Revealing Governance Patterns by Taking Snapshots of Open-Source Communities
August 2013
Alexandra Leța

1 Introduction

Chapter 1 consists of a context overview, section 1.2 presents the problem of this thesis. In section 1.3, the solution will be provided in the scope of section 1.4. Sections 1.5 and 1.6 provide the research questions and approach, respectively. Section 1.7 provides a planning. The final section of chapter 1 gives an overview of the content of this thesis.Studying the Impact of Social Structures on Software Quality.

1.1 Context

Software projects continue to grow in size and complexity and an important requirement for studying snapshots of development communities is the acquisition and analysis of data from software repositories. Analysis of software systems communities and their evolution is needed to better understand the characteristics of software systems and projects and can be used to uncover knowledge and assist in software developments. Literature in software development communities already recognizes the need to study the effect of organizational structures on the quality of software, motivating further studies for understanding, representing and supporting social structures and communities.

The relationship among software communities' practices and characteristics related to community types arises an interesting research problem as an emerging body of literature reveals open source software development as an alternative to proprietary software. ***The main challenge imposed by open source software is related to its basic motivations and values of the institutions that develop, market, and use open software.*** The purpose of analyzing snapshots of open-source software communities is to reveal patterns in the software development process and social interactions that occur in these communities. This set of community characteristics and metrics could be used to understand how developers interact while working on a project and to study the effects of these social interactions under conditions in which different projects have different levels of developer skill, and where features added by developers have the potential to influence other features.

1.2 Problem

Open source software communities are defined as a group of people who share similar goals, interests, beliefs, and value systems in a common domain of recurring activity or work **[1.2-Walt]**.
This characterization is used by researchers to investigate groups of people working together and using information technology systems.

The study of open-source community characteristics and their mapping to different community types help us understand the social structure of a community in order to be able to overcome different communication and collaborating barriers that may hinder a successful development process. A common method for acquiring knowledge on software communities patterns is to study the data provided by software tools used by community members. Previous studies on gathering data from software repositories have been conducted and their results were offered back to the community for further research purposes. Howison et al, in the FLOSSmole project **[1.2-**

Flossmole], extracted a dataset which consists of metadata about the software packages, such as numbers of downloads or the employed programming language. The Flossmetrics project provides a dataset consisting of source code size, structure and evolution metrics from several open source software projects. A common problem identified in software repositories analysis is tracking developer identities across data sources as most software configuration management systems rely on user names to identify developers, while mailing lists and bug tracking software use the developers' emails. *A possible solution to the problem is offered by Git, which uses emails to identify developers, while both bug descriptions and wiki entries coming from the same developers feature their identity details.*

1.3 Solution

The main purpose of this research is to construct and analyse information and metrics on software development and their relation with software communities, using existing methodologies and tools. Software communities members use tools such as project's mailing lists, versioning systems, bug-tracking systems, discussion forums to enable peer collaboration in the software development process. This process leaves an accessible trail of data upon which interesting analyses can be built, but despite of the large set of data researchers often face significant challenges in using it. The Mining Software Repositories field analyzes the rich data available in software repositories to uncover actionable information about software systems and projects [1.3-Ahmed].

This project is designed to gather collaborative data of open source software community, process and analyze this data in order to identify patterns that are most relevant for this community. We will focus on community types that are most consistent with open-source development, namely: Formal Networks, Informal Networks, Informal Communities and Networks of Practice **[1.3-Titans]**. Using data gathered from software repositories and attached collaboration tools we define a set of properties for each analyzed repository. We define quality attributes and their accepted values for community types most compatible with open-source software in order to understand patterns in the open-source development process. Using this data and additional information extracted from the literature in social communities we can validate metrics that measure these key-attributes.

The results provided by this research process are reproducible, extensible and easy to compare when applied on multiple repositories. The open source software community development method poses a serious challenge to the commercial software businesses that dominate the software markets **[1.3-ACS]** and this project creates the opportunity to exploit the diversity of existing research, to compare and contrast results in order to expand our knowledge regarding how social communities interactions could affect the software development process.

1.4 Research question

The number of people that manage to work together successfully to create high quality, widely used products is remarkably large. The research questions are aimed at understanding basic parameters of the process of creating open source software and how are these parameters related to the community type. The project aims to provide metrics resources on development practices in the open source software industry and to identify and understand patterns of development practices in this industry. **What are the community and organizational patterns in open source development communities?**

In order to provide an answer for these questions the grounded theory research method is considered. Using this technique, the research findings constitute a theoretical framework "grounded" in the data, with built-in test of

the theory incorporated into the results. The first step in analyzing data collected is to perform open coding, the process of breaking down, examining, comparing and categorizing the data **[1.4-Grounded]**. The next step involves reexamining the data, looking for connections between categories.

2. Repository analysis

In order to address the research question and understand how community members' interactions can influence the quality of the development process, we need key measures of project evolution and social interactions during the development process. The study of software developments snapshots includes creating basic summary reports about the state of the source code, as well as network analyses of open source development teams. Summary reports should include the number of operations, chosen programming language, the number of developers per project, etc. This data is useful for creating general statistical information about open source projects.

When conducting research with data from software repositories, an initial step involves retrieving of the project data to be analysed. Storing the processed data and running the analysis on updated collection of data will allow metrics to be compared over time, supporting historical comparisons and trying to make the analyses reproducible and extendable.

Members of the software development communities do not work in a single or central workplace, and often there is no formal management hierarchy in place to schedule, plan, and coordinate tasks and resources. Instead, developers contribute their effort to projects that they find significant, or otherwise professionally compelling. The snapshot data of this community includes data significant for the most common activities performed by its members: writing code and comments, fixing and submitting bugs, collaborating in order to find better solutions for their projects. GitHub Archive records the public GitHub timeline, archives it, and makes it easily accessible for further analysis and this archive dataset can be accessed via Google BigQuery.

Moreover, GitHub provides access to its internal data stores through an extensive REST API which researchers can use to access a rich collection of data and which enables retrieving commits to the projects' repositories and events generated through user actions on project resources. Moreover, GitHub facilitates project contributions from non-team members, offers software forge facilities, such as a issue tracker, a wiki, and developer collaboration channel. The provided JSON format data is constantly updated and it allows running arbitrary queries and analysis over the entire dataset. This community data is used to compute attributes values related to the software development process in order to study some of the known phenomena observed in the open source software development community and provide a better insight into their common practices and interactions.

We use metrics to increase our understanding of core software development activities and the overall nature of community participation in software projects. An important project quality metric is its complexity which can be used to predict the incidents of faults in code. A case study which used data derived from the change history of open source projects shows that change complexity metrics, based on the code change process, are reliable predictors of fault potential **[2-Hasan]**. Similarly, researchers investigate how collaborative activities carried out by the developers such as social networks [2-R30], work dependencies [2-R10] [2-R] and daily routines [2-R27] are connected to the software product quality.

Using these variables we can assess the important project properties such as formality, which can be determined by calculating the number of milestones assigned to the project, number of user groups, etc. Another important

property is the degree of formality which can be monitored by investigating the presence of social and informal interactions between developers. The degree of engagement from community members evaluates members' engagement, by analyzing member's participation levels.

The following open source tools were used for the data extraction, processing and visualization:

- GitHub Java API: library for communicating with the GitHub API, supporting the GitHub v3 API **[E-Git]**. The client package contains classes communicate with the GitHub API over HTTPS. The client package is also responsible for converting JSON responses to appropriate Java model classes. The package contains the classes that invoke API calls and return model classes representing resources that were created, read, updated, or deleted.
- Gephi Toolkit: Java library which provides useful and efficient network visualization and exploration techniques **[Gephi-Tool]**. The toolkit is a single JAR that could be used in applications and achieve tasks that can be done in Gephi automatically.
- Google Geocoding API: used for converting the addresses of repositories members into geographic coordinates, which is used to calculate distances. This API **[Geo]** provides a direct way to access services via an HTTP request.

The final set of data includes attribute values for software repositories, the name and version of the project, meta-data, and additional statistics. Evaluation and analysis is performed after the metric computations are performed.

Based on the described data extraction mechanism, we obtain a network of nodes representing community members and edges representing a particular interaction between two users. We considered that two nodes are connected if at least one of the following condition is fulfilled:

1. *Common projects: two community members have at least one common repository to which they are contributing, except for the currently analyzed repository*
2. *List of followers: between the considered community members exists either a "is following" or "follows" relation*
3. *Pull request interaction: we consider the connection between the pull request author and other community members that are participating on the pull request*

*The vast majority of repositories hosted on Github are representative for open source software projects. Based on the evidence from **[1.3-Titans, [39-Titans], [36-Titans]** the community types most consistent with open source software development communities are represented by: Formal Networks, Informal Networks, Informal Communities and Networks of Practice. Moreover, after analyzing the community types **[Titans]** produced a set of metrics defining the key attributes for these communities and a set of threshold values that are most representative for different community types using SourceForge project data. The current approach uses Github repositories as data sources as it has became the world's largest open source community. Github provides an extensive REST API, which enables researchers to retrieve both the commits to the projects' repositories and events generated through user actions on project resources and has emerged as a popular project hosting, mirroring and collaboration platform **[Ghtorrent]**. For the selected Github repositories we computed the attributes values identifying the mentioned communities types, using similar metrics. After computing the key-attributes values, the repositories where further analysed by defining a set of quality metrics and their values. **[Using the attributes values that define different community types and quality attributes values]***

*The most important attributes identified for Formal Networks are **formality** and **membership status**. Using the approach defined in **[1.3-Titans]** the attribute value formality can be computed using information regarding the number of milestones assigned to a project and the project lifetime. Only a limited number of projects hosted on Github have defined properties such as issues and milestones; our framework restricted the repository selection to projects that have these two properties. The membership attributes evaluates whether members and rigorously selected. For computing the value of this attribute we took into consideration the membership types assigned to a repository member: contributor and collaborator, as defined by the Github API. A contributor is member outside the core development team of the project who wants to contribute some changes to a project and a collaborator is a member on the core development team of the project and has commit access to the main repository of the project. The value associated to the hierarchy degree represents the percentage of contributors from the total number of repository members.*

*Informal networks are defined by properties such as **informality** [[INSERT PICTURE HERE]], **openness** and **lack of governance.** For each repository member we can extract the repositories to which he has contributed, using the RepositoryService and the company, in case we has filled this profile information. Using this collected data about users and their activity we can compute the value for the informality attribute. The lack of governance attribute is computed based on the information regarding project milestones and project lifetime. The Github API imposes a restriction for requests regarding team components and team organization. Only users that are members of a specific repository can access this data, therefore the value for the openness attribute could not be computed.*

*Figure [[INSERT PICTURE HERE]] represents the geographical distribution of members collaborating on one of the considered repositories. The same representation is available for each repository as most Github users have a location property defined. Using the Google Geocoding API we convert the addresses into geographic coordinates, which are used to calculate the **average distance** between community members. In addition to average distance attribute, cultural distance, self-similarity and number of active members are also key-attributes for Networks of Practice. We computed the values for **cultural distance** using the indices defined by Hofstede in a study on cultural dimension; for each community member who has the location property, we determined the index values for his country. Github users profiles don't include their skills; based on the defined programming language [[INSERT PICTURE HERE]] for repositories to which a user has contributed we were able to determine a set of skills for each user and compute the value for the **self-similarity** attribute. The CommitService allows retrieving the commits for a repository, limiting their number proportional to the maximum number of requests allowed by the Github API. Based of the list of commits we can determine the number of users whose total number of commits contribution is equal to at least half of the commit activity.*

*Based on the evidence from **[1.3-Titans], [36-Titans]** engagement, self-similarity and dispersion are key attributes defining a network of practice. One indicator of high engagement within a community is the number of comments members post. Using the PullRequestService and CommitService resources we extracted comments posted by community members that were commenting of a pull request or adding a comment for a commit event. Using properties such a comment author and date we were able to compute the average number of comments posted by repository members. Figure [[INSERT PICTURE HERE]] presents the monthly number of comments distribution for the **[X]** repository.*

I have used to following query to select the most popular repositories available on Github, considering the number of pull requests attached to the repository. GitHub Archive is a project which pushes its extensive collection of GitHub data to Google BigQuery. BigQuery in turn lets you write SQL-style queries on ginormous datasets like this one. The data generated from this activity can reveal interesting trends across many industries,

including popularity of programming languages over time, defect rates, contribution metrics, and popularity of specific frameworks and libraries.[1]

```
// top repositories by number of forks
SELECT repository_name, repository_organization,  repository_url, max(repository_forks) max_forks,
FROM [githubarchive:github.timeline]
WHERE repository_has_issues
   and repository_has_wiki
   and repository_watchers > 0
   and repository_organization!=""
GROUP BY repository_name, repository_organization, repository_url
ORDER BY max_forks DESC
```

For each of the analyzed repositories we computed the following attribute values.

Important software quality indicators include code metrics. The following variables can be determined by analyzing the content of the comments, issues descriptions and posts added by project members:

- The content of the comment or message which could be useful for placing a message in a topic space
- Number of new authors over time describing the evolution of the community which is an indicator of how long one author remains part of the community
- The contribution by community members during the project lifetime which can indicate the knowledge level, the depth of involvement in the community

The following variables can be determined by retrieving and analysing events generated through user actions on project resources:

- Aggregated number of commits over time which also includes the number of commits per committer - measuring individual productivity and activity
- Number of distinct committers/authors per file - the overall number of individuals participating in the development of one specific file
- Associativity - measuring preference to connect to other community members
- *Distribution of responsibilities – measuring show responsibilities, collaboration and communication are distributed across community members*

[IMPortant] Grafic cum au evoluat Opened Problems, Closed Problems, and Backlog Management Index by Month.

We performed three experiments to predict future faults in the subsystems of large software systems:
1. Modifications vs: Faults: We compare the performance of predictors based on prior modifications with ones based on prior faults.
2. Modifications vs: Entropy: We compare the performance of predictors based on prior modifications with ones based on our HCM entropy models.
3. *Faults vs: Entropy: We compare the performance of predictors based on prior faults with ones based on our HCM entropy models.*

3. Quality of the development community

Previous studies have mainly focused on rather simple proxies of social dynamics like the evolution of the number of contributors and contributions or the time span of a user's activity and were mostly based on a rather limited set of snapshots of a single project. Using a large scale dataset of time-stamped social interactions that has been collected from the Bugzilla bug-tracker installations of 14 major OSS projects, in this paper we study the fine-grained evolution of structural features of networks of user collaborations. We thus take a network perspective on OSS communities and highlight differences in the social organisation of software projects that can be related to their activity, their cohesion as well as their resilience against fluctuations in the community members. By applying standard measures from social network analysis we particularly quantify how tightly community members collaborate, how equal responsibilities are distributed and how resilient collaboration topologies are against the loss of (central) community members. While similar tools have been applied to OSS projects before [3, 6], to the best of our knowledge, the present paper is the first to study these network analytic measures on a dataset that covers the full, fine-grained history of 14 well-established and successful OSS communities.

In order to make substantiated statements about the structure and dynamics of the social organisation of OSS communities, we recently completed collecting data on the history of user collaborations recorded by the Bugzilla installation of 14 well-established OSS projects. Bugzilla[9] is an open source bug tracking system which is utilised by users and developers alike to report bugs, keep track of open issues and feature requests and comment on issues reported by others. Since the Bugzilla installations of OSS projects are used to foster collaboration between community members, it constitutes a valuable source of data that allows us to track social interactions between developers and users.
Data in the Bugzilla database are arranged around the notion of bug reports. Each bug report has a set of fields describing aspects like the user who initially filed the bug report, its current status (e.g. pending, reproduced, solved, etc), to whom the responsibility to provide a fix has been assigned, attachments which may be used to reproduce or resolve the issue, comments and hints by other community members, or a list of community members which shall be informed about future updates. Apart from an initial bug report, Bugzilla additionally stores the full history of all updates to any of the fields of a bug report. Each of these change records includes a time stamp, the ID of the user performing the change as well as the new values of the changed fields. While our dataset comprises change records for all possible fields, in this article we focus on those that indicate changes in the users that are assigned responsibility to fix an issue (henceforth called the ASSIGNEE field) and changes to the list of users to whom future updates of the bug shall be sent via E-Mail (henceforth called the CC field). We consider any updates in the CC and ASSIGNEE field of a bug report as a time-stamped edge from the user who performed the update to the user(s) who were added to the CC field or the ASSIGNEE list of responsible developers respectively.

3.1 Resilience against fluctuations of community members

A. Measures of Discussion Contents In a previous study [6], we asked developers of the ECLIPSE and MOZILLA projects, which of the information inside bug reports is most helpful for them when working on the reported issues. Among the top answers were information items like crash reports (in the form of stack traces), source code examples and patches. As a precise understanding of a problem is crucial for addressing a reported issue adequately, we conjecture that the presence or absence of such information items in bug reports can possibly

influence the quality of the source code changes carried out under the context of the reported issue. For example, discussions of test cases can help developers to implement
regression tests that safeguard the software against an accidental reintroduction of the problem in the future.
Therefore, we choose to incorporate structural elements as factors in our statistical models. We used the infoZilla tool [7] to extract structural elements from the textual contents of bug report discussions. In the following we describe the seven measures of discussion contents we used in our study.
1) Source Code: Our first measure is the amount of source code (NSOURCE) present in a discussion. Source code can find its way into an issue report due to several reasons: reporters point out specific classes and functions they encountered a problem with, or provide smaller testcases to exactly illustrate a misbehaviour; developers point users at locations in the source code they require more information about, and discuss possible ways to address an issue with peers. As the complexity of the code discussed might be an indicator for the intricacy of the reported problem and an indicator of future risk, we also compute the source code complexity (NSCOM) as a qualitative measure for each of the source code examples. Since the discussions often contain a mix of natural language text and structural elements, we use McCabe's cyclomatic complexity [18], rather than lines of code as our complexity measure.

2) Patches: Our second measure is the amount of patches (NPATCH) provided in the discussions. Publicly discussed patches provide peer-reviewed solutions to the reported issues and as such are less likely to contain errors. In addition to this quantitative measure we also compute a qualitative measure of patches by recording the number of files changed by a patch (PATCHS), which measures how spread out changes provided by the patch are. We motivate this choice with the idea that patches resulting in large or wide-spread changes to the source code might negatively impact dependent parts of the code (even though they correctly fix the reported issues).

3) Stack Traces: Our third measure records the amount of stack traces (NTRACE) provided in the contents. Information inside stack traces provide helpful information for developers to narrow down the source of a problem, and are hence valuable for finding and fixing the root causes of issues rather than addressing their symptoms [31]. We use the number of methods reported in the stack traces as a qualitative measure for the size of stack traces (TRACES).

4) Links: Our fourth measure of discussion contents records the amount of links (NLINK) present. Developers and users use URLs to provide cross-references to related issues and to refer to external additional information that might be relevant to the original problem. B. Measures of Social Structures In addition to the information obtained from the textual contents of discussions, we also compute a number of measures to describe the social structures created through issue reports. In the following we describe the five measures of social structures used in our study.

1) Discussion Participants: In order to contribute to the BUGZILLA system, users have to sign in with a username and password. The username acts as a unique handle for all his activity in the system. We conjecture that the total amount of unique participants in the discussion of an issue report is an indicator for the relative importance of the reported problem. Our first measure hence counts the number of unique participants (NPART) in the discussion.

2) Role: In this study we further categorize participants into two different roles: developers and users. We consider a participant as a developer if he was assigned to fixing at least one issue reported in the past. By measuring the number of unique users (NUSERS) and the number of unique developers (NDEVS) participating in the discussions we can distinguish between internal discussions (more developers than users), external discussions (more users than developers) and balanced discussions (even amount of developers and users).

3) Reputation: Another social property of participants, which is orthogonal to their role is their degree of reputation in the community. In our study, we determine the top three
participants with the highest reputation (expressed by the past amount of contributed messages) for each discussion attached to issue reports. These measures are captured in
the three factor variables (CON1), (CON2), and (CON3). The degree of reputation of a participant can influence the development process connected to an issue; for example Guo et al. show that defects reported by more reputable users have a higher likelihood to get fixed [15]. As each factor variable introduces many degrees of freedom to the model, we limit our study to the top three most reputable developers.

4) Centrality: Our last measure of social structure is taken from the area of social network analysis. For each discussion attached to an issue report in the bug database we first construct a discussion flow graph. The discussion flow graph is an undirected graph that has participants as nodes and contains an edge for every pair of two consecutive messages in the discussion connecting the message senders. We express the interconnectedness of nodes (participants) in the discussion flow graph as a measure of closeness-centrality (SNACENT). This measure focuses on how close each participant is to all other participants in the discussion [29]. However, closeness-centrality is a per-node measure, yet we want to express whether there is a healthy discussion between all participants, or whether there is a smaller set of key participants that drive the communication back and forth. To aggregate the closeness-centrality of all participants in the discussion into a single value, we use the normalized entropy measure. We need this normalization, as discussions do not all have the same amount participants.

C. Measures of Communication Dynamics In addition to information about the discussion content and the actors involved, we can measure the dynamics of a discussion. In the following we describe the six measures of communication dynamics we used in our study.
1) Number of Messages: By their very nature, issues that are complex, not well understood, or controversial require a greater amount of communication than simple problems. We represent this idea by a measure of the amount of messages (NMSG) exchanged in a discussion.

2) Length of Messages: In addition to this quantitative measure, we define two qualitative measures: first, the number of words in a discussion (DLEN), and second discussion length entropy (DLENE). We consider the “wordiness” of messages as an indicator for the cognitive complexity of the reported issue and greater fluctuations of wordiness (resulting in a higher measure of entropy) as an indicator for possible communication problems.

3) Reply Time: Cognitive sciences define communication as “the sharing of meaning” [1], [11]. The absence of communication for an extended period of time, or distorted communication, can be the cause of misinterpretations and misunderstandings. In the context of software development, such misinterpretations when carrying out changes to the source code can be the cause of errors. We capture this idea by measuring the mean reply time between messages (REPLY), and the reply time entropy (REPLYE) for discussions.

4) Interestingness: The BUGZILLA system allows users to get automatic notifications when an issue report is changed, via so-called “CC-lists”. We use a measure of the number of people who signed up for such notifications as an indicator the interestingness (INT) of an issue report. This measure is different from the number of participants, since users of the issue tracking system can be on the notification list, while not contributing to the issue report discussion. In addition we capture the variability of interestingness in a measure of interestingness entropy (INTE).

How closely community members interact, associativity, measures preference to connect to other community members.

- [[avgUserCollaborationFiles-**INSERT PICTURE HERE**]] worked together on files : files changed / which users
- [[avgCCFollowers-**INSERT PICTURE HERE**]]are following/followers for each other

3.2 Distribution of responsibilities

Measures how responsibilities, collaboration and communication are distributed across community members, team characteristics for realizing benefits

The data can be used to determine if complex network characteristics are also present in open source software development. We can analyze whether new links (developers) in this network are attracted to the largest, oldest, or fittest existing nodes, which is known as a common pattern in project teams communities.
Modern applications and systems continuously record their activities and other types of information in various ways, in databases, data warehouse, or simply in the data log files. This type of information is valuable for further processing using log analysis methods. The obtained information can be used to ease the monitoring, management and maintenance of such systems, or to discover the structure of network generated from the log files. Since record attributes contain information about the person who initiated the monitored activity (event), we can derive social

- [[avgBloggers]] number of users that have blogs
- [[avgCommiterLonvigity-**INSERT PICTURE HERE**]] average of committers longevity
- [[avgSubscriptions]] engagement, subscriptions within the project (watchers)

3.3 Task allocation

- [[avgUserCommits-**INSERT PICTURE HERE**]] commits / user
- [[avgFileContributors-**INSERT PICTURE HERE**]] files changed / which users
- [[activeMembers - Not sure that I need it]] active members:those whose total contribution is at least equal to 50%

3.4 Project characteristics

- [[hasWiki]] project having a description Wiki
- [[monthlyEventsStdDev-**INSERT PICTURE HERE**]] Evolution in number of events
- [[ReopenedIssuesPercentage]] Reopened issues: issue was reopened by the actor
- [[milestonesPerDay]] formality : calculating the number of milestones assigned to the project

3.5 Word Frequency

- Word Frequency for different community types

3.5 Social Structure

Discovering the basic characteristics of the network (graph) consists of various metrics [94, 111], like size and density, all types of centralities (degree, betweenness, closeness, eigenvector), clustering coefficient, path analysis (reachability, reciprocity, transitivity and distance), flow, cohesion and influence, and other

We analyze the open source movement by modeling it as a collaborative social network. The developers are nodes of a graph and joint membership on an open source project is a collaborative link between the developers. The open source software development movement is highly decentralized and is a volunteer effort where developers freely join projects that they find appealing all attributes of typical self-organizing systems. We hypothesize that the open source movement displays power-law relationships in its structure.

In general, there are many ways of partitioning given network into communities. It is necessary to establish which partition exihibit a real community structure. Therefore, we need a quality function for evaluating how good a partition is. The most popular quality function is the modularity of Newman and Girivan ***[G-12]****.*

where the sum runs over all pairs of vertices, A is the adjacency matrix, ki is the degree of vertexi i and m is the total number of edges of the network. The element of Aij of the adjacency matrix is 1 if vertices i and j are connected, otherwise it is 0.
The ı -function yields 1 if vertices i and j are in the same community, 0 otherwise.
Modularity can be rewritten as follows.

where nm is the number of communities, ls is the total number of edges joining vertices of community s , and ds is the sum of the degrees of the vertices of s . In the above formula, the first term of each summand is the fraction of edges of the network inside the community, whereas the second term represents the expected fraction of edges that would be there if the network were a random network with the same degree for each vertex. Therefore, the comparison between real and expected edges is expressed by the corresponding summand of the above formula.
The latter formula implicitly shows the definition of a community: a subnetwork is a community if the number of edges inside it is larger than the expected number in modularity's null model. If this is the case, the vertices of the subnetwork are more tightly connected than expected. Large positive values of Q are expected to indicate good partitions. The modularity of the whole network, taken as a single community, is zero. Modularity is always smaller than one, and it can be negative as well. Modularity has been employed as quality function in many algorithms. In addition, modularity optimization is a popular method for community detection.

- [[avgClusteringCoefficient-**INSERT PICTURE HERE**]] [3.AVGCC]
- [[avgDegree]]
- [[modularity]] [3.M]
- [[closenessCentrality]] [3.CC]

4. Case study

Finding patterns is another process used in the data mining to describe the structure of network. Patterns can be defined as structures that make statements only about restricted regions of the space spanned by the variables of the data [49].
These structures might represent Web pages that are all about the same topic or people that socialize together. The discovery of pattern structures in complex networks is an essential and challenging task in many disciplines, including business, social science, engineering, and biology. Therefore researchers have a great interest in this subject and have been approached it differently through data mining methods, social network analysis, etc. This diversity is not limited to the techniques used to implement this task, but it is also applied to its applications.
Data mining techniques are used widely to discover different patterns in data.
The authors of [53] provided an overviewabout the usage of frequent pattern mining techniques for discovering different types of patterns inWeb logs.While in [72] the authors applied different clustering algorithm to detect crimes patterns, and some data mining tools were used in [4] to solve the intrusion detection problem.

5. Discussion

Most of the limitations in the current study are products of the relatively small sample size of build data from the Jazz project combined with the sparseness of the data itself. For example, the ratio of metrics (48) to builds (120) is such that it is difficult to truly identify significant metrics. Whilst various strategies for reducing the number of metrics used in the classification have been investigated, this does not address the fundamental problem that the dataset is very small.

6. Conclusion

The project provides support for collecting data on open-source software projects. The process of collecting metric is applied on a range of **[X]** distinct software repositories. The next contribution is to determine a number of metrics and the correlation between these metrics and the community characteristics. Finally, we describe metric values statistically, getting a first overview of the value ranges for some metrics suits.

Some of the advantages of having access to Github data resources through REST API calls is that it allows fast data retrieval for all repositories that are shared publicly, without being forced to entirely check out this data. However, among the disadvantages we can consider the limit of 5000 requests per hour for authenticated users and the restricted access to data regarding team members and status for users that are not registered as team members.
[]
The obtained results are necessarily preliminary, the set of key-attributes that are frequently associated with open-source communities and the attributes measuring the product quality could be further extended.
[]
We have studied metrics that capture different attributes in the social organization of open-source software development communities.

We have studied measures that capture different structural dimensions in the social organisation of OSS projects. Our analysis is based on a comprehensive dataset collected from the bug tracking communities of 14 major OSS projects. We view the social organisation from the perspective of time-evolving networks and highlight how projects, although simi lar in terms of size, problem domain and age, a) largely differ in terms of clustering coefficie nt, assortativity and closeness centralisation and b) that some projects show interesting dynamics with respect to these measures that cannot be explained by mere size effects. We argue that the phase of high closeness centralisation and low clustering coefficient observed I n the Gentoo community between 2006 and 2008 may be interpreted as a lack of social cohesion which can possibly pose a risk for the project.

While our results are necessarily preliminary, we currently extend our work by adding spectral measures like algebraic connectivity and inequality measures like the Gini index.

We describe an empirical study of the open source projects registered at Github. We believe they are representative of open source movement worldwide. Those projects were modeled as a collaborative social network, with developers as nodes and joint membership in projects as links between the nodes. Analysis of the data displays a heavily skewed distribution, which has a good fit to a power-law relationship. Previous studies of social and collaborative networks with similar properties are believed to grow not as random networks, but as preferentially connected networks. Our study suggests that the same may be true of the open source movement. If this observation is true, then the active research on other such social networks may produce insights that may be applied to further research on open source software.

Several assumptions and limitations are present in the study. We assume that the projects at SourceForge are representative of open source projects in general. This needs to be confirmed. Although we have collected monthly data over 14 months, our analysis only looked at a monthly snapshot of the open source network at SourceForge. Once several years of data are collected, a longitudinal and dynamic analysis may provide better understanding of how node attach and detach from the network. Data on developers who dropped off of projects was not analyzed. We consider only the linking relationship of joint project membership; many of the developers are linked through other relationships, e.g., shared subscriptions to newsletters, listerservs, or reading common web pages. The effect of those other linking relationships, along with the effect of SourceForge itself, should be further investigated. Is the open source movement highly fragmented with SourceForge helping to link those fragments together into a larger connected collaborative cluster?

Additional graph theoretic properties can be computed to provide insight on the nature of the open source network, such as cluster coefficients, degrees of separation, network diameter, etc. This study does not include that analysis, but could be extended as such. Other analytical approaches could include agent-based modeling and simulation as reported elsewhere (Madey, 2002).

References

[1] Analyzing Millions of GitHub Commits, what makes developers happy, angry, and everything in between?, Brian Doll, Ilya Grigorik

[1.2-Walt] Free Software Developers as an Occupational Community:Resolving Conflicts and Fostering Collaboration, Margaret S. Elliott, Institute for Software Research, Walt Scacchi, Institute for Software Research University of California

[1.2-Flossmole] FLOSSmole: A collaborative repository for FLOSS research data and analyses, James Howison, School of Information Studies, Megan Conklin, Elon University, Kevin Crowston, Syracuse University

[1.3-Ahmed] The Road Ahead for Mining Software Repositories, Ahmed E. Hassan, Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Canada

[1.3-Titans] Supporting Community and Awareness Aspects in Open-Source Forges, Damian A. Tamburri, Elisabetta Di Nitto, Simone Gatti, Politecnico di Milano, Roberto Galoppini, Software Engineer, Politecnico di Milano, Patricia Lago, Hans van Vliet

[1.3-ACS] A Case Study of Open Source Software Development: The Apache Server, Audris Mockus, Bell Labs, 263 Shuman Blvd. Naperville, Roy T. Fielding, Information & Computer Science University of California, Irvine, James Herbsleb, Bell Labs, 263 Shuman Blvd. Naperville

[1.4-Grounded] Analysing data using grounded theory Steven R Wainwright MSc, BSc(Hons), PGCE, RGN, is Nurse Teacher, North London College of Health Studies.

[3.AVGCC] Matthieu Latapy, Main-memory Triangle Computations for Very Large (Sparse (Power-Law)) Graphs, in Theoretical Computer Science (TCS) 407 (1-3), pages 458-473, 2008

[3.M] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre - Fast unfolding of communities in large networks (2008)

[3.CC] Ulrik Brandes (2001). A faster algorithm for betweenness centrality

[BigBook] Handbook of Social Network Technologies and Applications, Borko Furht, Florida Atlantic University, Dept. of Comp. & Elect. Engin. and Comp. Sci.

[G-12] Newman, M. E. J., Modularity and community structure in networks, Proceedings of the National Academy of Sciences (PNAS), 103(23), 8577–8582, 2006

[2-Hasan] Predicting faults using the complexity of code changes, Hassan, A.E. ; Software Anal. & Intell. Lab. (SAIL), Queen"s Univ., Kingston, ON

[2-R30] T. Wolf, A. Schr¨oter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in ICSE '09: Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, 2009, pp. 1–11.

[2-R10] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," IEEE Transactions on Software Engineering, vol. 35, no. 6, pp. 864–878, 2009.

[2-R] Studying the Impact of Social Structures on Software Quality, Nicolas Bettenburg and Ahmed E. Hassan, Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University

[2-R27] J. ´ Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in MSR '05: Proceedings of the 2005, International Workshop on Mining Software Repositories. ACM, 2005, pp. 1–5.

[E-Git] https://github.com/eclipse/egit-github

[Gephi-Tool]https://github.com/gephi/gephi-toolkit

[Geo] https://developers.google.com/maps/documentation/geocoding/

[Ghtorrent] GHTorrent: Github's Data from a Firehose, Georgios Gousios and Diomidis Spinellis, Department of Management Science and Technology, Athens University of Economics and Business