| 9 | Peaks amplitude | -927.80 ** | -931.91 ** | -925.77 ** | -931.68 ** | -923.69 ** |
|   | 0.058 | 0.058 | 0.058 | 0.058 | 0.057 |
| 10 | Sparsity | 4501.25 | -4543.78 | -4570.62 | -4537.78 | -4555.97 |
|   | 0.021 | 0.021 | 0.022 | 0.021 | 0.022 |
| 11 | Activity Skewness | 340.27 ** | 336.47 ** | 333.78 ** | 336.51 ** | 333.35 ** |
|   | 0.065 | 0.062 | 0.062 | 0.062 | 0.062 |
| 12 | Activity Std. | 1.35 *** | 1.36 *** | 1.35 *** | 1.36 *** | 1.35 *** |
|   | 0.796 | 0.795 | 0.795 | 0.795 | 0.795 |

Figure 17: Regression Table for each RMSE of the residuals for each method. Table for the biggest projects.

We can observe in table fig:regr-table that 8 of the selected parameters are significantly correlated with the precision of the prediction, it’s however complicated in this table to compare the results between the different methods. For that we have to refer to the comparison regression table presented earlier for each methods.

6.2 Extensions

6.2.1 Multiple Model Adaptive Estimator

We will use the MMAE with 6 different filters, 3 Level for the branching ratio n (0.5,1 and 2) and 2 Levels for b (0.8 and 2). There is a strong incentive to limit the number of Filters first because a Significant part of the signal is noise and therefore taking too many filters would over-fit the data. Then it would change too often of regime and therefore not be meaningful. Moreover keeping low number of filters will allow a clear understanding of the more exact regime at a given time.
Figure 18: Junit MMAE With Filter 1 and 4 sub-critical ($n = 0.5$), 2 and 5 critical ($n = 1$), 3 and 6 super-critical ($n = 1.5$). Filter 1-3 $b = 0.8$ and 4-6 $b = 2$. We can observe that during the super-critical phases between week 60 and 75 The Filter 3 is the most used one as expected.

<table>
<thead>
<tr>
<th>MMAE</th>
<th>1 # of Commits</th>
<th>2 # of Developers</th>
<th>3 # of Files</th>
<th>Median C./Developers</th>
<th>5 # of main dev.</th>
<th>6 # of weeks</th>
<th>7 Mean activity</th>
<th>8 Median Activity</th>
<th>9 Peaks amplitude</th>
<th>10 Sparsity</th>
<th>11 Activity Skewness</th>
<th>12 Activity Std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.02 ***</td>
<td>2.65</td>
<td>0.02 ***</td>
<td>0.01</td>
<td>593.65</td>
<td>5.76 **</td>
<td>2.38 ***</td>
<td>4.01 ***</td>
<td>915.49 **</td>
<td>4486.99</td>
<td>340.15 ***</td>
<td>1.34 ***</td>
</tr>
<tr>
<td>R² 2</td>
<td>0.458</td>
<td>0.013</td>
<td>0.358</td>
<td>0.021</td>
<td>0.001</td>
<td>0.040</td>
<td>0.638</td>
<td>0.434</td>
<td>0.059</td>
<td>0.027</td>
<td>0.066</td>
<td>0.807</td>
</tr>
</tbody>
</table>

Figure 19:

We can observe that the significant parameters are the same compared to all the previous methods.
<table>
<thead>
<tr>
<th>MMAE Improve in %</th>
<th>Maven Projects</th>
<th>Larger Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-Process</td>
<td>-4.51 ***</td>
<td>-0.98 ***</td>
</tr>
<tr>
<td>Auto-Regressive</td>
<td>-3.94 ***</td>
<td>-1.62 ***</td>
</tr>
<tr>
<td>Kalman Filter</td>
<td>-4.01 ***</td>
<td>-1.36 ***</td>
</tr>
<tr>
<td>Particle Filter</td>
<td>-11.59 *</td>
<td>-1.15 ***</td>
</tr>
<tr>
<td>Kalman Lévy Filter</td>
<td>-4.15 **</td>
<td>-1.20 **</td>
</tr>
</tbody>
</table>

Figure 20:

As expected the MMAE outperforms all the previous methods. However it can hardly be compared since it adapt more precisely to the data over time.

6.2.2 Developer level predictions

Since the mapping between file hierarchy and class hierarchy has to be done manually, and that the jar dependencies are taking long to compute the results following are done on Junit project and have been verified for consistency on Joda with equivalent results.

Figure 21: Histogram of probability for both Positive and Negative. On the left is the probability of having a modification, knowing that a modification has been done. On the left knowing that no modification has been done. Then the threshold is varied in the X axis. This graph is again for Junit with the last A matrix described in Section 5.3.2 with parameter p=3, n=1, b=1.5

On Figure 21 we can observe on the right that the prediction are really accurate for predicting when nothing happens (99% of the cases). However, we can on the other hand get good predictions when files are committed, moreover the False positive rate can easily be modified by selecting the threshold for predictions.
Figure 22: This graph is for Junit with basics A matrix described in Section 5.3.2 with parameter p=3, n=1, b=1.5. We can observe that the identity matrix performs the best when low true positive rate is required but the Matrix with the A Matrix that consider the joint contributions can provide higher TPR.

Indeed modifying the threshold gave the following results. For the only ones matrix, meaning that all the developers activate each others equivalently, the results are interesting only as a benchmark. However for the two others matrix we get good prediction that are relatively close, however the evolutive matrix can achieve slightly better TPR with some loss on the False Positive.
6.2.3 Files - Java Classes Level

Figure 23: This graph is for Junit with the last A matrix described in Section 5.3.2 with parameter \( p=3, n=1, b=1.5 \). We can observe that the identity matrix performs the best when low true positive rate is required but the Evolutive Matrix outperforms the other for high true positive rates.

Figure 24: This graph is for Junit with on the right the A matrix composed on undirected files and on the left on directed files. We can see that the transpose gives slightly better results. Since the blue and red line tends to be higher than the green and purple.
Taking the undirected dependencies matrix can provide high true positive rate with some loss on the true positive rate. We can observe in Figure (24) that the transpose of the D matrix provides better results, meaning that using this model if Class B calls a method in Class C there is more chance that the developer will first modify B and then C.

6.2.4 Logistic Regression

As a test to justify the previous results a logistic regression has been applied on the same data, taking up to 5 time-frames before, for the developer case it returned an AUC of 0.75 with a best TPR of 0.58. For the files taking 3 time-frame before (the optimal result) the AUC was of 0.58.

7 Discussion

7.1 Filtering

7.1.1 Auto-Regressive Model

The Auto-Regressive model is assumed to give less precise results than the point-process in the majority of the cases, therefore it has to be ensured that the results of the auto-regressive model are acceptable.

A regression is performed on the characteristics of the projects to understand the consequences of the markov model presented and it provides two interesting results (i) the number of developers impacts the consequences of the binning. If they are less developers the binning doesn’t decrease the predictions performance, however with large amounts of developers the binning significantly decrease the quality of the results. This tends to promote various bin size depending on the number of developers involved in a projects; however since the bias induced by the day of the week has to be taken in consideration it will not be tested during this project. The commit rate by developer highly depends on the total number of developer involved in the projects or even active at the same time[37]. Therefore the rate of cross-excitation is higher if more developer are active and then the binning induce a larger loss of information.

Considering that the number of commits per week follows a heavy tailed distribution the skewness represents the length of the right tail. Since we measure the skewness of the data binned in weeks, large skewness implies that more weeks are in the right tail. For the weeks in the right tail binning the data leads to a significant loss of information.

We can conclude that depending on the projects the results of the Auto-regressive model can vary significantly however the loss don’t exceed 5% of the point-process performance. Therefore we can admit that binning the data provides results that are good enough to continue in this direction.

7.1.2 Kalman Filter

We can observe again from the dataset with bigger projects that two parameters are correlated with the performance improve due to Kalman Filter over Auto-regressive Model. First of all, the performance of the Kalman Filter increases with the number of files. There is a strong assumption that the fake exogenous shocks affect a number of file that
is not correlated with the number of lines modified. Therefore, projects with bigger number of files are likely to have higher fake shocks, and the Kalman filter moderates them more accurately. The second correlated parameter is the sparsity of the data that involve less precise results from the Kalman Filter. This correlation can be easily explained; if the data is mainly empty then the shocks are less likely and don’t have their full weight in the predictions even if they are innovation exogenous shocks.

The Kalman Filter is a good way to reduce the impact of extreme shocks, therefore it outperforms the auto-regressive model when the number of file is extremely high, meaning that the number of file changed during a fake exogenous shock is high. On the opposite, it shows some limitation if the data is very sparse and that a majority of the shocks are significantly higher than expected.

7.1.3 Particle Filter

The key difference between the Particle-Filter and the filters previously used is that the exogenous shocks are not modeled as zero but are sampled from an heavy tailed importance density. Therefore the prediction can adapt to the rate of exogenous shocks. The conclusion from the Figure 15 is that the more developers are involved the better the particle filter is. One of the reasons would be that when the number of developers is larger its more likely to observe multiple generations of development. Therefore, parameters found by cross-validation are not optimal and adapting the exogenous shocks all over the process gives a significant advantage. For the sparse data the particle filter never predicts zero since the exogenous shocks are randomly draws. Therefore the particle filter performs badly in the situation of a majority of the week are inactive. Since the projects on maven are generally involving few developers and are relatively sparse the results are worse than with the Kalman-filter even if nothing is significant.

7.1.4 Filtering overview

We can observe that 9 of the 12 parameters selected are significantly correlated with the quality of predictions.

- **Number of Commit:** The number of commit is correlated with the Root Mean Square Error (RMSE) because the bigger the number of commits, the bigger the amplitude of the signal is. And therefore, the bigger is the noise even if the signal noise ratio is not bigger.

- **Number of Files:** In addition to the remarks from the number of commits that are also valid for the number of files, the fake exogenous shocks are usually higher if the number of file is bigger and therefore the prediction is significantly worst. The correlation with the number of commits is the same as with the number of file.

- **Development duration:** The bigger the number of weeks, the better the prediction. During the cross-validation step we split the development time in three folds, and try to find the better parameters knowing the first two folds for testing over the last fold. The bigger the folds are, the less likely it is that the difference between the folds will be significant and therefore that the parameters will be different.
• **Mean and Median Activity:** Same comments as for the number of commit and the number of files. The more there is signal the more there is noise.

• **Peaks Amplitude:** Bigger the difference between the peaks and the 90th percentile is better the results are, if the opposite can easily be explained because bigger peaks are usually unpredictable and leads to fake exogenous shocks this results seems to be less explainable, even more since the sparsity is not correlated with this result. The reason is that given that 10% of the shocks are a lot larger than average the 10% are mainly real exogenous shocks that will trigger endogenous activity. It should be tested in further development with the 99th percentile.

• **Activity Skewness:** Skewness explains the number of bins with activity larger than average that are therefore hardly predictable. Higher skewness leads to worst predictions in the majority of the cases.

• **Activity Standard Deviation:** The standard deviation is a measure of volatility, and therefor the prediction gives worst results if the standard deviation is large, because the variation between events is larger.

As a summary a table is proposed to understand the advantages of each filtering method. It provides a good way for systematic choice of the optimal method.

<table>
<thead>
<tr>
<th></th>
<th>Auto Regressive</th>
<th>Kalman Filter</th>
<th>Kalman-Lévy Filter</th>
<th>Particle Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Number of Dev</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Large Number of File</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Sparse Projects</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Skewed projects</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Projects with high volatility</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 25: Best method considering the characteristics of each projects, the empty is the neutral meaning no correlation with the performance of each method.

### 7.2 Extensions

#### 7.2.1 Multiple Model Adaptive Estimator

The Multiple Model Adaptive Estimator (MMAE) can hardly be compared with the other models since the memory kernel is adapted all over the process and not fixed by cross-validation, therefore we can see that for both data-set the MMAE outperforms all the others methods. More than providing a optimal filtering method, the MMAE gives a new method for finding the optimal parameters of the memory kernel at a given time.

#### 7.2.2 Developers level predictions

We can see in the Receiver Operating Characteristic graph from figure 22 that using the Identity Matrix gives very good results, it means that developers do not really activate each others but they activate themselves a lot. Meaning that the models used previously for all the developers could be used efficiently with separate developers. However we can observe that taking all the developers together, (as we did in the first part of the project)
lead to relatively bad results. Finally using the evolutive A matrix, meaning that we consider that developers who worked together tends to activate each others, allows to get slightly better true positive rate but returns a globally worst prediction. It can be concluded that people who worked together at one time are not significantly predisposed to cross-excite later on. The building of the A Matrix is based on 3 different methods based on our choices, there is not evidence that finding a better A Matrix from the data would not lead to significantly better results. Therefore, these results have to be taken carefully since it does not mean that there is no correlation between developers developing together and their future activity only that in this form the correlation is not significant.

7.2.3 Files level Predictions

Basics The first conclusion from Figure 23 is that predictions on the developers are a lot more trustworthy than prediction on files. This is specially true for the identity A matrix, the files tends to excite themselves a lot less than developers. If a list of the active developer can easy be done, the results are less obvious for files. If the Identity matrix gives no result better than 0.5 True Positive Rate the evolutive A matrix lead to significantly higher true positive rates with obviously lost in the True Negative Rate. However, in cases the true positives are more interesting than true negatives, the use of the evolutive matrix is recommended. Therefore we can consider that files can be much more clusterized than developers. Files that are developed at the same time are put in same cluster and be cross-excited more often later.

Undirected Neighboring files tend to cross-excite more often, and all the files that are connected taking as cross-excitation weight of the inverse of the cube of their distance returns higher True positive rate with some lost in the true negative.

Directed In Figure 24 we can observe that there is no big difference between taking directed or undirected graph in the sense of area under the curve (AUC), however the undirected graph can allow better true positive rate (TPR) than the identity matrix. At the opposite, the results from the directed graph are not different from the results of the Identity Matrix. We can observe in this graph that using this model one way of dependencies (the Transpose of the matrix) returns better results. If class B call a method in class C then \( D_{B,C} = 1 \) then B will activate later C. However since the inverse is more accurate we can conclude that C excites B more often than the opposite. It leads to the conclusion that people tend to modified the called function first and then later apply this modifications at the higher level which is totally intuitive.

8 Further Developments

One of the limitations of the model we used is the invariable size of the binning. There is no technical issue to change the binning for instance to increase it to a month or so. However, reducing the time bin could be implemented by adding a new probability considering the current situation for instance the day of the week. That would slightly complicate the model but can make it a lot more flexible to include external implications that can be measured as for instance the activity on the communities forums, etc.
Moreover, it would allow the model to be used in many fields where faster predictions are required as the delay in the airline industry, etc. Then, if the results are valid for the larger projects found it should be validated in other fields, the interest would be fields with measurements errors or with measurement that are not fully significant in this case. As an example, emails could be an interesting domain, some of the emails are spams that are sent to a large group of person but will usually not involve any later activity at the opposite of normal emails that usually requires an answer.

Then the multiple model adaptive estimator (MMAE) can be modified to work not with a set of Kalman-Filter but auto-regressive models, it would produce a system that would detect the change of regime without the filtering. Since changes of regimes are one of the points of focus in the economics research [23], this approach, even if not as precise as the point-process, can allow large scale studies on financial data, since the evaluation of the regime is done online. Moreover, for continuing the research on open source software, the MMAE could be developed with other filtering methods that provides better results such as Kalman-Lévy Filter.

Finally the File/Developer level prediction gives in our case an estimation that can’t be reached by linear classification and that is computationally cheap; it therefore can be applied in many other systems. The A Matrix can also be computed by reverse engineering and therefore gives a good insight of the dependencies between agents of an endo/exo system. This model should be applied to more projects however it requires method for an automatic mapping between the Java Classes names and the files names, method that is not straight forward in many cases.

One of the possible extensions of the predictions at a fine level would be to focus on the network of projects given by Maven. It would provide an interesting overview of the cross-activity between the projects as we did between the classes within a project. The relations between the results find at fine and coarse level would give an interesting overview of the very complex network of open source projects and classes.

9 Conclusion

A new method based on linearization of the Hawkes Self-Excited process after a binning of the data has been developed. Transforming the point-process in a Hidden Markov Model provides new opportunities to explore the endogenous/exogenous model with exponential memory kernel. First, it allowed testing three filters to reduce the impact of exogenous shocks that are usually far from the predictions and that do not lead to later endogenous activity. Each filter have some strengths and weaknesses depending on the characteristics of the projects. However the Kalman-Lévy Filter gives better results in a majority of projects. In addition to the filtering, another possible application was the multiple model adaptive estimator (MMAE) that allows multiple regimes and online gives weights to each regime, the results are obviously significantly better than the mono-regime predictions. Moreover, it gives a new way of understanding the behavior of the process at a given time. Finally we created a new classifier based on logistic function and the endo/exo model. The results are surpassing the logistic regression classifier and allow a good understanding on the relations between files and between developers.

These are explorative results of a new method that can be used for any process fol-
lowing the endogenous/exogenous model, it has a strong potential towards systematic prediction of activity in open source software, and as result of software innovation. Moreover, these techniques have the potential to be applied to many other complex adaptive systems, involving complicated networked structures, and large deviations.
References


A  Demonstrations

\[ \phi(k + dt - i) = a \cdot e^{-b(k + dt - i)} \]
\[ = a \cdot e^{-b(k - i) - b \cdot dt} \]
\[ = e^{-b \cdot dt} \cdot a \cdot e^{-b(k - i)} \]
\[ = e^{-b \cdot dt} \cdot \phi(k - i) \]  \hspace{1cm} (36)

\[ N_k = N_k^{exo} + N_k^{endo} \]
\[ N_k^{endo} = \sum_{i<k} \phi(k + dt - i) \]
\[ = \sum_{i<k} \phi(k + dt - i) + \sum_{k<i<k+dt} \phi(k + dt - i) \]
\[ = \left( e^{-b \cdot dt} \cdot \sum_{i<k} \phi(k - i) \right) + N_{k-1} \cdot \phi(k + dt - i) \]
\[ = \left( e^{-b \cdot dt} \cdot N_{k-1}^{endo} \right) + \left( N_k^{exo} + N_k^{endo} \right) \cdot \phi(dt) \]
\[ = \left( e^{-b \cdot dt} \cdot N_{k-1}^{endo} \right) + \left( N_k^{exo} + N_k^{endo} \right) \cdot a \cdot e^{-b \cdot dt} \]
\[ = N_{k-1}^{endo} \left( e^{-b \cdot dt} + a \cdot e^{-b \cdot dt} \right) + N_k^{exo} \left( a \cdot e^{-b \cdot dt} \right) \]  \hspace{1cm} (37)

B  Methods Details

B.1  Kalman Filter

**Prediction Step:**

\[ X_{pred} = FX_{update} + U \]  \hspace{1cm} (38)
\[ P_{pred} = F \cdot P_{update} \cdot F^T + Q \]  \hspace{1cm} (39)

**Update Step:**

\[ y = z - Hx_{pred} \]  \hspace{1cm} (40)
\[ S = H \cdot P_{pred} \cdot H^T + R \]  \hspace{1cm} (41)
\[ K = P_{pred} \cdot H^T \cdot S^{-1} \]  \hspace{1cm} (42)
\[ X_{update} = X_{pred} + K \cdot y \]  \hspace{1cm} (43)
\[ P_{update} = (I - K \cdot H)P_{pred} \]  \hspace{1cm} (44)

Where \( P_{pred} \) and \( P_{update} \) are the covariance of the difference between the true state and respectively the predicted state and the updated state. Finally \( S \) is the covariance of the residuals and \( K \) the optimal Gain (\( X_{update} = K \cdot Z + (1 - K) \cdot X_{pred} \)).
B.2 Kalman Lévy Filter

As for all the filters the process can be decomposed in two steps the prediction step, and the update step, the prediction step is very near to the one from the Kalman filter, with some change to the covariance matrix that is the "tail-covariance" matrix.

The Prediction Step:

\[ X_{\text{pred}} = FX_{\text{update}} \quad (45) \]
\[ B_{\text{pred}} = (F \ast G_{\text{update}})\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \ast C_{\text{update}} \ast (F \ast G_{\text{update}})^T \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} + B_\eta \quad (46) \]

The update step is more complex because it requires to optimize a function. However in two dimensional systems as in the current one it is still a lot faster than particle filter.

The update Step:

\[ H^G = HG_{\text{pred}} \quad (47) \]
\[ B_{\text{update}} = (G_{\text{pred}} - KH^G)\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \ast C_{\text{pred}} \ast (G_{\text{pred}} - KH^G)^T \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} + KG_{\text{pred}} \quad (48) \]
\[ x_{\text{update}} = X_{\text{pred}} + K(z_k - H \ast X_{\text{pred}}) \quad (49) \]
\[ B_{\text{update}} = G_{\text{update}}^{1/2} \ast C_{\text{update}} \ast G_{\text{update}}^{T 1/2} \quad (50) \]

\( G_{\text{update}} \) and \( C_{\text{update}} \) are the eigenvectors and eigenvalues of \( B_{\text{update}} \) and \( K \), the Gain of the filter is obtained by minimizing the trace of \( B_{\text{update}} \)

\[ \frac{\partial}{\partial K_{ij}} B_{i,i} = 0 \]

\[ = H \left( - \sum_{p=1}^{N} (G_{i,p}^{\text{pred}} - \sum_{m=1}^{L} K_{i,m} H_{m,p}^{G} )^{[\mu-1]} \ast G_{j,p}^{\text{pred}} C_{p}^{\text{pred}} \right. \]
\[ + \sum_{q=1}^{L} \sum_{m=1}^{L} (K_{i,m} C_{m,q}^{\eta})^{[\mu-1]} G_{j,q}^{\eta} C_{q}^{\eta} \) \quad (51) \]
B.3 Fine Level Predictions

\[ X_{k+1}^{endo+exo} = X_{k+1}^{endo} \]  
\[ X_{k+1} = \begin{pmatrix} X_{k+1}^{endo} \\ X_{k+1}^{endo+exo} \end{pmatrix} \]  
\[ F^{endo} = I_n * e^{-b \cdot dt} \]  
\[ F^{endo+exo} = c A * e^{-b \cdot dt} * a; \]  
\[ F = \begin{pmatrix} F^{endo} & F^{endo+exo} \\ F^{endo} & F^{endo+exo} \end{pmatrix} \]  
\[ X_{pred} = F * X_{update} \]  
\[ residuals = Z - X_{pred}^{endo+exo} \]  
\[ X_{update} = \begin{pmatrix} X_{pred}^{endo} \\ Z \end{pmatrix} \]

C Projects Details
Figure 26: Maven Projects Characteristics
<table>
<thead>
<tr>
<th>Large Projects</th>
<th>Nb. Commits</th>
<th>Nb. Devs</th>
<th>Nb. Files</th>
<th>Median Commit per dev</th>
<th>Max. Commit Rate nb. weeks</th>
<th>Mean activity</th>
<th>Median activity</th>
<th>peaks</th>
<th>Span</th>
<th>Skewness</th>
<th>Std</th>
<th>kurtosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>sglportal</td>
<td>99733</td>
<td>21</td>
<td>99733</td>
<td>21</td>
<td>2.0</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>1.16</td>
<td>0.71</td>
<td>0.15</td>
<td>2.11</td>
</tr>
<tr>
<td>subversion</td>
<td>90737</td>
<td>21</td>
<td>90737</td>
<td>21</td>
<td>1.0</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>1.16</td>
<td>0.71</td>
<td>0.15</td>
<td>2.11</td>
</tr>
<tr>
<td>autowikibrowser</td>
<td>133277</td>
<td>16</td>
<td>90958</td>
<td>135.5</td>
<td>0.7</td>
<td>2.11</td>
<td>6.19</td>
<td>0.53</td>
<td>0.01</td>
<td>9.31</td>
<td>2744.7</td>
<td></td>
</tr>
<tr>
<td>tvbrowser</td>
<td>127779</td>
<td>21</td>
<td>92166</td>
<td>151</td>
<td>2.27</td>
<td>0.07</td>
<td>6.43</td>
<td>87392</td>
<td>15536.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>marsyas</td>
<td>125237</td>
<td>32</td>
<td>65070</td>
<td>58</td>
<td>1.18</td>
<td>0.56</td>
<td>4.41</td>
<td>1316.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>winmerge</td>
<td>108211</td>
<td>18</td>
<td>67295</td>
<td>335</td>
<td>0.84</td>
<td>0.23</td>
<td>4.45</td>
<td>578.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ledger_smb</td>
<td>104229</td>
<td>7</td>
<td>91136</td>
<td>3096</td>
<td>5.32</td>
<td>0.12</td>
<td>3.59</td>
<td>1415.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clamwin</td>
<td>101908</td>
<td>9</td>
<td>53501</td>
<td>690</td>
<td>0.85</td>
<td>0.14</td>
<td>2.35</td>
<td>2828.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kmkey</td>
<td>50051</td>
<td>3</td>
<td>40985</td>
<td>4200</td>
<td>4.15</td>
<td>0.07</td>
<td>2.46</td>
<td>807.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>romaframework</td>
<td>49906</td>
<td>16</td>
<td>21505</td>
<td>349.5</td>
<td>1.02</td>
<td>0.05</td>
<td>4.13</td>
<td>460.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rubyeclipse</td>
<td>45718</td>
<td>14</td>
<td>28914</td>
<td>166</td>
<td>1.34</td>
<td>0.36</td>
<td>5.1</td>
<td>449.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gdcm</td>
<td>43443</td>
<td>7</td>
<td>19565</td>
<td>24</td>
<td>1.42</td>
<td>0.13</td>
<td>3.32</td>
<td>38468</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>git</td>
<td>38319</td>
<td>847</td>
<td>2766</td>
<td>4</td>
<td>1.54</td>
<td>0.02</td>
<td>1.54</td>
<td>71.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>servicelanguage</td>
<td>37706</td>
<td>5</td>
<td>15696</td>
<td>579</td>
<td>2.37</td>
<td>0.52</td>
<td>6.23</td>
<td>1071.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>176184</td>
<td>57.78</td>
<td>97945.3</td>
<td>17174</td>
<td>1.34</td>
<td>0.14</td>
<td>5.9809</td>
<td>2334.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 27: Large Projects Characteristics