ETHZ (SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH)

CHAIR OF ENTREPREUNARIAL RISKS

Factor of Success in Open Source Software

Master Thesis

Thomas Frendo

07/10/2009

Supervisor : Prof. D. Sornette

Tutor: T. Maillart

In collaboration with

Chair of Strategic Management and Innovation (ETHZ)
Prof. G. von Krogh

Software Evolution and Architecture Lab (University of Zürich)
Prof. H. Gall

ABSTRACT

In this master thesis, we present state of the art work on Open Source Software (OSS) and propose an approach to understand cooperation and efficiency of developers in the open source community using waiting time distributions of commit of developers. We apply this methodology on Eclipse and Mozilla, two widely recognized open source projects with two different community models and missions. Our data source is essentially composed of the Concurrent Versioning System (CVS) log of activities that is freely accessible on their Eclipse and Mozilla website.

Each large OSS is composed of multiple sub-projects. We found out that the aggregation of the sub-project level is presenting power law distributions of waiting times with a coefficient $\mu = 1.54$ and $\mu = 1.33$ over 3 and 4 decades for Eclipse and Mozilla. Secondly, for the same projects, we found out the aggregation of the developer level is presenting power law distributions of waiting times with a coefficient $\mu = 1.45$ and $\mu = 1.07$ over 4 and 2 decades. The difference in coefficient could be one measure of collaboration impact in waiting times distribution.

Secondly most of the sub-projects are presenting power law distributions for their waiting times between activities with a coefficient varying from $\mu = 0.5$ and $\mu = 3.5$, both on Mozilla and Eclipse. We differentiated between the waiting times of debugging activities – that we consider critical and contributing to reliability – and the waiting times of non debugging activities – that we consider as strategic and contributing to creativity. Based on the change of regime in the ccdf in power laws at $\mu = 1$, we propose a methodology – the strategic critical analysis – in order to distinguish strategic and critical sub-projects. We found out that the OSS community emphasizes either strategy or criticality depending on the project.

Our results shows that although Mozilla projects overall are less creative ($\mu_{creativity} = 1.23$) as reliable ($\mu_{reliability} = 1.36$), and there are many more projects oriented towards reliability than creativity. On the contrary Eclipse look more balanced approach and do not favor one over ($\mu_{creativity} = 1.43$, $\mu_{reliability} = 1.49$) the other.

This study is easily extendable using python script furnished along with the CD as CVS/SVN or similar collaborative tool to exchange code provides similar log. Therefore, we propose to industrialize the Strategic Critical Analysis for future research, and use it to assess organizational capabilities of any organization in software industry.

TABLE OF CONTENTS

Abstract ..... *- 2 -*

1. Introduction ..... *- 5 -*

2. Background ..... *- 6 -*

3. Overview of data ..... *- 8 -*

3.1 Data structure ..... - 8 -

3.2 The Mozilla Project ..... - 9 -

3.3 The Eclipse Project ..... - 10 -

4. Developers' experience and dynamics of collaboration ..... *- 13 -*

4.1 Developer's Experience ..... - 13 -

4.2 Dynamics of Cooperation at File Level ..... - 15 -

5. Waiting times between actions (in OSS) ..... *- 17 -*

5.1 Definition of Waiting Time for CVS ..... - 17 -

5.2 Coarse Grain approach: Space Definition ..... - 18 -

5.3 Coarse Selection of Grain approach: Selection of the right Zoom level ..... - 19 -

5.4 Measurement Methods ..... - 20 -

5.5 Waiting times of Mozilla and Eclipse Developers ..... - 22 -

5.6 Waiting Times for Mozilla and Eclipse Projects ..... - 23 -

6. Strategic critical analysis ..... *- 27 -*

6.1 Definition ..... - 27 -

6.2 Methodology ..... - 28 -

6.3 Mozilla and Eclipse screened by the SCA ..... - 30 -

7. Limitations ..... *- 34 -*

8. Conclusion and Outlook ..... *- 36 -*

References ..... *- 38 -*

1. INTRODUCTION

Business and political leaders recognize and are using Open Source Software (OSS). OSS is creating competition to commercial standards and therefore reduces the possibility for commercial monopolies. Scott McNealy, a co-founder of Sun Microsystems said recently that It's intuitively obvious open source is more cost effective and productive than proprietary software (1). Even politicians are using it in their day to day life. The French parliament is equipped with Linux since June 2007 (2). The French Gendarmerie decided on strict open standards IT policy since 2002, which led to an IT budget reduction by 70% in 2009 and claim no negative impacts on IT standards (3). Individuals and companies are also widely using OSS standards. Mozilla Firefox has close to 20% market share of the web-browser market (4). Apache has 46.62% of Market Share for Top Server across all domains since September 2009 (5).

How does an open source succeed? What are the underlying factors? Can we learn from their success stories? We think that determining factor of success in open source software (OSS) is a first step towards quantitatively understanding the success of new ventures. Our hypothesis is that some external factors (market, utility) as well as internal factors (organization, distribution of work) contribute to the success of an Open Source project. The phenomena is called Endo/Exo framework (13). In this thesis, we aim to understand some of the internal factors by looking closely at developers' behavior.

The next section of the paper presents the background of study, the third section presents an overview of the data used, the third sections presents some preliminary results on developers' experience and on the dynamics of cooperation. The fifth section presents the waiting times between activities approach in OSS projects. The sixth section presents the strategic critical analysis that helps to distinguish between critical modules and strategic modules and its results with Eclipse and Mozilla. Finally, the last two sections will discuss the limitation of the study and possible outlooks.

2. BACKGROUND

OSS community has been once described as a “bazaar” of source code and projects with no real organization by opposition to the “cathedral” of commercial (closed source) software (6). OSS is indeed proposing a very different model from usual Commercial Software. Commercial Software developers[1] will usually release a binary version of the program which makes it difficult to read and understand for programmers (7). Open Source Software is freely accessible for all, and in most cases programmers wish to enable others to understand, update and modify their software. Therefore, they provide the software with its source code (8)(9).

Interestingly, OSS projects seem to expend no effort to encourage contributing over free riding. Anyone is free to download code or seek help from project websites, and no apparent form of moral pressure is applied to make a compensating contribution. Even more, such projects typically engage in no active recruiting beyond simply posting their intended goals and access address on a general public website customarily used for this purpose (10). Even more, the Mozilla community is very protective towards possible contributors. Their webpage entitled “becoming a Mozilla developer” shows in bold and large font a “STOP” notice with the following text: “Have you written enough patches for Mozilla so that the patch reviewers have a good feel for your work and so that it's clear you understand the review process? If you haven't, you'll want to do that -- people will want a feel for you and your code before vouching for you. If you have, read on...” (11).

These warnings and unwelcoming messages do not really impact their popularity in applicant numbers. We could even argue that it attracts an elitist group of developers sharing certain community principles, the same passions, knowledge and hobbies, similar to hacker thinking (10). In fact, according to a study conducted by Ghosh et al., 49% of developers tends to feel that working in the proprietary software field can be very boring vs. 13.9% for OSS, and for 76.1% of them working in proprietary software field is usually associated with time pressure vs. 2.4% for OSS. Moreover, they tend to think that working in proprietary software is much more efficient for only 12.1% of them compared to 42.6% for OSS. Finally 78.9% feels that working in OSS in joyful vs. 0.4% for proprietary software (12).

[1] Developers, Committers and Authors will be used to describe the same group of people: developers.

The Mozilla project its one of the best examples of this popularity, regrouping a cumulative number of 800 code committers in 2007 (see next section).

Empirical studies based on certification standards and conducted by IBM shows than OSS are not less secure than proprietary systems. However, they claim that having source code open enables technical personnel to understand the immediate threat. In addition, OSS developers have the ability to analyze how previous systems were constructed and build on them lowering their cost (14). They claim that once a critical mass of users has formed, the systemic effect will make the software meet and exceed the security and reliability metrics of their proprietary counterparts – at a much reduced cost (15).

Other quantitative studies have been done in software engineering to evaluate quality of code source according to code design and relations. Baxter (16) proposes a methodology to analyze java code based on human editable aspect of an application's construction.

Open source projects are composed of a self generated system where demands and needs are auto-regulated by the interaction between tasks and developers. Distribution of waiting time of human response has been found in many situations to be a power law. This power law phenomenon has been documented quantitatively in many researches. For example: the time intervals between consecutive e-mails sent by a single user and time delays for e-mail replies (17); the waiting time between receipt and response in the correspondence of Darwin and Einstein (18); the waiting time associated with other human check-in patterns which extend to web browsing, library visits and stock trading (19).

Oliveira et al. (20) proposed a minimal queuing model of human dynamics taking into account human interactions. The coarse-grained version of the model allowed them to observe that the inter-event[2] distribution of interacting tasks exhibit the scaling exponents $\mu = 2$, 3/2 and a series of numerable values between 3/2 and 1.

However not many studies have been done with empirical data of OSS projects concerning this waiting time phenomena. In this report, we will try to understand human interactions in OSS projects using waiting times between development related events.

[2] Later on we will assimilate inter-event and waiting times as the same measure.

3. OVERVIEW OF DATA

3.1 Data structure

Our data set was strictly contained in their CVS log and their links to the Bugzilla system for Eclipse and Mozilla.

Concurrent Versions System (CVS) is a free software revision control system. Version control system software keeps track of all work and all changes in a set of files, and allows several developers (potentially widely separated in space and/or time) to collaborate. (21)

For example, several developers may work on the same project concurrently, each one editing files within their own "working copy" of the project, and sending (or checking in) their modifications to the server. To avoid the possibility of people stepping on each other's toes, the server will only accept changes made to the most recent version of a file. Developers are therefore expected to keep their working copy up-to-date by incorporating other people's changes on a regular basis. This task is mostly handled automatically by the CVS client, requiring manual intervention only when a conflict arises between a checked-in modification and the yet-unchecked local version of a file.

The Bugzilla system is an OSS developed by the Mozilla community that helps to track and manage debugging issues. It uses a system of bug reporting and issue tracking. When correctly used with CVS, each check-in related to a bug is linked to a Bugzilla entry.

The CVS log allows one to observe the past exchange of code between the developers. Each line in the CVS log is attributed to one specific file at a specific time. If a committer check in multiple files changed, the CVS system will create different lines. The CVS log we used includes the following fields:

- CVS check-in[3] date and time
- Email of developer that we use at its ID
- File name and path
- Number of added line and number of deleted line
- Use of the entry (bug or no)

[3] Later on, we will assimilate “check-in”, “activity” and “event” by abuse of language. These three words will always be assimilated in the context of CVS log data.

The CVS/SVN standards are widely used in open source projects and therefore all the work presented in this report can be easily replicated using the same python scripts furnished with the CD provided along with this report.

Mozilla CVS log is covering all activities from 1998-03-27 to 2007-07-16 and the Eclipse CVS log is covering the period all activities from 2001-04-28 to 2009-05-14 for the Eclipse main modules (Galileo)

3.2 The Mozilla Project

The development of Mozilla was initiated by Netscape Communications Corporation, before their acquisition by AOL.

According to Wikipedia, The Mozilla Application Suite is a cross-platform integrated Internet suite.(22) Mozilla.com themselves claims that the Mozilla project is a global community of people who believe that openness, innovation, and opportunity are key to the continued health of the Internet. Since 1998, Mozilla worked to ensure that the Internet is developed in a way that benefits everyone. (23)

It is based on the source code of Netscape Communicator. The development was spearheaded by the Mozilla Organization from 1998 to 2003, and by the Mozilla Foundation since 2003. According to the Mozilla development roadmap published on April 2, 2003, the Mozilla Organization indeed planned to focus development efforts on the new standalone applications: Firefox and Thunderbird.

The Mozilla Suite is composed of several main programs: Navigator, Communicator, a web page developer, an IRC client and an electronic address book and many others...

Figure 1 shows developers' activity within Mozilla project with the time. The abscissa is the time in year and the ordinate represents the committers' ID organized by the date of their first check-in in the project. Each point represents a CVS check-in. We can visually see the rate of arrival of new committers by looking at the slope and we can see as well the lifetime of committers. There are 3 phases of immigration.

1. The migration of the Netscape Navigator to Mozilla takes place at the beginning of the project. A very high rate of committers are joining and leaving the project. We can suppose that these accounts have been created for migration purpose.

2. A stabilization phase occurs between 1999 and 2003. Mozilla is finding its place in the Open Source Community. A moderate rate of new committers can be seen.
3. From 2003 forward (after day 1500 on the figure), the Mozilla community decided to focus on their two main products, Firefox and Thunderbird, and the rate at which the committers are joining is decreasing. Additionally, committers from the generation before 2003 are leaving the project. Can this change of staffing can be explained by the change of strategy? We will leave this question open for future research.

Additionally, we can observe certain clusters of activity. In phase 3, we notice a peak of activity for all developers, even those who seemed to not be active anymore. It is visible by a straight vertical line around day 2900- 2950. We can also see developers coming back after a long leave (developers' id between 100 and 500). Finally we can guess that some of the developers are leaving the project early when looking at the quickly decreasing density on the upper border.

3.3 The Eclipse Project

The Eclipse Project was originally created by IBM in November 2001 and supported by a consortium of software vendors. The Eclipse Foundation was created in January 2004 as an independent not-for-profit corporation to act as the steward of the Eclipse community. The independent not-for-profit corporation was created to allow a vendor neutral and open, transparent community to be established around Eclipse. Today, the Eclipse community consists of individuals and organizations from a cross section of the software industry.(24)

The Eclipse Foundation manages the IT infrastructure for the Eclipse open source community, including CVS/SVN code repositories, Bugzilla databases, development oriented mailing lists and newsgroups, download site and web site. The infrastructure is designed to provide reliable and scalable service for the committers developing the Eclipse technology and the consumers who use the technology.

Interestingly, Eclipse is widely used in the professional world by a large variety of users from developers to simple business analysts. Derived applications are creating the standard in this open source community. Its plug-in architecture makes it easy for software companies to use its main framework and to build and add-up functionalities in order to sell the tool.

Figure 1.b. shows the Eclipse project activity. Even if less obvious than in Mozilla, we can see 3 different major phases with a decreasing rate of immigration. The first one is starting from day 0 to day 100. The second one is starting from day 100 to day 1600. Finally, the last one is from day 1600 to day 3000. We can guess that after a short phase of set-up of the project where many hands are needed, the projects reach a higher maturity level that either makes the OSS less attractive to join for developers or makes the OSS more restrictive about the number of people joining. It is also clear in the case of Eclipse that some developers are joining the project for a very limited time, even less than 100 days. We can also observe the full commitment of certain champions that stay active during the entire measured period of the project. Also visible is the extreme density of points for certain groups of committers that commit changes much more frequently than others. These heuristic dynamics will be later analyzed on section 3.

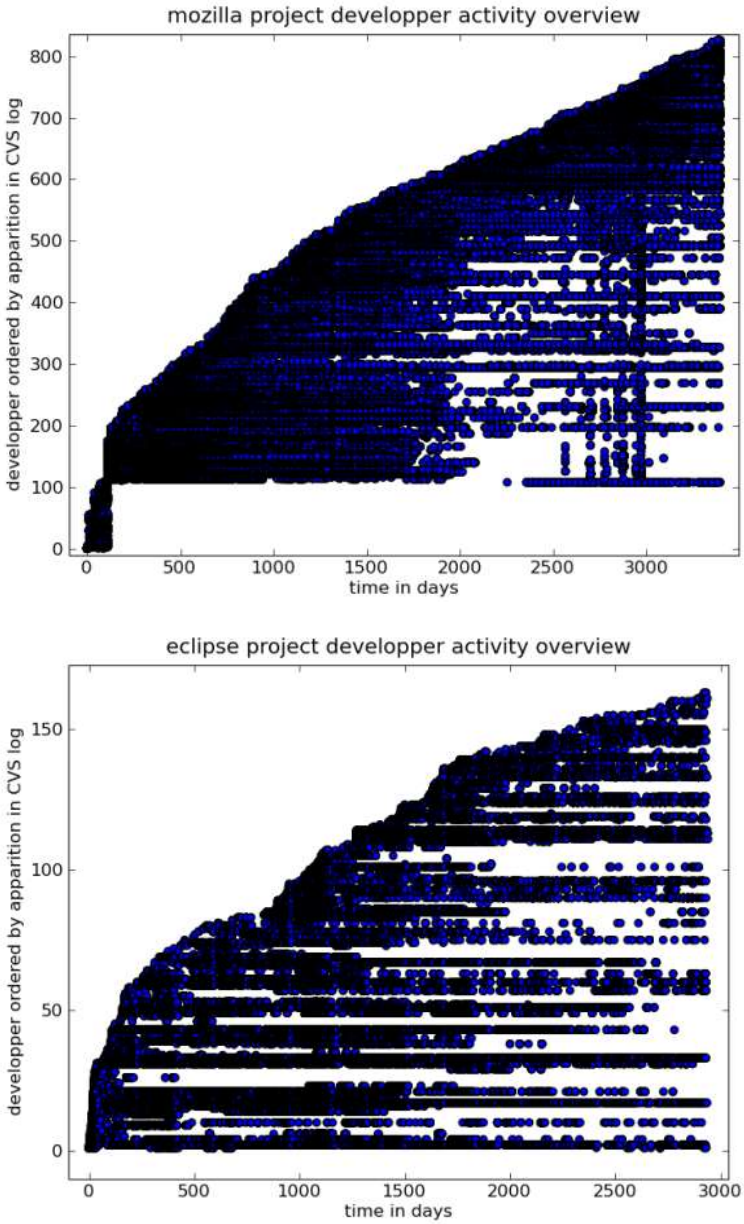

Figure 1: Mozilla / Eclipse Project Developers check-ins overview on Mozilla / Eclipse project overall. The left figure represents the Mozilla project and the right figure represents the Eclipse project. The abscissa represents the time in days; the ordinate represents the developers id ordered by first apparition in the CVS log. Each point represents an event (check-in). Note that when there is a big concentration of point in a region, the color of the points becomes black.

4. DEVELOPERS' EXPERIENCE AND DYNAMICS OF COLLABORATION

4.1 Developer's Experience

As we have seen on the previous section, the Mozilla developer group is very wide. One of our interests was to find how we could determine experience of developers. From the CVS log, we considered three possible metrics: the lifetime of the developer, the number of lines modified, and finally the number of check-ins by developers. We made the tests both on Eclipse and Mozilla. Interestingly the distributions of these metrics are very similar, as we can see on the next page. The lifetime of a developer in Eclipse and Mozilla seems to follow an exponential law, with a clear cut-off in the fat tail most probably due to the size of the system. Both number of lines modified and number of commits seems to follow a stretch exponential with a dragon phenomena.

We calculated the correlation of the three metrics for each project using the Spearman Rho Calculation as these were not Gaussian. The Spearman Rho is adequate for non Gaussian and heavy tail distributions. It is a correlation calculation based not on the value of the metrics, but on their rank. The results presented in table 1 show that the three distributions are correlated to each other for both projects. In the Mozilla project, check-in number per developer is correlated to lines modified per developer at 0.95 (Eclipse: 0.84) and to lifetime per developer at 0.76 (Eclipse: 0.74). The number of lines modified is correlated to lifetime at 0.71 (0.63). These numbers means that we can take any of those metrics to assess what we assume to be experience.

We have plotted the conditional distributions of all possible combination between the distributions per developers, check-in number, lines modified and lifetime. We found no significant results except for this distribution of lifetime depending on check-ins number. Interestingly, we can see the following result on Figure 2: the more developers will check-in, the more likely the distribution of their lifetime will be Gaussian.

Mozilla (per dev.)	Check-in Number	Lines Modified	Lifetime
Check-in number	1	0.95	0.76
Lines Modified		1	0.71
Lifetime			1

Eclipse (per dev.)	Check-in Number	Lines Modified	Lifetime
Check-in number	1	0.84	0.74
Lines Modified		1	0.63
Lifetime			1

Table 1: the left table represents a correlation table between check-in number per developer, lines modified per developer, lifetime of developer.

In terms of management, this measure is significant – it means that after a certain threshold of check-in, your employee lifetime will follow a Gaussian rule and you can therefore apply widely known statistics using the mean and variance in order to determine your needed regeneration rate in the pool. It also shows that if your developers are not submitting code frequently, these laws do not apply and most likely a longer test period is needed.

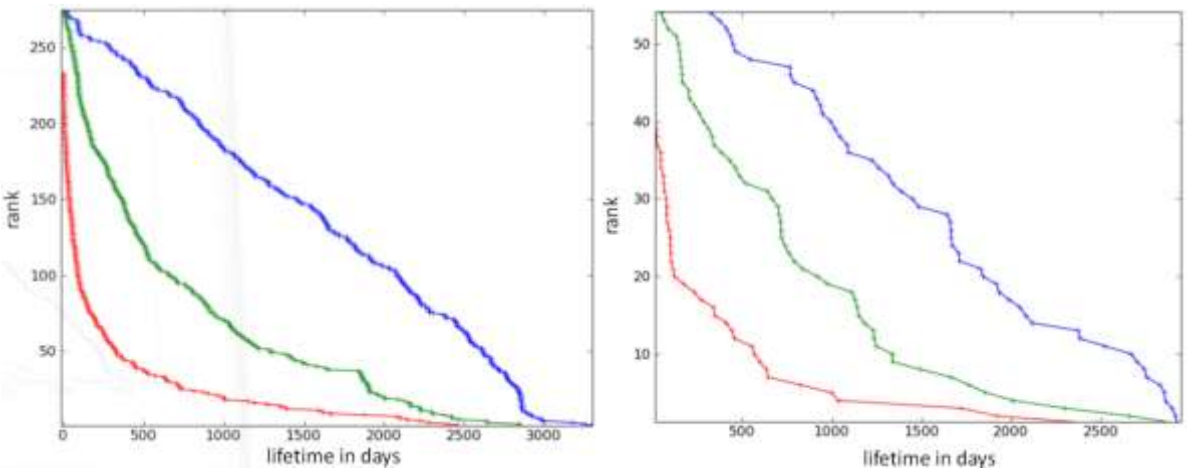

Figure 2: conditional distributions of lifetime depending on check-ins number (Left figure, Mozilla, Right Figure, Eclipse). The abscissa represents the time in days. The ordinate represents the rank. Mozilla Legend: blue: lifetime | 541 < check-in number < 3648; green: lifetime | 79 < check-in number < 539; red: lifetime | 2 < check-in number < 78. Eclipse Legend: blue: lifetime | 1318 < check-in number < 2620; green: lifetime | 164 < check-in number < 1312; red: lifetime | 2 < check-in number < 163

4.2 Dynamics of Cooperation at File Level

We look first at a micro-level and build our intuition based on our preliminary results. We defined our micro-level here as being the file level. For each file, we wanted to observe the activities of different developers with time, the activity type and its intensity. We discovered evidence of cooperation between differently experienced developers; burst of activity with many people contributing, long memory process (i.e. a contribution triggers a decreasing flow of contribution by others, etc.).

Figure 3 presents the check-in with time of developers depending on their check-in type (bug related or non bug related) and their check-in intensity. We consider that check-in intensity is represented by the number of lines the committers added or deleted. The red squares represent the bug check-ins, and their size represents the intensity. The blue bubbles represent the non-bug related check-ins and their size represents the intensity. The Abscissa represents the time in days and the Ordinate represents the committers ordered by time of entry in the CVS log that we assume being time of entry in the project.

In the following plot, we can observe a clear leadership of the new developers at the beginning which forms a real cluster of check-ins of 1700 days. We can observe a second cluster of check-ins after 1700 days where one experienced committer and new comers are taking the lead[4] of the project. Next example is the Calendar.js file. We observe that the leadership of the project is assumed by developer number 510. This developer is apparently a newcomer in the project when referring to Figure 1. During 2 years, this developer will control all the development of the project and develops the main functionalities of the application. After this period we see a mix of older developers and younger developers joining the project and adding rather small improvements, and create a burst of check-ins where many more new developers are joining the project.

Although those examples are fascinating, we generated these story plots for more than 132’471 files present in our set of data. Note a filter is necessary when generating story plots as around 2/3 of the files are presenting less than 10 activities over the 10 years Mozilla was measured. Making qualitative hypothesis on each of them could take years and with no

[4] By lead here we understand taking share of the majority of the intensity.

certainty on the usability of the result. Specific groupings of folders and files might be much more relevant as they have been designed for the same goal.

We therefore end our exploration of the micro-world the CVS log can offer us. In the following section we will use quantitative methods to highlight the dynamics seen in this heuristic approach. We will build our work based on existing work on waiting times between events.

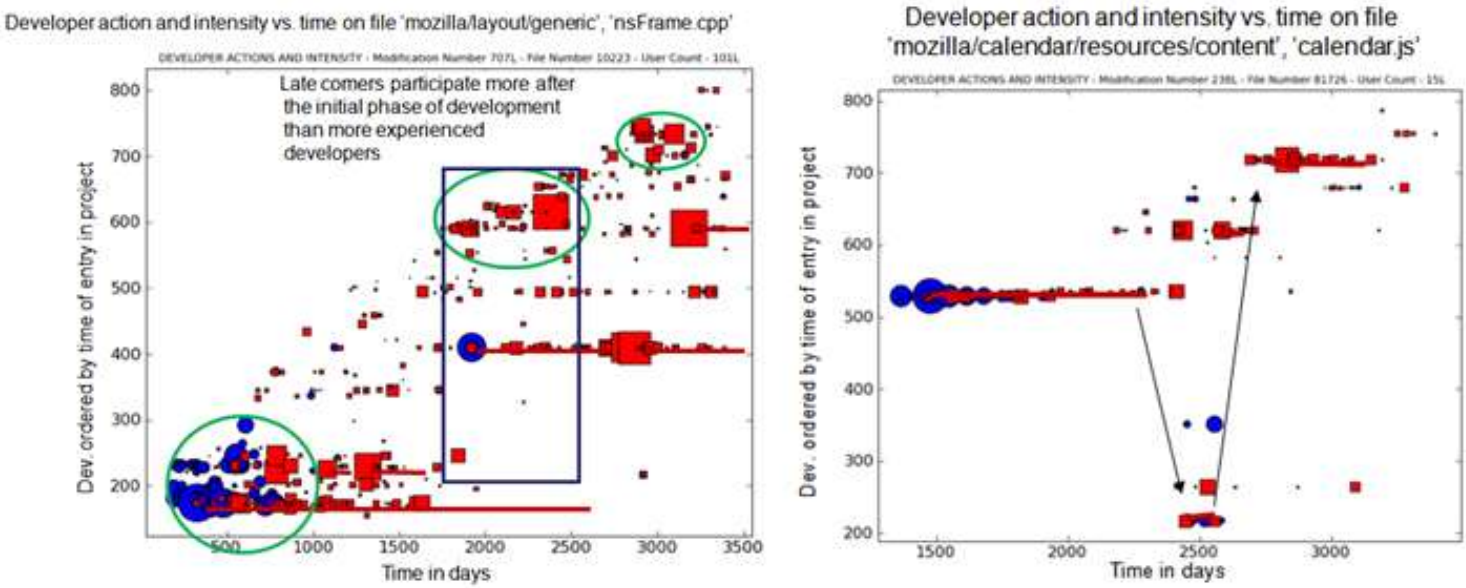

Figure 3: check-in of developers in two files in Motilla. The abscissa represents the time in day and the ordinate represents the developer id order by date of entry in the CVS log. Each point represents a check-in. The red points are check-ins linked to bugzilla and the blue points are check-ins non-linked to bugzilla. The size of the points represents the number of added lines + deleted lines for each check-in.

5. WAITING TIMES BETWEEN ACTIONS (IN OSS)

5.1 Definition of Waiting Time for CVS

Each line of the CVS log represents a check-in and can be decomposed on bug related check-ins and non bug related check-ins. If the line includes an html link to the Bugzilla system in the “bug” column, then we consider this data as being related to debugging. If there is no HTML link related to the bug check-in, then we consider it as being non-debugging related. Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware thus making it behave as expected.(25)

We defined the waiting time as the time between two consecutives check-ins according to their category. Assume there are N check-ins for a specific file, M related to debugging, and N-M related to non-debugging. We will consider as waiting times the three following vectors: The first vector dt_events_v will include N-1 points and represents all the times between two consecutive events; the second vector dt_bug_v will include M-1 points and represents all the times between two consecutive events related to debugging; Finally the third vector dt_nonbug_v will include N-M-1 points and represents all the waiting time between two consecutive events related to non-debugging.

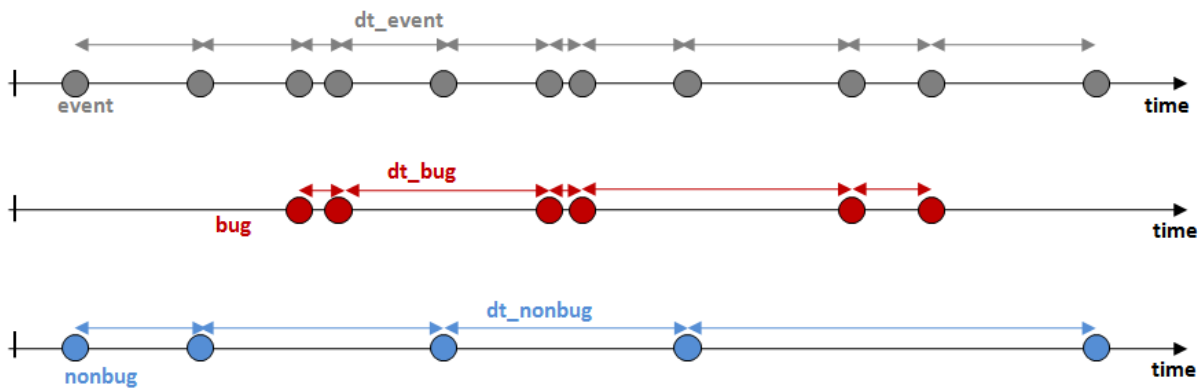

Figure 4: the time between two events is the difference between the next event and the current event. dt_events_v groups all dt_event. An event is either a debug event or non-debug event.

5.2 Coarse Grain approach: Space Definition

We need to find the best zoom level in order to understand cooperation in OSS. The right zoom level should provide results with patterns of activity and of cooperation. Our zoom space is composed of the Mozilla sub-folders.

We used a script that counts the "/". We defined the zoom level 1 as containing 1 "/". The zoom level 1 can be seen as the root folder. The zoom level 2 contain 2 "/". It is the first subfolder view. Figure 5 shows a graphical representation of the different zoom level. Note that each different zoom level will include main folders. Figure 6 presents the number of main folders for each zoom level repository. Starting from 0 at level 1, it reaches 99 at level 2, around 900 at level 3 and goes beyond 2000 at level 4. It is logical to see that the number of main folders start to decrease after level 5 as we can assume that programmers will try to reduce as much as possible the depth of their working folder in order to find the documents and code easily.

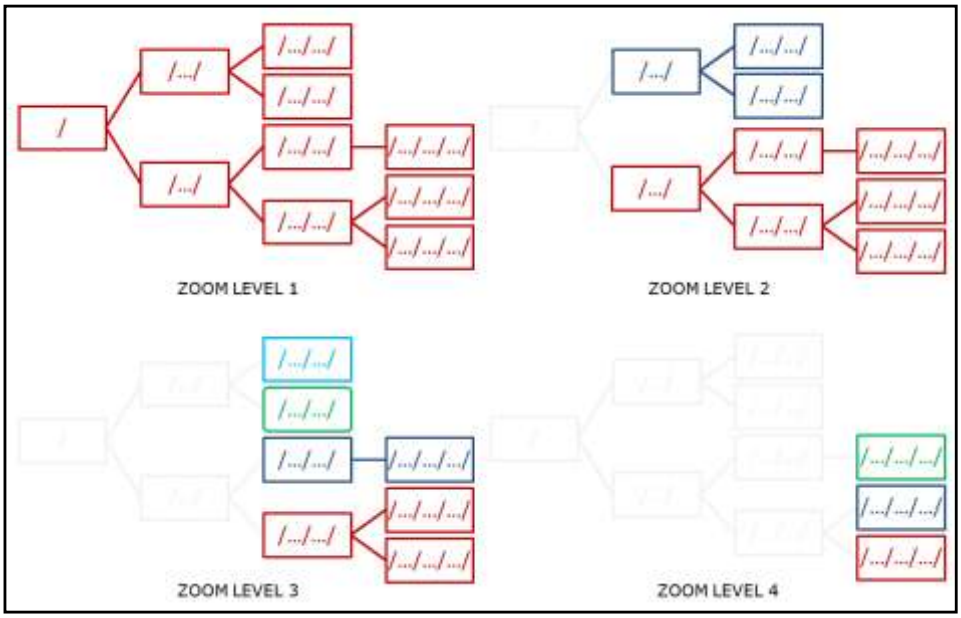

Figure 5 : Example of characterization of zoom level. We look at the individual main folder distribution of check-ins but also at the union of these main folders that we call Aggregated view. i.e. Aggregated view of zoom level 1 (AV1) will be AV1 = {view red} U {view blue}. AV2 = {view cyan} U {view green} U {view blue} U {view red}.

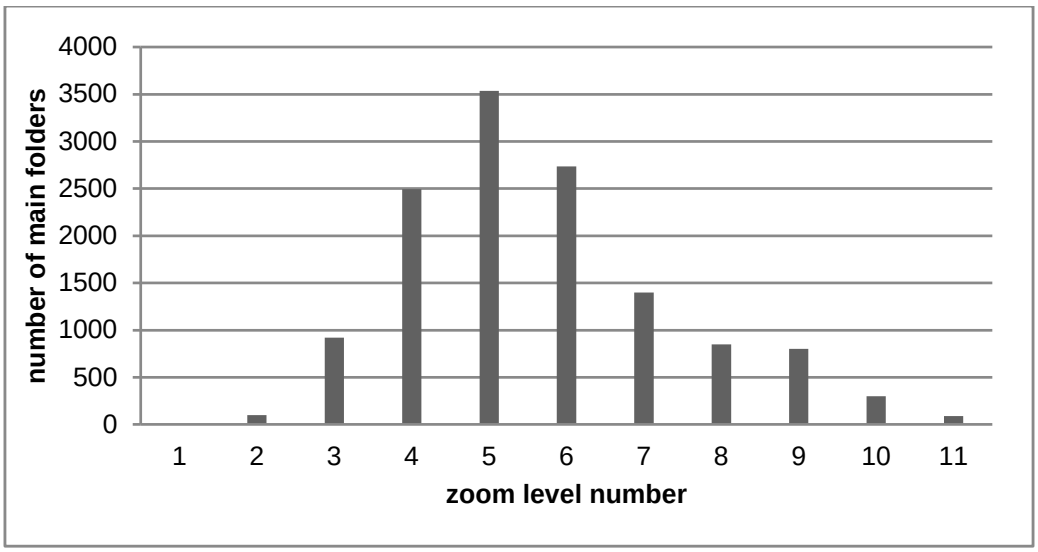

Figure 6: bar chart showing the number of main folders for each zoom level in Mozilla. The abscissa shows the zoom level and the ordinate shows the number of main folders.

5.3 Coarse Selection of Grain approach: Selection of the right Zoom level

We analyzed the distribution of the waiting times dt_event, dt_bug and dt_nonbug in our coarse grain space, i.e. for different zoom level. Figure 7 presents the results of our analysis on the different levels. Interestingly the more we increase the zoom level the more the nature of the distribution go towards an exponential or even Gaussian. At zoom level 2, the distribution of dt_event seems to be a power law, with a finite size effect after 1000 days. Zoom level 3 is showing the same behavior but contains close to 1000 folders and the level 2 is composed of 100 main folders. One of them is the Mozilla directory that contains different project files. Among the level 3, the sub-folders of /Mozilla/ directory represent 87% of the overall folders. From Mozilla documentation, the project based view of Mozilla is composed of different sub-folders and sub-sub-folders of Mozilla.

Considering those results and the fact that the zoom level 2 is close to the hierarchical structure of the projects, in addition to the fact that the number of main folder to analyze is limited, we will consider level 2 as representing the project level view of the Mozilla project. We can therefore claim that the interesting zoom level to understand human responses in

open source is most probably the project level, and in our two cases this holds to be true. The results hold for dt_bug and dt_bug and Table 3 presents the full results.

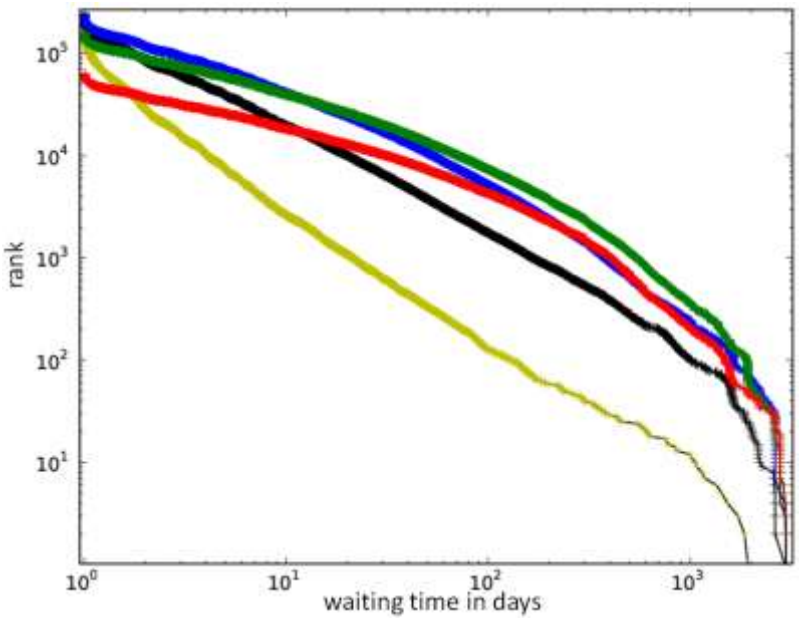

Figure 7 : distribution of dt-check-in at zoom level (ZL) 2, 3… 10 for Mozilla. {gold = ZL2; black = ZL3; blue = ZL4; green = ZL5; red = ZL6 } The abscissa is the waiting time in number of days and the ordinate represents the rank. dt_event is following a power law for the zoom level 2 of 4 decades and for the zoom level 3 over 3 decades. The bootstrapping of the distribution ZL2 is visible in the figure below.

5.4 Measurement Methods

The characterization of power laws is complicated by the large fluctuations that occur in the fat tail of the distribution -- the part of the distribution representing large but rare events -- and by the difficulty of identifying the range over which power-law behavior holds (26).

Our approach combines a semi-automatic maximum-likelihood fitting methods with a bootstrapping. The maximum likelihood estimator (MLE) will assume the distribution to be a power law and will estimate its coefficient given a certain cut-off. Then validate the results by a bootstrapping of the given distribution using the parameters give by the maximum likelihood estimator. The bootstrap method is a computer-based method for rejecting with a certain confidence interval the null hypothesis that is the sample cannot be the random generation of a power law given its exponent. This technique allows estimation of the sample distribution of almost any statistic using only very simple methods. (27). Figure 8 : shows the

methodology used to bootstrap our sample of power laws. We generated 100 power laws using the MLE method (cross in cyan) and we created 40 log-bins that split the figure on 40 different zones (here the bins were added manually). For each zone we selected the 5th, 95th and 50th percentile based on the ordinate value and placed them on the center of the bin (abscissa). 5th, 95th and 50th percentile are respectively the, lower red line, the upper red line and the yellow line.

If our distribution is between the red lines, then, we cannot reject that the distribution follows a power law distribution of parameter μ and cutoff. Additionally we calculate also the confidence interval using the standard deviation and the 95th percentile of μ.

Finally, in order to do the Analysis, we developed a small tool called BSPWLAW (Bootstrapping Power Law) that combined the two techniques in a friendly interface. Refer to Figure 9, Figure 11 and Figure 12 that present as well screenshot of the application. The application allows loading an array in a text file in order to fit it. The user has the possibility to select the cut-off by clicking on a slide bar. The exponent μ will then be automatically by MLE and the user will be able to either refine the cut-off or visually refine the exponent μ. The statistics containing the estimated exponent of the distribution, the exponent of the 95th percentile, the cut-off and the standard deviation are then saved in a text file that can be reused later. The image is saved in EPS format.

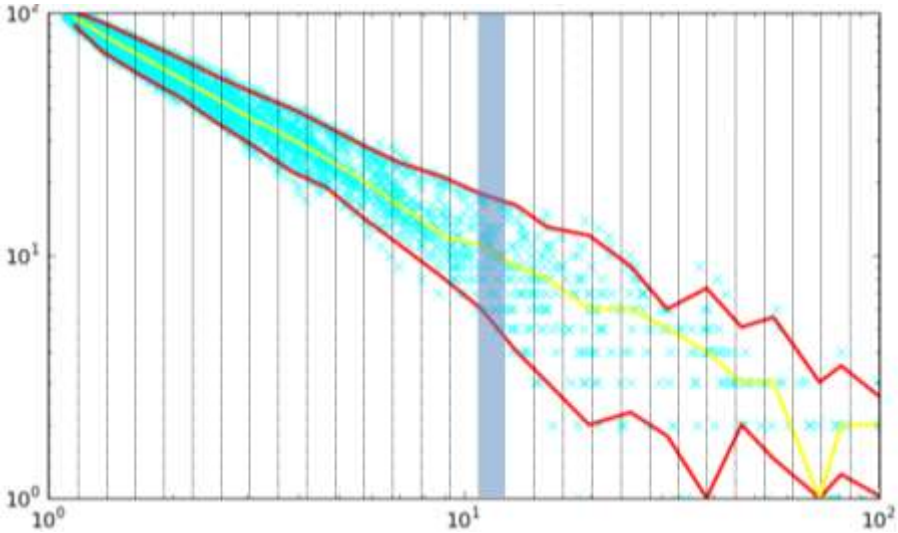

Figure 8 : shows the methodology used to bootstrap our sample of power laws. We generated 100 power laws using the MLE method (cross in cyan) and we created 40 log-bins that split the figure on 40 different zones (here the bins were added manually). For each zone we selected the 5th, 95th and 50th

percentile based on the ordinate value and placed them on the center of the bin (abscissa). 5th, 95th and 50th percentile are respectively the, lower red line, the upper red line and the yellow line.

5.5 Waiting times of Mozilla and Eclipse Developers

Figure 9 presents the Eclipse and Mozilla waiting time distributions for all developers (Aggregation of waiting time of each developer for each project). We observe a clear power law, confirmed by bootstrapping, at the project level in Eclipse and Mozilla, with respective coefficient 1.41 and 1.10 on 3 and 2 decades. Note that for the Mozilla project, we observe a finite size effect after a waiting time of 1000 days.

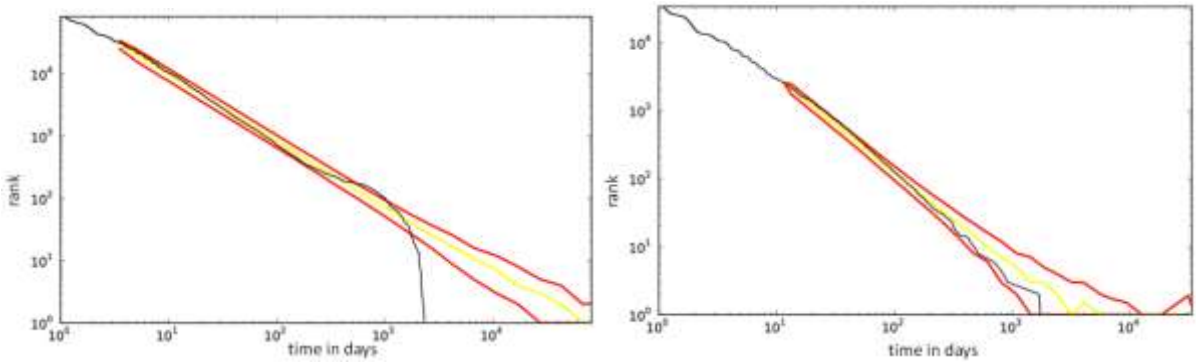

Figure 9 : represents the distribution of the aggregation of waiting times of all developers in each project (Left Mozilla, Right Eclipse) and their bootstrap. The abscissa represents the waiting time in days and the ordinate is the rank of each point. The cut-off and mu are visible at the bottom of the chart.

Mozilla	μ	Cutoff	95th
dt_event	1.06	3.1	1.06 ± 0.01
dt_bug	1.04	5.5	1.04 ± 0.02
dt_nonbug	0.98	5	0.98 ± 0.02

Eclipse	μ	Cutoff	95th
dt_event	1.41	11.47	1.41 ± 0.05
dt_bug	1.24	12.4	1.24 ± 0.05
dt_nonbug	1.47	19	1.47 ± 0.06

Table 2 : results of bootstrapping of the distributions of dt_event, dt_bug, dt_nonbug for Mozilla, Eclipse at the aggregated developer view.

Figure 12 shows some examples of distribution the waiting times of Mozilla and Eclipse Developers at the individual level. We observed a large variety of coefficients for the power

laws starting from μ = 0.5 to μ = 2.0, on both Mozilla and Eclipse. Our short study show that we find also exponent with μ = 1 and μ = 1.5 for some developers.

These results show that there is a wide variety of power law exponent in the distribution of waiting times between actions. These depend on developer individualities but also – depending on the projects – on the different tasks performed.

5.6 Waiting Times for Mozilla and Eclipse Projects

The Eclipse hierarchical folder structure is much cleaner than Mozilla's. Therefore, we use the project level of Eclipse to define the sub-project level of this analysis. In our analysis, Mozilla included 100 projects numbered from 0 to 99 and ordered by date of apparition. Eclipse includes 24 projects. Figure 10 presents the Eclipse and Mozilla waiting time distributions at the project level. We observe a clear power law, confirmed by bootstrapping, at the project level in Eclipse and Mozilla, with respective coefficient 1.54 and 1.33 on 3 and 4 decades.

If we compare these results with the ones presented in section 5.5 Figure 9, we observe in the case of Mozilla that a significant change of coefficient at the project level vs. the developer level (Δμ = 0.2). The developer level is characteristic of the individual level whereas the project level should reflect (or not) cooperation. We can assume that this gain of reactiveness at the project level is due to the cooperation and cooperation is playing an important role in the management of the Mozilla project. However when we look at the coefficients of Eclipse between the developer level and the project level, we observe a smaller gain of reactiveness (Δμ = 1.14) at the project level. We could say cooperation is playing a role in the management of Eclipse, but less important that in Mozilla.

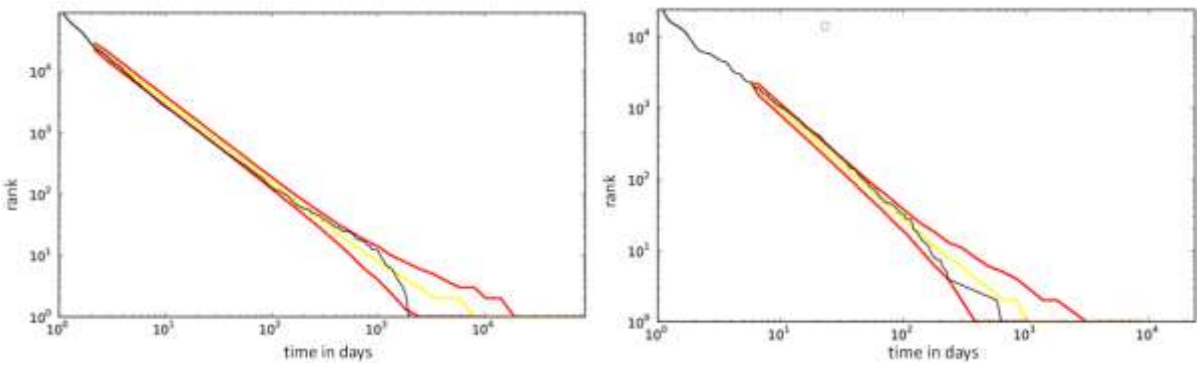

Figure 10 : bootstrapping of the dt_event distribution for Mozilla (Left) and Eclipse (Right). The abscissa represents the time in days and the ordinate is the rank of each point.

Mozilla	μ	Cutoff	95th
dt_event	1.33	1.99	1.33 ± 0.011
dt_bug	1.36	1.1	1.36 ± 0.009
dt_nonbug	1.23	1.07	1.23 ± 0.010

Eclipse	μ	Cutoff	95th
dt_event	1.56	5.65	1.56 ± 0.055
dt_bug	1.49	13.26	1.49 ± 0.084
dt_nonbug	1.43	6.73	1.43 ± 0.066

Table 3 : results of bootstrapping of the distributions of dt_event, dt_bug, dt_nonbug for Mozilla, Eclipse at zoom level 2, i.e. the project level view.

In section 5.2, we presented the concept of main folders. The activities in zoom level 2 are composed of activities of 100 folders for Mozilla and 24 for Eclipse. We would like to individually check the behavior of the distribution of waiting time and see whether we obtain power laws or not. We used the tool presented in 5.4 to fit around 300 distributions with power laws. We assumed having power laws all along the project. We cannot reject the hypothesis that these are power laws for around 70% of them. For most cases of the remaining 30%, we observe a low rate of activity (from 2 check-ins to 100 check-ins). Figure 11 show the bootstrapping of some waiting times distributions of dt_event, dt_nonbug and dt_bug in Mozilla.

We observed a large variety of coefficients for the power laws starting from μ = 0.5 to μ = 3.5, on both Mozilla and Eclipse. In section 6 we show how we used the power law coefficients of dt_bug and dt_nonbug in order to analyze our results more in-depth.

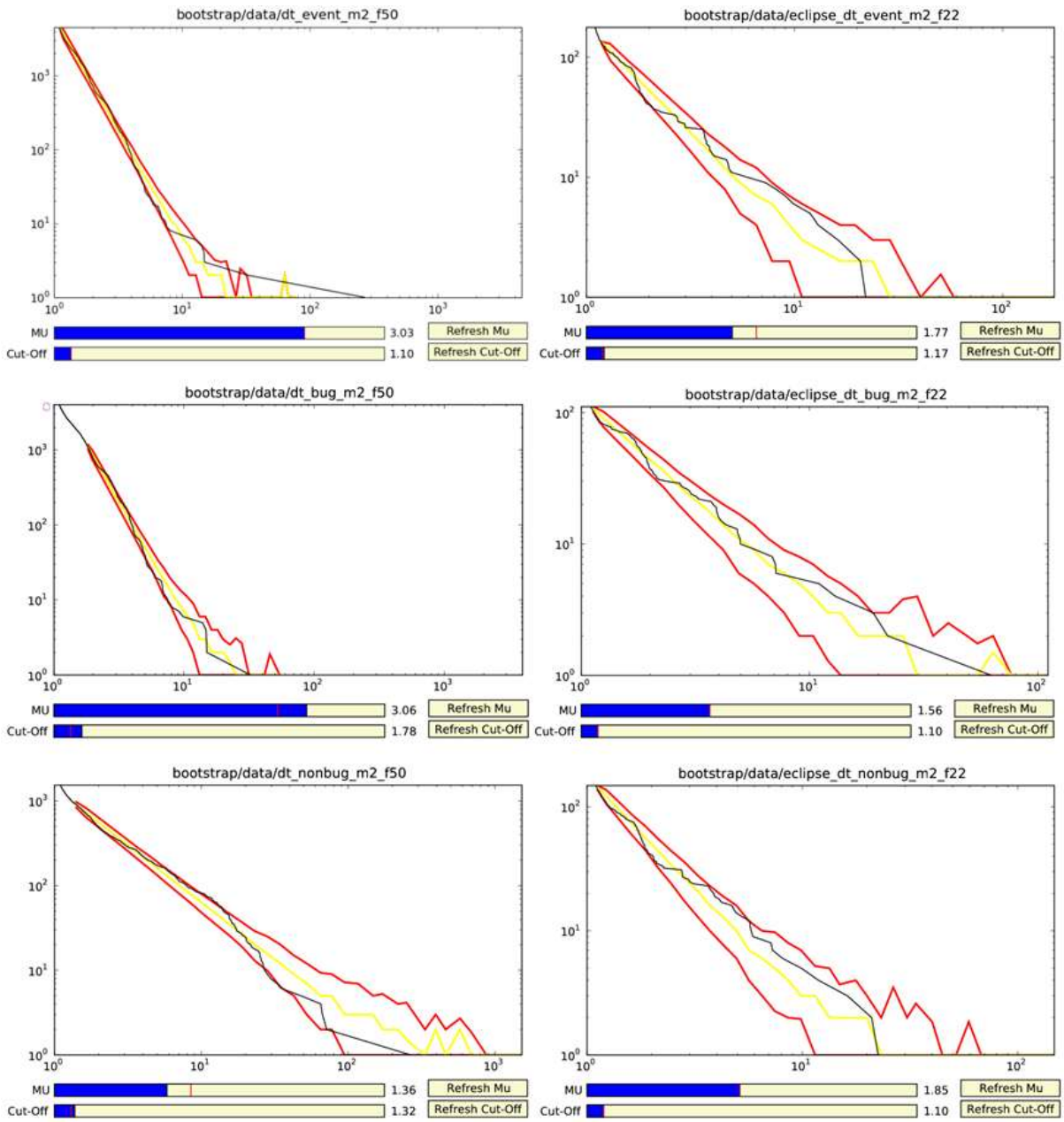

Figure 11 : Screenshots of the GUI while performing bootstrapping. It shows the results of 2 folders (columns) and the 3 metrics (rows). Each row corresponds to one metric, the first one being dt_event, the second one dt_bug, and the third one dt_nonbug. The abscissa represents the waiting time in days and the ordinate is the rank of each point. The cut-off and mu are visible at the bottom of the chart. For example: the distribution dt_event for folder 50 has a coefficient μ of 3.03 for a cut off of 1.10.

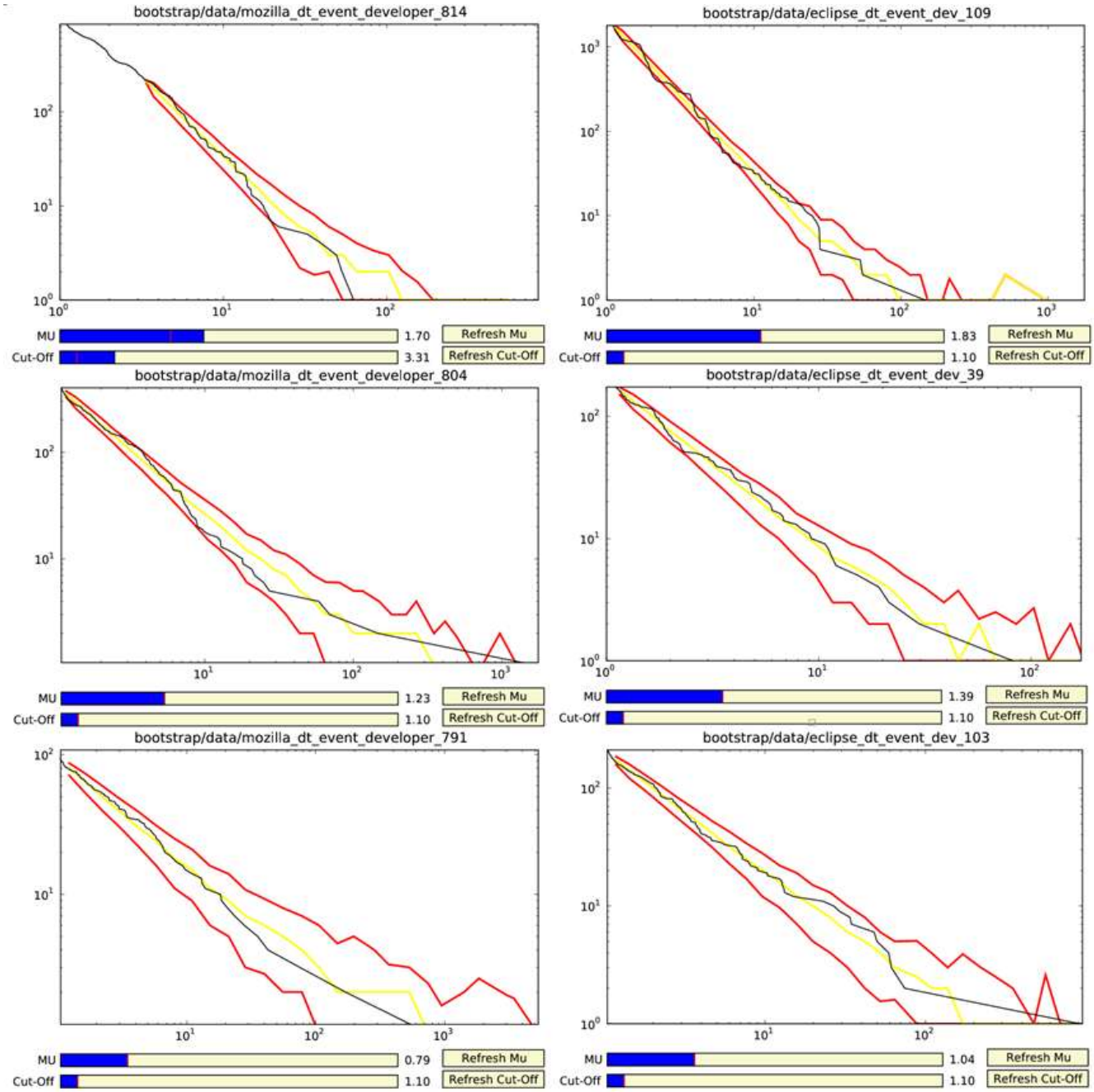

Figure 12 : Screenshots of the GUI while performing bootstrapping. It shows the waiting time distributions of all check-ins (dt_event) of different developers in Mozilla and Eclipse (left figures are Mozilla and right figures are Eclipse). The two first row figures represent the distributions of 4 different developers. The last row figures represent the distribution of the aggregation of waiting times of all developers in each project. The abscissa represents the waiting time in days and the ordinate is the rank of each point. The cut-off and mu are visible at the bottom of the chart. For example: the distribution dt_event for developer 791 has a coefficient μ of 0.79 for a cut off of 1.10.

6. STRATEGIC CRITICAL ANALYSIS

6.1 Definition

We made the following two assumptions in order to discriminate between critical events and strategic events within projects.

Assumption 1: the debugging activities of a system can be related to the system's criticality

Assumption 2: the non debugging activities of a system are related to the system's strategy implementation

The coefficient in power laws plays a decisive impact. It is important to notice that μ = 1 is a threshold representing a change of regime in ccdf. Indeed, both the average and the variance are not defined when μ < 1. There is an « infinite memory » of the process as extreme events dominate, and the system has the tendency to explore always more extreme events (i.e. waiting times are always longer). When μ > 1, the average is defined because the average size of the maximum of waiting times will remain stable enough. If the average of the distribution of dt_bug is defined, we consider the project critical. Otherwise, we consider the project as being non-critical. The same rules are applied for dt_nonbug with the strategic factor. Figure 14 presents the 4 possible zones in which a project can be assigned.

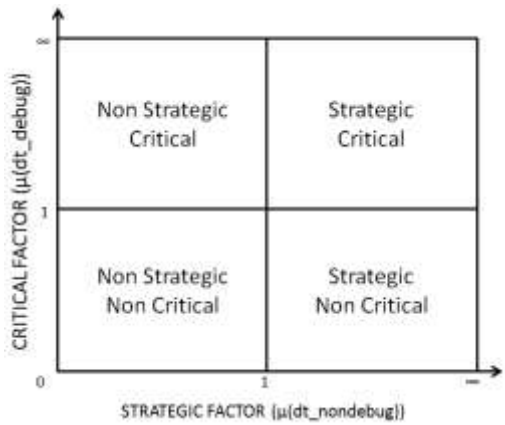

Figure 13 : The SCA Matrix. The abscissa represents the strategic factor and the ordinate represents the critical factor. The abscissa shows the strategic factor and the ordinate shows the critical factor. The threshold for both factor is represent at x = 1 or y = 1. The matrix is composed of 4 quarters, the lower left is the non strategic – non critical zone, the upper left is the non strategic – critical zone, the lower right is the strategic – non critical zone, and the upper right is the strategic – critical zone.

6.2 Methodology

Figure 14 is presenting it in a diagram form. From a CVS log for a project, we extract the waiting time between debug check-ins (dt_bug), the waiting time between non-debug check-ins (dt_nonbug), the total check-in number (check-in_number) and the developer number (author_number). author_number and check-in_number are two different metrics used to determine the size of the project that we will call respectively z1 and z2. We introduce dt_bug vector into the BSPLAW tool to obtain μ(dt_bug) and conf(μ(dt_bug)). μ(dt_bug) will be the x coordinate of our point and conf(μ(dt_bug)) gives two results confStd(x) and conf95(x) that represent respectively the confidence interval based on the standard deviation of x and the 95 percentile of x. We apply the same of dt_nonbug and obtain the corresponding results for y.

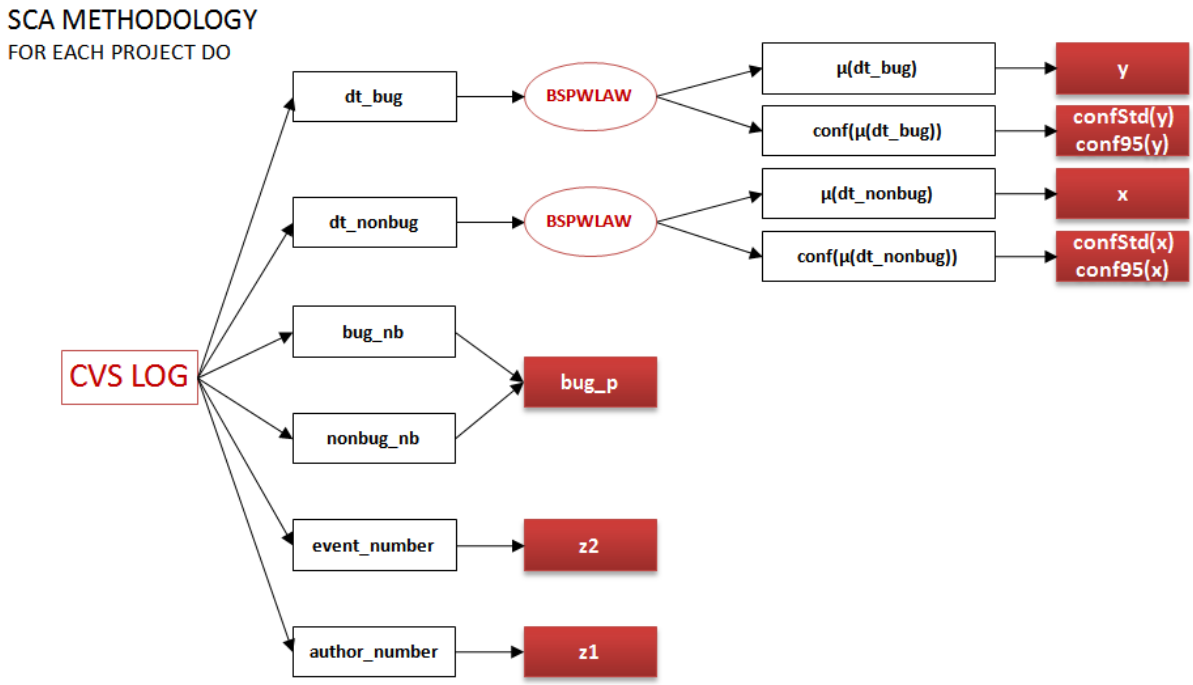

Figure 14 : methodology to perform the SCA. The red boxes represents the results of the methodology, the white box represents the intermediate metrics and BSPWLAW is the method to the GUI in order to determine the μ and the confidence interval of a power law distributions.

The main information is in x, y, and z (by z we understand z1 or z2 depending on your analysis focus). This information will help to place the project on the SCA matrix and if we have multiple projects, we can obtain a constellation of projects / sub-projects that could help

any new manager to understand where to focus. The confidence interval (confStd and conf95) results are used to statically validate our claim that one project is actually in one of the possible quadrants proposed. If one of the projects has an error bar between two zones, we will not be able to say where the project is categorized between the two zones. We distinguished between the very conservative measure at 95 percentile and the rather optimistic measure with the standard deviation.

Finally we extract for the proportion of debug check-in (bug_p) by looking at a set of projects, checking the distribution of debug check-in numbers and non debug check-in numbers and then create a parameter bug_p. This parameter will be red if the project is above the used error metrics added to the mean, blue if it is below the used metric subtracted from the mean, and purple if it is in between. Blue will represent projects in which non debugging check-ins are much more present, red will represent projects in which debugging check-ins are more present and finally purple will represent projects in which we cannot determine if debugging or non debugging are occurring more often or not.

Figure 15 shows the graphical representation of one project on the SCA matrix.

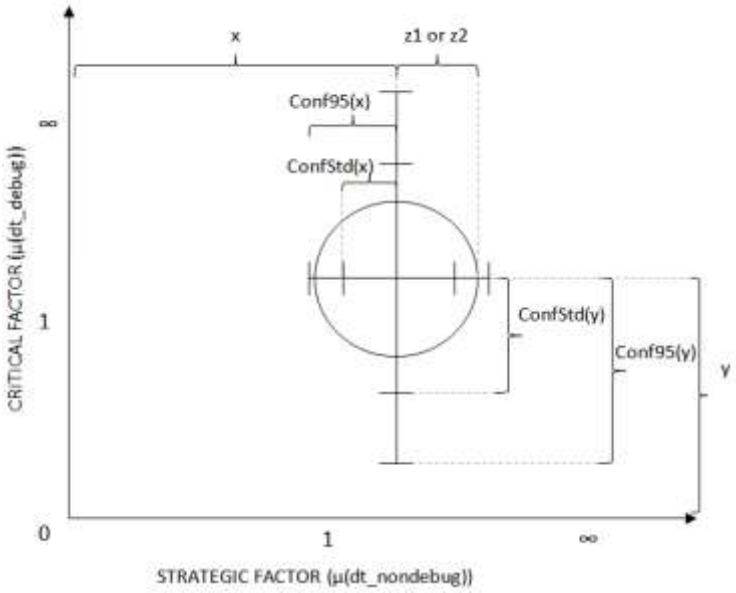

Figure 15 : legend of graphical representation of the SCA matrix. One bubble represents one project. Its size depends on z1 or z2, depending on which size factor we want to focus on. Its positions depends on x and y, or in other words μ(dt_nonbug) and μ(dt_bug). We added the two confidence intervals with bars for each axis, the first one representing the standard deviation and the second one represents the mu of 95 and 5 percentile.

6.3 Mozilla and Eclipse screened by the SCA

Figure 17 presents the results of the SCA on Mozilla data. Note that we have modules all over the Matrix. Many are at least critical or strategic, and dozens of them are Non Strategic Non Critical. Among them, 7 include more strategic check-ins, and 3 of them only contain a majority of critical check-ins. If we look at the Strategic Critical quadrant, the majority of the projects include a mix of strategic and critical check-ins with no clear advantage for one of the event types. The number of authors per project is also visually growing with the strategic and critical factors. 4 projects are categorized as strategic only vs. 16 projects are categorized as critical only. There is a small number of developers in strategic only projects if we compare them to critical only projects. We can also see 3 main clusters with the strategic projects (above 1.7 for the strategic factor), the critical projects (below 1.7 for strategic factors, and above 1.5 for critical factor) and the non strategic non critical projects. The 45° dashed line reveals that there is a clear focus on critical events than strategic events. Projects in Mozilla are rather critical than strategic. If reliability was closely associated with criticality, we could stipulate that Mozilla is composed of rather more reliable projects. This is not surprising as Mozilla offers above all web-applications where security and reliability is a condition for users to adopt the product. It also holds with the aggregated results, where we obtain $\mu_{reliability} > \mu_{creativity}$.

We can compare these results with the Eclipse results presented in Figure 18. Eclipse shows the same trend globally. We note the fact that projects are also more balanced in terms of strategic / critical check-ins. We could also identify 3 clusters with the highly strategic and highly critical projects (above 2 for strategic and critical factors), the “average” projects (from 1 to 2 for strategic and critical factor) and finally the last cluster is composed of the non strategic non critical cluster. Maximum critical factor for Mozilla is 3.2 vs. 2.8 for Eclipse. Maximum strategic factor for Mozilla is 2.4 vs. 3.2 for Eclipse. We can therefore assume that the Mozilla community is composed of more people concerned by security for which reliability have an advantage on criticality. The Eclipse community is composed of a very mixed organization that does not prefer reliability over features, except for the big projects that favor features and creativity. This results against hold with the logical sense. Eclipse is not (yet) a web-application and bugs are have less impact in the adoption of the product that bug that Mozilla. However the Eclipse Guidelines and the Eclipse Community agreed to the fact to make Eclipse a web application. We could suggest based on this short

case study that the Eclipse community should focus more on reliability in order to meet the standards imposed by Mozilla.

Finally Considering the fact that Eclipse and Mozilla are only two projects among the wide variety of OSS (SourceForge regroups by itself 230’000 software projects as of February 2009 (28)), and that most of them are using a CVS / SVN versioning system or similar, the SCA has to be applied to a larger pool of software. It will help to discriminate not only projects but also the community according to the metrics we looked at. The results could then be used for the OSS and its community to realign themselves for missing properties. Note that commercial software are often using versioning system that produces similar logs to CVS. With small adaptations, the same analysis could be done and help managers to improve their performance of specific parts of their projects.

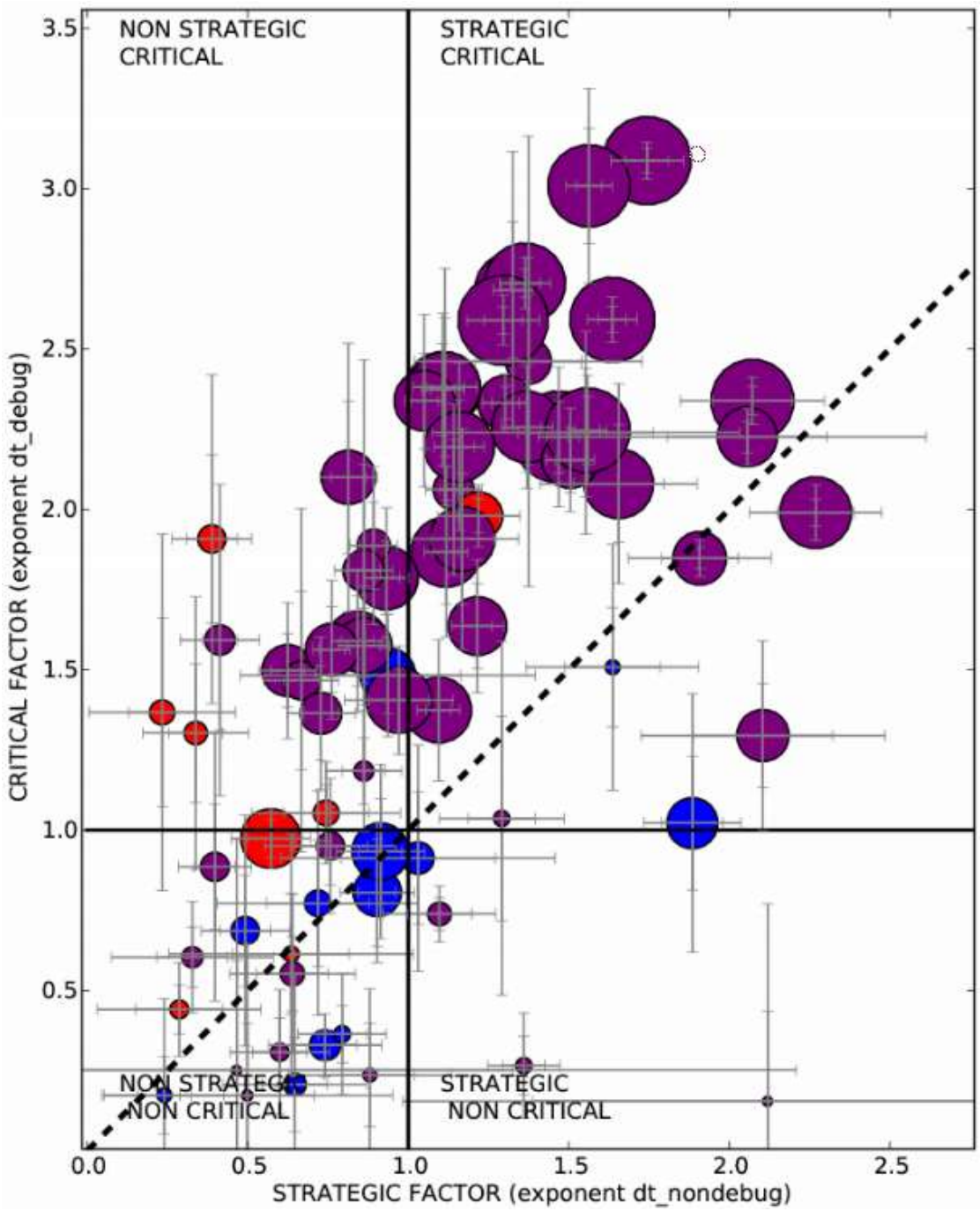

Figure 16: graphical view of the SCA matrix for Mozilla. Each point is defined according to the methodology and the legend to Figure 15. The 4 zones are represented with the lower left being Non Strategic – Non Critical, upper left Non Strategic – Critical, bottom right Strategic Non Critical, upper right Strategic – Critical. We can see visually that it the strategic factor and the critical factor evolve together with the number of author. Red Bubbles are project with significantly more debugging activities, blue bubbles are projects with significantly more non debugging activities and Purple are projects for which we cannot give a clear advantage to one of the two activities. The dash lines shows the boundary between projects that are more creative than strategic depending on the their coefficient. It shows clearly the strong focus on the critical factor.

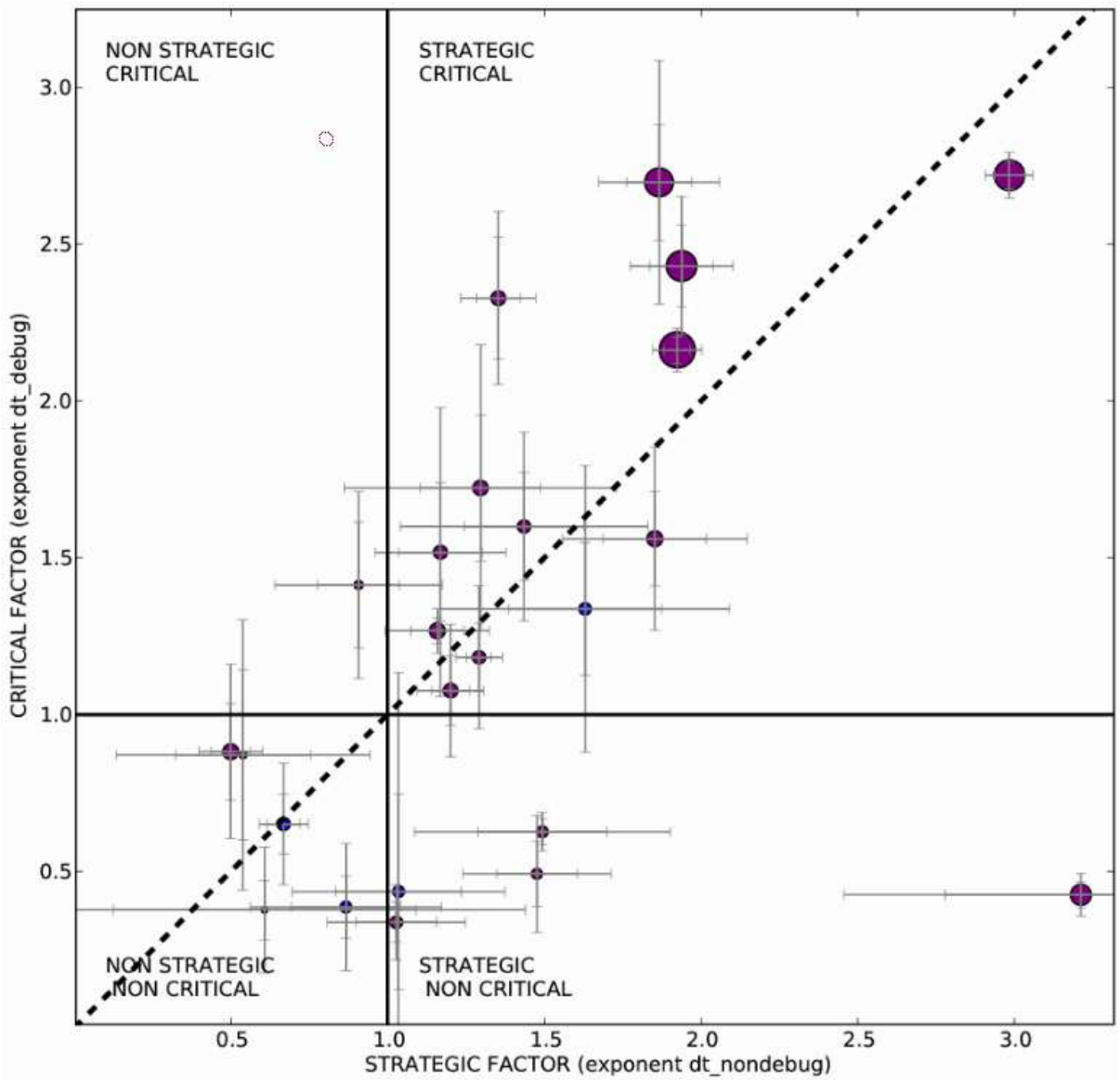

Figure 17 : graphical view of the SCA matrix for Eclipse. Each point is defined according to the methodology and the legend to Figure 15. The 4 zones are represented with the lower left being Non Strategic – Non Critical, upper left Non Strategic – Critical, bottom right Strategic Non Critical, upper right Strategic – Critical. We can observe a liner regression of y = and we can see visually that it the strategic factor and the critical factor evolves together with the number of author. Red Bubbles are project with significantly more debugging activities, blue bubbles are projects with significantly more non debugging activities and Purple are projects for which we cannot give a clear advantage to one of the two activities. The dash lines shows the boundary between projects that are more creative than strategic depending on the their coefficient. It does not highlight any focus for one of the factor.

7. LIMITATIONS

We assumed having power laws all along the project. Many indicators reveal that there might not always be a power law but also sometimes cross-over or exponentials. We could implement an algorithm to determine the nature of the distribution – power law, exponential and cross over.

Additionally the range of our studies is limited to 2 large OSS projects. All assumptions made based on the two projects are to be verified for a maximum of projects. Moreover the size and the number of check-ins in the projects are one critical parameter that one has to take into consideration when applying the SCA methodology. We propose an arbitrary limit of at least 10’000 check-ins for each OSS measured.

We stated the hypothesis that the developers will use bugzilla as a bug tracking system. For the sake of the model and to provide an example, we accepted this fact. However, after discussion on the developer's IRC chart of Mozilla, developers told us that actually the bugzilla system was used not only as a bug tracking system but also as a project management system. We cannot make any interpretations related to strategic vs. critical for the Mozilla software with the current way of processing the data. The developers were very skeptical about finding a way of automatically processing if check-ins are related to a bug check-in or not. We propose therefore to find projects that use a CVS/SVN collaboration system that is known to use bugzilla as a bug tracking system only. In the outlook part, we also propose research questions based on project management and change management based on the comments of the developers.

Also assuming someone will find out how to automatically track if a CVS entry is a bug or not, then this person should consider recreating the level 2 main folders structure. We used a bottom-up approach in order to determine the best granularity to look at and we assumed that these were projects considering the similarity of the folder structure obtained and the project list of the Mozilla website. One could better discriminate the project and assess them more realistically by either using the Mercurial repository (MCX) system that replaced CVS (29). MCX offers more log functionalities and we can also track the owner of modules which leads to modules themselves. The second option using CVS could be to use the documentation on Mozilla that lists the folders and sub-folders included for each module. Taking this list of folders and integrating into the management will make it easier to assess

the project and perform qualitative research with developers as the material of discussions will start from the same basis. Finally Eclipse repository has been provided by the courtesy of the SEAL. However the data includes the core modules of Eclipse only. This could explain the emphasis on equilibrium of this project. Receiving all the CVS log check-ins from all the Eclipse projects would provide a very interesting pool of project to perform the analysis.

8. CONCLUSION AND OUTLOOK

We contributed in showing the importance of waiting times in OSS projects and proposed a methodology using this metric and the CVS / Bugzilla combination to discriminate between Strategic and Critical projects. This methodology holds with the market reality with the two case studies represented event without perfect data. It can be used by a wide variety of business users ranging from Strategic Consultants to Investors to Project Managers in order to improve efficiency and focus of the developers effort on software projects. Moreover we introduced the role of cooperation to reduce the turnaround time in OSS projects.

A next step in the studies of waiting times in OSS would be to explain the influence of cooperation on waiting times. Our results show that the number of developers impacts positively the Strategic and Critical factors and that the coefficient of the power laws is projects is higher than at the developer level. However it is not because many developers are joining one project that they collaborate with each other. One should be able by comparing on projects by projects basis, the exponent of the aggregated developer level vs. the project level and from the difference interpret where cooperation really takes place.

One parallel approach could be done by taking advantage the limitations presented above for Mozilla. We can argue that project management is used to its full standard when external people need to understand the project progress, or in other words when collaboration is needed. Figure 18 presents two story plots of two different files. The blue bubbles represent the check-in non-linked to the Bugzilla system (Non Collaborative Behavior) and the red squares represent the check-in linked to the Bugzilla system (Collaborative Behavior). These two figures show that project management tool is often not use when either one is integrated in the project (beginner mistake) or at the beginning of the project (left figure). When more and more user are joining (right figure), we can see that project management is widely used by the vast majority of developers. Figure 19 shows an interesting perspective of the development of the usage of bugzilla with time on the overall project. There has always been usage of the bugzilla system. It is only after 700 days the project management took more share than the usage of non project management. Before that date, developers were not correctly respecting the tools of cooperation. We can see

however a clear transition that will last 3000 days where developers will be using more and more the project management system to reach a 70% rate of check-in.

If we consider the use of CVS Bugzilla mix as the perfect collaborative behavior, we could capture 1- the influence of collaboration in waiting times (turnaround time) and 2- the resilience of the use of project management tools by developers and by projects, and by modules owners.

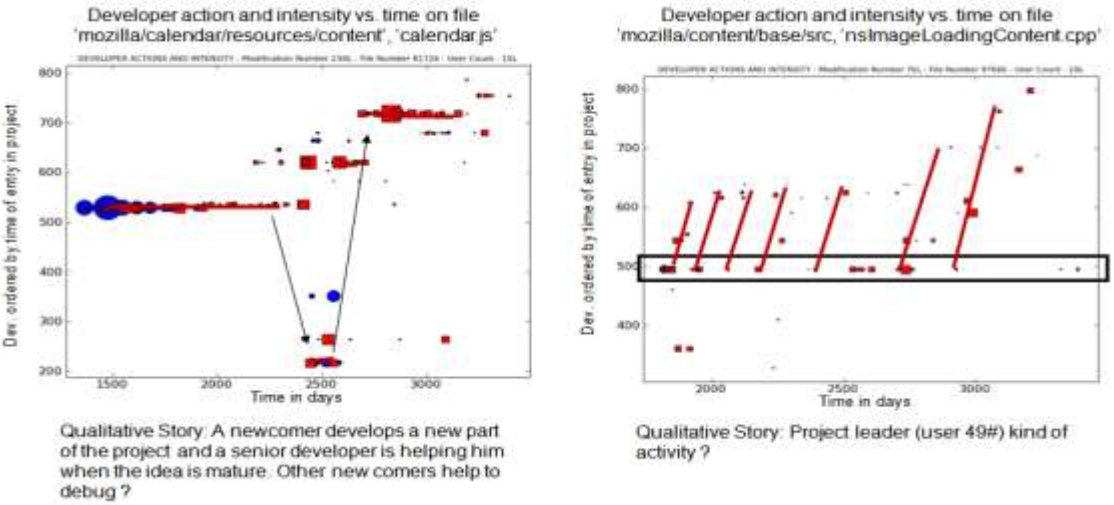

Figure 18 : check-in of developers in two different files. The abscissa represents the time in day and the ordinate represents the developer id order by date of entry in the CVS log. Each point represents an check-in. The red points are check-in linked to bugzilla and the blue points are check-in non-linked to bugzilla. The size of the points represents the number of added lines + deleted lines for each check-in.

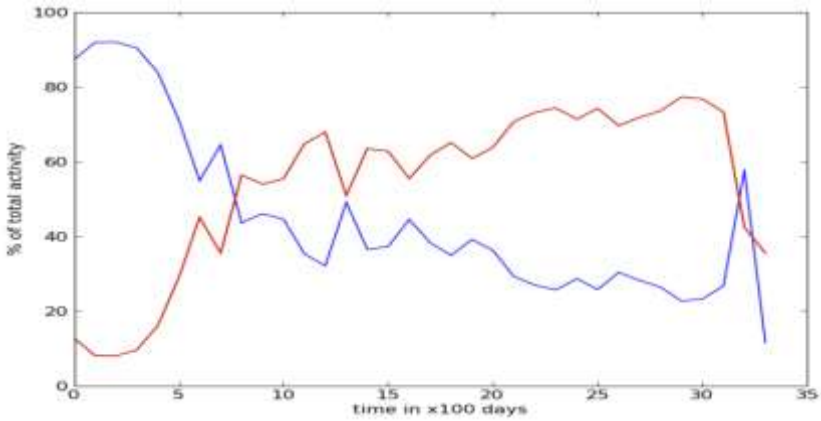

Figure 19 : Figure: Average per 100 days of the linking between bugzilla and CVS in the Mozilla project. The blue line represents the percentage vs. all check-in of non-linking CVS to bugzilla and the red line represents the percentage vs. all check-in of linking to CVS to bugzilla.

REFERENCES

1. **Shiels, Maggie.** Call for open source government. *BBC News.* [Online] 01 21, 2009. http://news.bbc.co.uk/2/hi/technology/7841486.stm.

2. **Guillemin, Christophe.** French parliament dumping Windows for Linux. *CNET NEWS.* [Online] 11 27, 2006. [Cited: 09 24, 2009.] http://news.cnet.com/2100-7344_3-6138372.html.

3. **Hillenius, Gijs.** FR: Gendarmerie saves millions with open desktop and web applications. *Open Source Observatory and Repository.* [Online] 03 06, 2009. http://www.osor.eu/news/fr-gendarmerie-saves-millions-with-open-desktop-and-web-applications.

4. **TechCrunch.** Report: Firefox 3.5 Jumps To 4.5% Market Share In A Month, IE Hemorrhaging Slows. *TechCrunch.* [Online] August 4, 2009. http://www.techcrunch.com/2009/08/04/report-firefox-35-jumps-to-45-market-share-in-a-month-ie-hemorrhaging-slows/.

5. Web Server Survey Archives. *Netcraft.* [Online] Netcraft. [Cited: 9 28, 2009.] http://news.netcraft.com/archives/web_server_survey.html.

6. **Raymond, E.** The cathedral and the bazaar. *Knowledge, Technology and Policy.* 1999.

7. **von Krogh, von Hippel.** Special issue on open source software development. *Research Policy.* 2003, pp. 1149-1157.

8. **Moerke, K.A.** Free speech to a machine. *Minnesota Law Review.* 2000, pp. 1007-1008.

9. **Simon, E.** Software Development: process and performance. *IBM Systems Journal.* 1998, pp. 552-569.

10. **Eric von Hippel, Georg von Krogh.** Open Source Software and the “Private-Collective” Innovation Model: Issues for Organization Science. *Organization Science/Vol. 14, No. 2,.* March–April 2003, pp. 209-223.

11. **Mozilla.org.** Becoming a Mozilla Committer. *Mozilla.org.* [Online] [Cited: 09 23, 2009.] http://www.mozilla.org/hacking/committer/.

12. **Rishab A. Ghosh, Ruediger Glott, Bernhard Krieger, Gregorio Robles.** Free/Libre and Open Source Software: Survey and Study. *International Institute of Infonomics, University of Maastricht and Berlecon Research GmbH.* June 2002, p. Part 4: Survey of Developers.

13. **R. Crane, D. Sornette.** Robust dynamic classes revealed by measuring the. *Proc. Nat. Acad. Sci. USA 105 (41), 15649-15653.* 2008.

14. **Stefan Haefliger, Georg von Krogh, Sebastian Spaeth.** Code Reuse in Open Source Software. *Management Science.* 2007.

15. **Boulanger, A.** Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM SYSTEMS JOURNAL, VOL 44, NO 2.* 2005, pp. 239-247.

16. **Gareth Baxter, Marcus Frean, James Noble, Mark Rickerby, Hayden Smith, Matt Visser, Hayden Melton, Ewan Tempero.** Understanding the Shape of Java Software. *School of Mathematics, Statistics, and Computer Science - Victoria University of Wellington, New Zealand.*

17. **A-L, Barabási.** The origin of bursts and heavy tails in human dynamics. *Nature.* 2005, Vol. 435, 207–211.

18. **Oliveira JG, Barabási A-L.** Human dynamics: Darwin and Einstein correspondence patterns. *Nature.* 2005, Vol. 437, 1251.

19. **Vazquez A, et al.** Modeling bursts and heavy tails in human dynamics. *Physical Review.* 2006, 73:036127.

20. **J.G. Olveira, A. Vasquez.** Impact of interaction on human dynamics. *Physica A.* 2008, Vol. 388, 187-192.

21. **Wikipedia.** Concurrent Versionning System (CVS). *Wikipedia.* [Online] [Cited: 10 01, 2009.] www.wikipedia.org/wiki/CVS.

22. —. Mozilla. *Wikipedia.* [Online] [Cited: 10 01, 2009.] www.wikipedia/wiki/mozilla.

23. **Mozilla.** About. *Mozilla.* [Online] [Cited: 10 01, 2009.] www.mozilla.com/about.

24. **Wikipedia.** Eclipse. *Wikipedia.* [Online] [Cited: 10 01, 2009.] www.wikipedia.org/eclipse.

25. —. Debugging. *Wikipedia.org.* [Online] [Cited: 09 23, 2009.] http://en.wikipedia.org/wiki/Debugging.

26. **Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman.** Power-law distributions in empirical data. *arXiv.* June 2007.

27. **Wolfram MathWorld.** Bootstrap Methods. *Wolfram MathWorld.* [Online] [Cited: 09 27, 2009.] http://mathworld.wolfram.com/BootstrapMethods.html.

28. **Sourceforge.** About. *Sourceforge.* [Online] [Cited: 09 28, 2009.] http://sourceforge.net/about.

29. **Mozilla.** Developer Guide - Source Code. *Mozilla.* [Online] https://developer.mozilla.org/En/Developer_Guide/Source_Code.