Analyzing GitHub as a Collaborative Software Development Platform: A Systematic Review

by

Arturo Reyes López
B.Sc., Universidad Veracruzana, 2004

A Master's Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Analyzing GitHub as a Collaborative Software Development Platform: A Systematic Review

by

Arturo Reyes López
B.Sc., Universidad Veracruzana, 2004

Supervisory Committee

Dr. Daniel M German, Supervisor
(Department of Computer Science)

Dr. Bruce Kapron, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Daniel M German, Supervisor
(Department of Computer Science)

Dr. Bruce Kapron, Departmental Member
(Department of Computer Science)

ABSTRACT

GitHub is a popular social coding site where developers not only host their code and use git functions, but also use social features to communicate, collaborate, and be aware of changes and others' activities. This new paradigm to code together, and the availability of data have given rise to much research studying collaboration from different angles. However, the vast accumulated knowledge about GitHub tends to be scattered and fragmented.

The goal of this study is to collect the available research on GitHub that is focused on identifying the impact of GitHub in software development. The design of the study includes two sections. First, a systematic search in 7 electronic digital libraries was conducted using a defined search protocol, which included a keyword string and exclusion/inclusion criteria. Second, the extraction of data from each publication and manual coding was conducted to define categories of knowledge based on research questions and findings.

The study results show a growing trend in research with an increase in mixed methodology. The preferred data sources for empirical studies about GitHub are the GitHub API and GHTorrent in 72.57% of publications. The study reveals that a group made of 30 researchers publish 45.86% of total research. The research in NorthAmerica represents 26% of publications. The research on GitHub is focused on the evaluation of pull requests and use of issues(30.77%), popular projects characteristics (20.88%), collaboration and transparency (15.38%), developers' roles (9.89%), influence of popular developers (8.79%), quick-start package with guidelines and datasets (8.79%), tools to improve contributions and collaboration (4.40%) and other (1.1%).

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Daniel German for giving me this invaluable opportunity to study abroad, support and guide me through my studies. I am deeply honored to be under your supervision.

Eirini for your friendship, comprehension, support and those constructive conversations that helped me to focus when I used to loose the objective and your priceless support, corrections and guidance throughout this research.

Wendy for being always ready to help and your attentions with my little baby.

Aditi Gupta for your invaluable assistance in defining the methodology and providing corrections to this report.

I would like to thank to my family:

my wife, Cristina for being my eternal companion and walk through this long journey we decided to take to give a better place to our children when even she was not here. Thanks for your love, comprehension and care. This achievement belongs not only to myself, but also to you.

my mother, Angeles for everything, for raising me on your own and teach me honesty, humbleness and being hardworker to achieve whatever I have on mind.

my father, Arturo for your love, financial support and regardless the distance teach me how to persevere regardless how hard is the journey ahead.

my sister, Greetcher for your love and care during hardship in the last year. Thank you for being here with us aunty.

Finally I would like to thank:

Dr. Florence Leclair and Dr. Daniel Warder for providing me the best medical treatment and help me to recover my vision that allow me to finish my studies and have a better life. I am deeply in debt with both of you.

DEDICATION

To my little princess, Maria Cristina and my wife, Cristina.

Chapter 1

Introduction

With over 53 million repositories[1], GitHub[2] is currently the most popular social coding site. Both open source software communities and commercial companies have been increasingly using GitHub – either public or private repositories – to host their code and manage their development projects. GitHub builds on the features of the git version control system, and offers a friendly web-user interface with embedded workflows and social features which leverage collaboration in software development.

GitHub originally became popular with well-known open source projects[3], which identified GitHub as the means to increase contribution and collaboration [80]. However, these project communities migrating from other platforms such as SourceForge need to adapt themselves to a new form of collaboration in software development through a new workflow and social features.

GitHub provides social features in a style that is similar to other social media sites [115]. Users have a public profile which includes personal information and project and activity information for each developer [10]. Users can also subscribe to event feeds by watching projects or follow popular users [107]; this provides awareness of development activities (e.g. pull requests, issues, comments)[31]. In this social coding environment, developers are able to create social networks [10] and make social and technical inferences [32] that can affect the way developers collaborate.

Due to the popularity, social features and availability of data by using either GitHub API[4] or GHTorrent [40], researchers have shown interest in mining infor-

[1] `https://github.com/features`
[2] `https://github.com/`
[3] `http://rubyonrails.org/`
[4] `https://developer.github.com/v3/`

mation from GitHub, as it has become the platform of choice for many open source projects, and is home to a lot of development activity.

Since 2011, when the first quantitative study was conducted by Heller et al. [49], many researches followed up analyzing collaboration in open source, through the use of GitHub data. Currently, the keyword *GitHub* gives more than 10,732 entries in the most popular digital collections in Computer Science [5].

While there is a lot if accumulated knowledge about GitHub, it tends to be fragmented. As research interest on GitHub continues, researchers would have to spend significant time to find the themes that have been discussed through GitHub-based research, and the questions that remain unanswered.

In this report, a systematic scoping review is presented with the aim to collect and organize the vast knowledge the research community has gained through GitHub studies. The following two research questions will guide this report:

- **RQ1. What is the research studying GitHub as collaborative factor?**

  The purpose of this research question is to collect the available research on GitHub and provide the current trend of research and main authors leading the research.

- **RQ2. What are the different topics covered by research on GitHub?**
  This research question is aimed to categorize the identified research on GitHub based on the research purpose.

The systematic scoping review was used to obtain a selection of publications from 2008 (GitHub launch [6]) to 2016 in order to identify the research trend. To avoid any bias during the search, six electronic databases widely used in Computer Science, defined keywords based on GitHub features, and inclusion/exclusion criteria were considered.

The selected papers were analyzed to find emerging themes related to the impact of GitHub in collaboration. Ground theory methodology [25] was used to code all publications in order to identify high-level abstractions which defined the role of GitHub in software development.

The results showed that there is a growth trend in research related to GitHub. Initially, the research in GitHub was based on quantitative methodology since the first

[5] `https://link.springer.com/search?query=GitHub`
[6] 7

quantitave study in 2011 [49] until reaching 76% of total research in 2014. However, since 2013 the mixed methods approach has been more popular in studies, reaching the 47% of total publications for 2016 as the methodology of the studies. In addition, a group of 16 researchers were identified as the researchers leading the studies on GitHub with 31.07% of all publications.

When analyzing the included papers for classification, 7 emerging themes were identified. The most popular is related to the use of pull requests, popular project characteristics and impact of transparency on collaboration through GitHub. Other themes emerged such as influence of popular developers, developers' activities which define the developers' roles in the GitHub ecosystem and tools proposed to increase contribution and collaboration in GitHub. Further, a set of papers included considerations when mining GitHub and presented available datasets for research.

The rest of the report is structured as follows:

Chapter 2 provides an overview of version control systems, SourceForge as the most popular GitHub predecesor, and the most well-known GitHub features. It also introduces the systematic literature review concept, as the means to assist in collecting and organizing current knowledge. In addition, related work is presented.

Chapter 3 presents the systematic literature review methodology and its various components, as well as the research questions aimed to be answered.

Chapter 4 provides that findings of the systematic review. One part is dedicated to presenting aggregated statistics information about the research activity regarding GitHub. The second part presents the organization of the accumulated knowledge; publications have been classified by emergent themes and the most cited papers in the category are discussed in terms of their findings.

Chapter 5 discusses identified threats in this study and discusses the findings of the review and and how they may affect further research.

Chapter 6 contains the summary of the scoping review.

Chapter 2

Background and Related Work

In this chapter an overview of version control systems and their use in software development is provided. In addition, SourceForge is presented as the predecessor of GitHub in open source development, as well as the transition to GitHub and its features. Finally, systematic literature reviews are introduced as the means to gather and organize current knowledge.

2.1 Version Control Systems

Version Control Systems (VCS) are software tools that record changes of files and allow reverting those files back to a previous state (i.e. group of files in a repository) to specific point on time [1]. VCS not only assist in software management through the ability to track the modifications and the author, but also prevent conflict collision of concurrent work without locking current development [2].

Centralized VCS (CVCS) (CVS and Subversion are well-known examples) are VCS that store the development history in a central repository and regularly the majority of them saves only the latest change of the repository at any time. If the local repository crashes or is unavailable, then the development history is lost [83]. The write-access is granted only to a selected group of developers (i.e. core team) [34], and in order to commit changes, the developers need network access to the central repository [83].

As an alternative, Distributed Version Control Systems (DVCS) (e.g. Mercurial and Git) save the complete development history on each local developers' machine

[1] https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[2] `https://www.atlassian.com/git/tutorials/what-is-version-control`

[83]. The majority of the operations are executed locally and faster due to creation of local branches. As a result, developers tend to increase their performance through committing locally [87]. Branching has become simple and it has encouraged a practice called *feature branches* that allows the creation of isolated "sandboxes", either to explore alternative solutions or conduct tests before merging into the main repository. In the DVCS, there is no failure point because the development history is well distributed among the local developers' copies reducing the risk of some disaster scenarios [34].

In 2005, Git was launched to coordinate the development of the Linux Kernel and gradually became the most popular DVCS. An important difference of Git over previous VCS is that developers can select which revision will include an integrated change and maintain a complete history of the graph related to the changes merged, so developers conceptualize changes as software versions instead of differences in files [112].

2.2 SourceForge

In 1999, SourceForge was launched as a web-based service aimed to to leverage Collaborative Software Development for open source projects [7]. SourceForge offers code hosting, issue tracking, follow up discussions, and creation of wiki pages, among other tools. Currently, SourceForge provides compatibility with Git, Mercurial and Subversion and hosts over 430,000 projects with 3.7 million of developers [3].

In 2000's, SourceForge offered for the first time a vast amount of data which became reachable for research purposes [100],[23],[30]. The collection of the data from SourceForge and other forges had limitations and challenges which made them almost impossible to use in research [53]. For instance, researchers had to obtain data either as database dumps provided only by Notre Dame University [29],[100] or through Web spiders which usually introduced noise when parsing and processing due to the data format [53].

As SourceForge was not conceived as a version control, researchers used to complement data with CVCS logs to obtain the development history [29]. Nonetheless, this research approach could be inaccurate at times because only a small amount of developers had access to the central repository and external/casual contributions

[3] `https://sourceforge.net/about`

would be lost. Parsing the mailing lists provided the only means to find out the casual and external contributions. Then, the data extracted from SourceForge and mailing lists was partially useful, but it required a significant effort.

2.3 GitHub

GitHub [4] is a social coding site that launched in 2008 [5]. It is based on the Git version control system, with social media functionality exposed through a friendly web-user interface as social-networking style. With over 53 million hosted repositories [6], including popular open source projects [7],[8], GitHub became the most widely-used hosting service for software development projects. This social coding site integrates a number of social and Git features that together provide transparency on all activities across projects.

2.3.1 Pull-based Model and Code Review

GitHub presented the *"pull-based"* development model [42] in which the incorporation of source code to the main repository is unlocked through implementing a local main repository for each developer. In GitHub, the project's main repository access is shared only among core-developers and contributors have to *fork* the main repository to create a local branch (which is a copy of the main repository). After, developers can work on the software locally and submit commits to their local branches. When the set of changes are ready to be incorporated into the main repository, the contributor issues a pull request which includes the commits. Pull requests are enhanced by adding social media mechanism which allows to expose publicly the pull request to all GitHub users and even create permanent links which include line(s) of the pull request code [32].

Once a pull request has been submitted, project members can review it to make a decision on whether to accept the changes, and if there are other adjustments that need to be made first. In the public code reviews any GitHub member, not only the project owners can provide either general or inline comments in the pull request [121].

[4] `https://github.com/`
[5] `https://github.com/blog/40-we-launched`
[6] `https://github.com/features`
[7] `http://rubyonrails.org/`
[8] `https://jquery.com/`

The pull request itself becomes a social forum to discuss the appropriateness and quality of the contribution. As other social media platforms, GitHub introduces the use of @-mention, to direct messages to specific GitHub users to join the discussion. The @-mention on GitHub is implemented as a notification mechanism, with emails sent to a member whenever they have been mentioned on any artifact.

The project owner or core team members are the ones that have the permission to merge or reject pull requests into the main repository based on the code review process. In addition, GitHub includes, on the main profile page, the users' contributions graph which is a record of contributions they have made to GitHub repositories [9]. This tracking of activities allow developers to build reputation. According to Dabbish [32], this mechanism provides the effect of improving awareness of project and user activities.

2.3.2 Social Features

In addition to code hosting, collaborative code review and issue tracking, GitHub has integrated social features in its design. As other social media sites, when developers open a GitHub account, a public profile is created which includes personal information, personal repositories and activity information of the developer. As the developer's profile is visible to other users, it has become a new form of resume for programmers [10]. Developers can subscribe to receive information by becoming watchers of projects or followers of developers. When subscribing, developers receive an "activity feed" [77] (e.g. pull requests, issues, comments) from those developers or projects with updates for active projects. In addition, GitHub users can *star* their favourite repositories [10], an activity very similar to bookmarking.

Popular users – usually referred to as *"rockstars"* – are followed by a crowd and can attract an important number of *followers* who eventually may be influenced by their actions [107]. GitHub has become a "social and transparent" environment which promotes learning through easy forking and encourages collaboration in software development with the social features available [32].

The boundaries around building software have changed; the practice has moved from sending patches through mailing list to submitting *pull requests*, but also the public code review of pull requests, and the complexity added by GitHub's social

[9] `https://help.github.com/articles/viewing-contributions-on-your-profile/`
[10] https://github.com/blog/1204-notifications-stars

features. This new landscape has attracted the research community's attention to analyze social and technical aspects which affect how developers contribute and interact in this new social coding environment. Since 2011, researchers [49] started analyzing data from GitHub, which included all the developers activities. However, few efforts have been conducted to gather and organize this current knowledge from a large number of researches.

2.4 Systematic Reviews

"Systematic reviews provide a systematic, transparent means for gathering, synthesizing, and appraising the findings of studies on a particular topic or question. They aim to minimize the bias associated with single studies and nonsystematic reviews" [116]. The research protocol defines the search strategy in such a way as to gather as much of the relevant available literature as possible, and provides reasons to justify the exclusion and inclusion of publications. The protocol includes a search string based on keywords aimed to answer the research questions. The keywords usually include boolean operators *AND* and alternative words and synonyms with boolean *OR* operator [2].

Conducting systematic reviews may assist both new and experienced researchers to have a clear picture about the work that has been done in an area and target their future research based on the current studies. Moreover, systematic reviews are a signal of "health", that a field is flourishing and evolving. For instance, several systematic literature reviews have focused on the prestigious MSR (Mining Software Repositories) conference [11]. These literature reviews have surveyed the current research in the area to extract recommendations [50], lessons learned and barriers in replicability[99], and present a brief history of the MSR field [47].

2.4.1 A Note on Related Work

In 2016, Consentino *et al.* [27] conducted a systematic literature review on GitHub-related studies to analyze how researchers have conducted data mining on GitHub repositories; they reviewed over 93 papers using data mined from GitHub and extracted the methods, datasets and limitations of the studies. The authors found several limitations in the analyzed studies such as low level of replicability (i.e. available

[11] http://2017.msrconf.org/

datasets or code to collect data and replicate the studies), poor sampling techniques and scarce variety of methodologies. However, this systemic review [27] considered a broader variety of topics in its search. Several topics such as Android applications, security vulnerabilities and licenses were included because the main purpose of the authors was to evaluate the data mining methodology used in the publications when extracting data from GitHub. In comparison, the systematic scoping review to be presented in this report is aimed to collect the current knowledge body and organize it on themes which would reflect the impact of GitHub in software engineering research.

2.5 Summary

In this chapter, the centralized and decentralized version control systems and their use in software development were reviewed. In addition, it was explained how SourceForge was used in open source projects and acknowledged limitations in research using SourceForge. GitHub and its highlighted features were introduced and why systematic literature reviews can assist to gather and organize existing knowledge. In the next chapter, the details of the methodology used in this study will be presented.

Chapter 3

Methodology

In this chapter, the different types of systematic reviews are explained to understand the selection of publications in the systematic review study on this report. In addition, the research questions and their purpose as well as the search protocol to be followed are presented. Then, in the study selection, the inclusion and exclusion criteria is presented to select the final papers. Finally, the methodology is introduced to understand how the data was extracted and how it was classified into emergent themes.

3.1 Systematic Reviews

There are different types of systematic reviews depending on the purpose of the research, and they vary on breadth of the selection and depth of the analysis [64][89]. If systematic literature reviews do not exist in the field to be studied, it is advisable to conduct scoping and/or mapping reviews to confirm the need of systematic literature review [66]. The following sections will assist the reader in understanding the differences.

3.1.1 Systematic Literature Review

A systematic literature review is aimed to identify, evaluate and interpret all available research related to specific field or phenomenon. The analyzed studies, which contribute to a systematic literature review, are called *primary* studies and the systematic literature review itself is called a *secondary* study [66].

Kitchenham *et al.* [66] has reported important benefits when undertaking a systematic literature review process:

- Summarize the existent research related to a specific topic.
- Identify gaps in current research in order to suggest future research direction.
- Provide a framework to plan new research activities.
- Gather evidence to support or contradict theoretical hypothesis.
- Create new theoretical hypothesis based on current research

Systematic literature reviews must be conducted based on a defined search protocol. Because systematic literature reviews are focused on in-depth analysis, narrow research questions are considered, as well as inclusion and exclusion criteria to detect as much of the relevant information as possible. However, the amount of literature to be analyzed is not as large as other systematic reviews because more importance is placed on quality assessment rather than large amount of research.

3.1.2 Systematic Mapping Review

Systematic mapping reviews are the most basic systematic reviews, and are commonly used in medical research and recently used in Software Enginering [89]. The main focus of systematic mapping reviews is to search in the literature for the best available evidence using a well-defined search protocol to answer the research questions to classify research, conduct thematic analysis and identify publication origin. In systematic mapping studies, the importance of including the breadth of publications in the field is more important than evaluating the quality of individual studies [89]. The mapping reviews present a broad research question in the form of "What do we know empirically about topic X?" and assign the identified literature on a set of categories such as the type of methodology[67]. Usually, such mapping studies provide the visual trend of current research and they are a preliminary stage to conduct full systematic literature reviews.

3.1.3 Systematic Scoping Review

Systematic scoping reviews provide a mapping of relevant literature following a search protocol which is conducted in more depth [79] than full systematic literature reviews.

Similar to mapping reviews, the scoping studies consider broader research questions and do not involve the assessment of quality of the primary studies. In comparison to mapping reviews which tend to be an inventory of research, the scoping reviews sort the publications based on extraction of key issues and themes, summarize and report the findings [4]. The difference with mapping reviews is that scoping studies analyze the 'what' and 'why' aspects to provide a comprehensive and valuable overview which can illuminate future research.

3.1.4 Snowball Method

Snowballing is offered as a complement to systematic reviews; it builds on the references included in a paper and citations to the paper to obtain additional literature [136]. The identified papers are evaluated based on exclusion and inclusion criteria and recursively searching for references and citations in backward and forward mode until no more papers are found. Wohlin [136] claims that the efficiency of a systematic snowballing may provide an alternative option for a systematic literature study instead of searching on several databases.

3.2 Research Questions

Although software engineering research has conducted several systematic literature reviews [50], [99], [47], there is no available systematic review considering the findings on how GitHub is improving collaboration in open source projects. Systematic reviews in any discipline provide us cumulative knowledge to avoid redoing work already done.

This research is aimed to collect (RQ1) existing literature and organize (RQ2) it in order to confirm the need of conducting full systematic literature reviews and identify areas of research where further studies are desirable. The following research questions will identify not only the trend of current research, but also a comprehensive overview of the current knowledge to be considered for new and experienced researchers.

- **What is the research studying GitHub as collaborative factor?**

  To address this question, a systematic mapping review approach was followed to provide an overview of current research on GitHub. This includes research demographics, research characteristics and main authors leading the research. The information was retrieved by applying a predefined search protocol, on well-known electronic research libraries.

- **What are the different topics covered by research on GitHub?**

  This question was addressed through the main part of the scoping review. After identifying the relevant papers to include in the analysis, based on the exclusion and inclusion criteria, information related to research questions, findings, methodology, dataset were extracted for each paper. The papers included in this report investigated GitHub as the main subject of the research and attempted to understand the impact of GitHub on software development. From all papers, the research questions and related contributions were extracted and then classified by using coding process defined in the Grounded theory [25],[102] which regularly is applied to classify diverse information such as interview transcripts, journals or documents.

3.3 Search Protocol

3.3.1 Keywords

The keyword string in systematic reviews is meant to allow finding as much information as possible related to a specific topic. The following query was used in the search engines of the libraries, looking for the keywords in the title and abstract of the publication when the digital library allowed it (e.g. Springer Link search engine does not offer search for only title and abstract [1]). The keyword string included not only GitHub as keyword, but also the main GitHub features used to collaborate and the actors involved (e.g. followers, watchers) in GitHub throughout the collaboration process.

> *"Github AND (Follower OR Watcher OR Fork OR Star OR Issue OR Commit OR Comment OR "Pull-request" OR Pull OR Request OR Merge OR Repositor* OR Project OR Dataset)"*

3.3.2 Databases

The search was conducted using digital collections of publishers and organizations related to Software Engineering and Computer Science and included a period of time

[1] `http://www.springer.com/authors/book+authors/helpdesk?SGWID=0-1723113-12-799804-0`

from 2008 to 2016. The selection of the electronic databases was made based on the analysis of current literature review in Computer Science, consultation with expert researchers, and advice from the subject librarian at the University of Victoria. Table 3.1 shows the number of results during the search using the defined keywords, in each of the digital libraries and libraries are shown in order of search by the author:

Digital Collections	**Count**
IEEE Xplore	152
ACM Digital Library	176
Compendex(Engineering Village) [2]	921
Google Scholar	170
Springer Link [3]	1,292
Wiley Online Library	29
Science Direct	15

Table 3.1: Digital libraries

As mentioned, the selection of the digital libraries was based on analysis of Computer Science publications and consultation with experts and they were included regardless of the number of papers retrieved from each digital library.

3.4 Study Selection

The definition of inclusion and exclusion criteria is often necessary in systematic reviews [68] to ensure a scoping of the best available evidence. The data extraction is not guided by this criteria, but it is used in the data collection to avoid research bias. The definition of the exclusion and inclusion criteria was reviewed by more than one reviewer who have expertise in the area and this process was iterative until the reviewers agreed with all the defined criteria, as suggested in [119].

3.4.1 Inclusion Criteria

The research papers were included in the research collection if they complied with all the inclusion criteria. An explanation of the criteria follows:

[2]Included databases: Compendex, Inspec, Inspec Archive, GEOBASE and Knovel
[3]Discipline: Computer Science and subdiscipline: Software Engineering

1. Title and/or abstract contain the keywords defined in the search string.
2. Research papers published in journals, conference proceedings or book chapters.
3. Research papers published as technical reports, or in magazines **ONLY** if they are cited by other papers.
4. Full publication content is available.
5. Publication date is between 2008 (GitHub was launched [4]) and 2016.
6. After reviewing the title of the paper and abstract, the subject should be about how GitHub features are used for collaboration in software development or how GitHub features have impacted the way developers interact. If any doubt, the revision of full content of the paper should be conducted.
7. Research papers written in English language
8. Research papers referenced by any of the initially extracted papers – following a snowballing [16]– and following the previous criteria.

3.4.2 Exclusion Criteria

The exclusion criteria provided us a guidance to exclude research papers with confidence. The following criteria was considered when discarding the research papers:

1. Research papers published after 2016.
2. Research papers in different language than English language.
3. Research papers addressing GitHub as the means, but not as the main theme (e.g. use of licenses, bugs in android applications)
4. Papers describing how GitHub is used in academic environment.
5. Datamining research not including insights on how GitHub is used in software development. For instance, papers presenting only evaluation of classifiers (e.g. precision, recall, F-Measure, AUC, etc.), sentiment analysis or classifiers not being part of the design of a tool are discarded.

[4] `https://github.com/blog/40-we-launched`

6. Grey literature such as technical reports without citations, white papers, papers under review, not peer review, Masters and PhD theses as well as self-archived papers (e.g. arXiv preprint) regardless number of citations are excluded.

3.4.3 Grey literature publications

A set of publications, which did not meet the third inclusion criterion (i.e. technical reports or magazines with citations) and met the sixth exclusion criterion related to grey literature without citations, were not included in the statistics and analysis of themes. However, these publications met the remaining inclusion criteria. For completeness, the details of the 12 discarded papers are included in the following table:

Source	**Publications**
Technical reports (not cited)	[21], [131]
ArXiv preprint [5]	[72],[26], [139], [18], [109], [17]
White papers	[90], [103]
Papers under revision	[117]
Not peer review	[86]

Table 3.2: Excluded papers

3.4.4 Data Extraction

The selected papers were read in full and the data extracted from each paper were: research questions, findings, research methodology, year and source to retrieve the data from GitHub (e.g. GitHub API, GHTorrent, GitHub Archive). The questions and findings were maintained in the original form to avoid any bias in interpretation. Subsequently, the data was analyzed through qualitative coding to obtain emergent categories. The annotated bibliography is provided as an appendix to this report. In the following chapter, the streams of the research will be presented in more detail.

[5] `https://arxiv.org/`

3.4.5 Coding Themes

Coding was used as the fundamental analytic process based on Grounded theory to categorize the research papers into themes. The coding process followed the three basic types of coding techniques: open, axial and selective [25].

During open coding, basic concepts were extracted (i.e. follower, watcher, fork, star, issue, commit, pull request, merge, dataset) to form simple categories corresponding to each concept. Some papers belonged to more than one category. During the next stage, axial coding, the classification evolves to more appropriate categories by further analysis to find out additional concepts and understand relationships. During this stage, new categories emerged (e.g. continuous integration (CI), @-mention, gist, contribution driven-by commits, guidelines, successful project, popular project, rockstar, developers' activities, developers' roles, community structure, transparency, collaboration, and tools), others disappeared because they were absorbed by a higher category (i.e. star included into new category popular projects, commit and merge included into pull request) and other categories were kept (i.e. follower, watcher, issue, pull request, dataset). This new set of classifications was the basis for a more abstract categorization.

In selective coding, all categories are unified around "core" categories which are more abstract or convey a more general concept. During this stage, classifications with similar concepts were joined. For example, CI, @-mention, issue, gist and contributions in the form of drive-by commits were grouped in the *Managing Pull Requests*. Datasets and guidelines were grouped in the *Mining GitHub* category. The fork category was aggregated with popular projects and successful projects classifications in the *Forking popular projects* classification to include papers focused on projects widely used in the open source community. The watchers, followers and rockstars categories were put together in the *Influencing developers* category; papers there focus on the impact of influence on the developer behaviour. The developer's activities, developers' roles and community structure were grouped under *Developers' roles*, to include research defining the developers' role in the community based on their activities. The categories of transparency and collaboration appeared to be related due to the impact of transparency on collaboration among developers. Then, the category *Collaboration and transparency* was created. The last category, *Tools*, was kept without any change.

3.5 Summary

In this chapter, the different types of systematic reviews were described, electronic libraries selected and criteria to inspect the studies were set up as well as the information that was extracted from each paper. In the next chapter, the findings of the study related to the research questions will be analyzed.

Chapter 4

Results

This chapter addresses the two research questions defined in chapter 3. The goal of the research questions is to gather and identify the trend of current research on GitHub, as well as organize the body of knowledge.

4.1 Statistics of research conducted on GitHub

4.1.1 Electronic libraries search results

The search on electronic libraries, with the keyword string and research protocol, was conducted during September 1-13 2016. The search resulted in 67 included papers, after removing 2688 publications that did not match the inclusion criteria. The inclusion rate was 2.43%. Mainly, the criterion that was not satisfied was the subject of the papers; those that were not focused on the use of GitHub for collaboration were rejected.

The search in the digital libraries was conducted in the order shown in the table 4.1. The first library, IEEE Xplore, provided the greater number of papers. Similarly, ACM Digital Library, the second, yielded similar results. About 80.6% of the publications were found on these two electronic libraries.

In contrast, Compendex (Engineering Village) and Google Scholar libraries presented the majority of papers located in IEEE Xplore and ACM Digital Library, but added 11 more publications together. Springer Link library resulted in the lowest percentage of selected papers due to restrictions in the search engine to limit the search for title and abstract[1] and the order of the search taken during the search.

[1] `http://www.springer.com/authors/book+authors/helpdesk?SGWID=`

The table 4.1 provides the details with respect to each electronic library:

Database	Papers Included
IEEE Xplore	26
ACM Digital Library	28
Compendex	7
Google Scholar	1
Springer Link	3
Wiley Online Library	1
Science Direct	1

Table 4.1: Number of selected papers, by electronic library

By applying the snowball method to the 67 papers, 25 additional publications were added. As a result, the final number of papers found by the keyword string and additional references were 92 selected publications.

4.1.2 Statistics information

What is the rate of publication on research discussing GitHub per year?

The results of this study showed the earliest publication mentioning GitHub to be in 2010, even though GitHub was launched in 2008[2]. This research was an exploratory study conducted by Storey *et al.* [115] that provided an overview of the characteristics of several tools supporting collaboration among developers. The authors introduced GitHub as a "Social Coding Environment" that combines social networking with the power of the Git DVCS. In 2011, Heller *et al.* [49] conducted the first quantitative study on GitHub that provided the visualization of collaboration and influence in GitHub. The authors found most of the development was done in US and Europe. Some South American countries (i.e. Brazil, Argentina and Uruguay) connected more with Europe than North America and the collaboration occurred between developers near each other.

In 2014, the research focused on GitHub grew by almost 50%, compared to the previous year, due to the official presentation of GHTorrent [3] in the MSR challenge [4]. The availability of alternative datasets on-demand made the access to data on GitHub

0-1723113-12-799804-0

[2]https://github.com/blog/40-we-launched

[3]http://ghtorrent.org/

[4]http://2014.msrconf.org/challenge.php

activities easy for the research community. In 2015, the research on GitHub achieved its highest growth in number of publications, with 29.35% of total publications from all years. However, in 2016, the research decreased by 6.52%, which represents a lower level of number of publication (qualitative, quantitative, mixed publications) than in 2014 with 27.17% of total number of publications. The table 4.2 shows the number of papers by year:

Year	**Publications**
2008	0
2009	0
2010	1 [5]
2011	1 [6]
2012	4
2013	13
2014	25
2015	27
2016	21

Table 4.2: Number of selected papers, by year

What research methods are used in GitHub research studies?

In general, the number of publications applying quantitative methods in GitHub research corresponds to 60.87% of the total number of publications. Only 21.74% of publications are using mixed methods (i.e. quantitative and qualitative methodology), while 16.30% of the research studies include only qualitative methodologies. 1.09% of the publications correspond to systematic review studies.

The rate of qualitative research studies has been stable since 2014, with an average of 2 publications per year. On the other hand, the number of studies using quantitative methods increased more than 50% from 2013 to 2014, reaching the highest number of publications using this methodology across all years. In 2016, the number of quantitative studies was 8, the same as in 2013. Since 2013, studies using mixed methods have gone up from 7.69% of the total number of publications, to 47.62%, in 2016. In 2016, the first systematic literature review on GitHub studies [27] was conducted, focusing on analyzing the papers' data collection process and size, as well as whether papers supplied their datasets to the community.

[5]Storey *et al.* [115]
[6]Heller *et al.* [49]

In 2016, there was a decrease in the number of publications that report quantitative research on GitHub, while the mixed methods studies increased by about 40% relative to 2015. Publications with mixed methodologies represent 47.62% of total research in 2016. This trend is in line with [63] that mentioned researchers may need to include qualitative input to ensure validity of the results. Figure 4.1 shows the number of studies by methodology type (i.e. methodology is indicated by color and percentage represents the amount of papers with respect to the total number of publications) through the time:

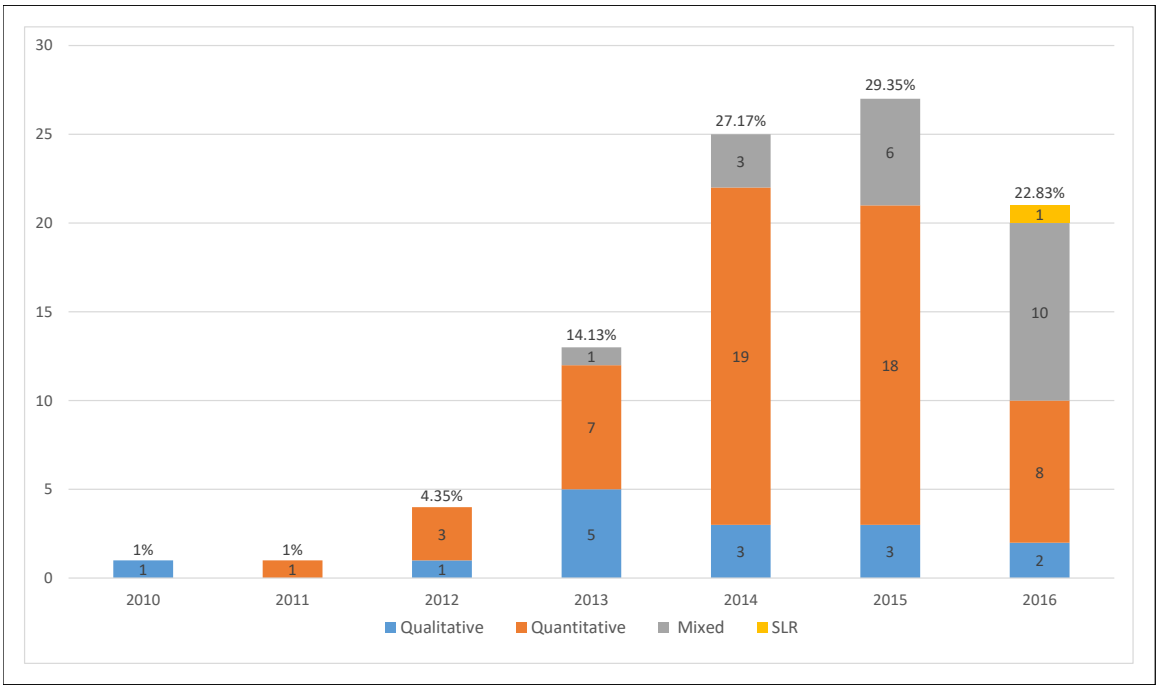


Figure 4.1: Research statistics by year

What datasets are used for GitHub research?

The majority of papers included in the current systematic literature review used either the GitHub API or GHTorrent as datasets (72.57%); this is similar to 74.2% found in previous research [27]. The use of these datasets imposes some limitations to the research. Some of the limitation mentioned in studies [40],[44] include that data in GitHub is additive, important entities are not timestamped, pull requests are merged outside GitHub, and some events may be missing. In addition, the GitHub API limits the requests to up to 5,000 per hour. The use of alternative sources of data, such as GitHub Archive, is only seen in 4.35% of the studies and an additional 4.34% used GitHub Archive with either GHTorrent or GitHub API. Only one paper [104] reported the use of Boa (1.09%) as an alternative source to obtain data. Figure 4.2 shows the number of GitHub studies, per data sources used to obtain data:

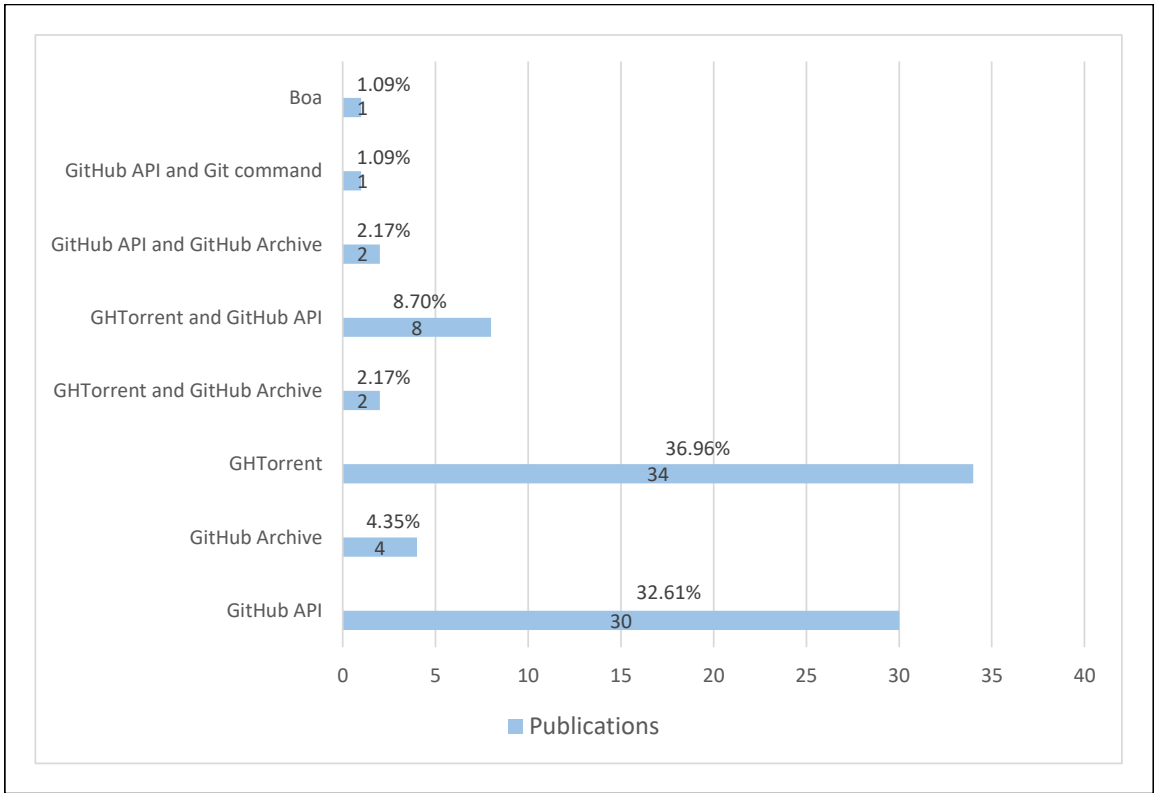


Figure 4.2: Data sources to mine GitHub

With respect to the availability of the datasets to replicate the research, 17.39% of the publications used either a new dataset or available datasets from other research. The remaining 82.61% research included the characteristics of the used dataset, but did not mention how the dataset was collected neither provided any link to replicate

the findings. Similar results are reported in [27] with 31.2% of works sharing the dataset. The figure 4.4 includes the availability of datasets in research:

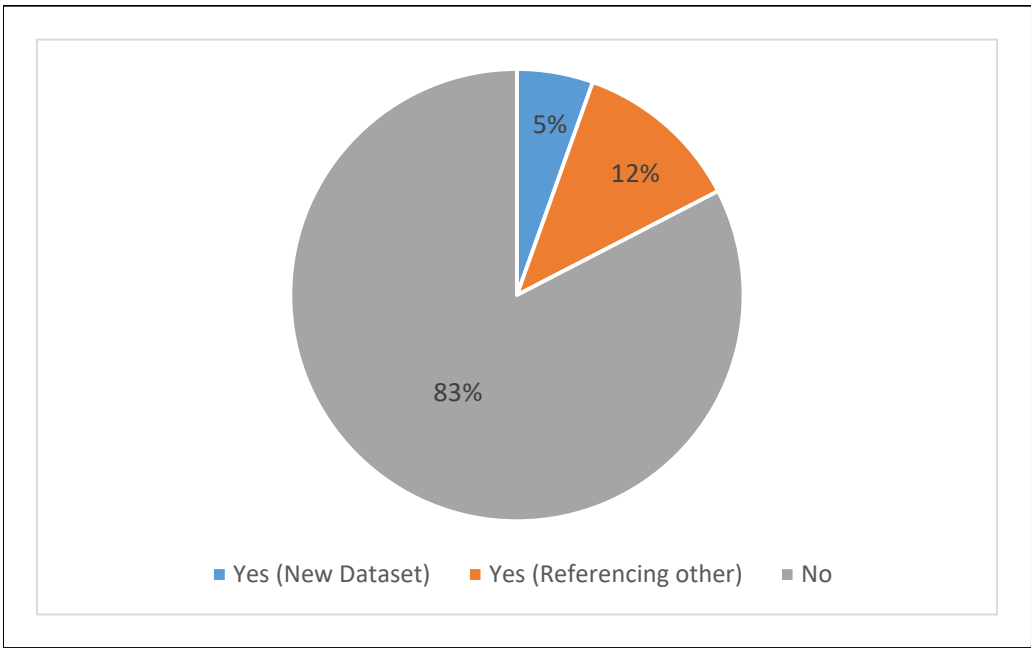


Figure 4.3: Dataset availability for replication

Citations and Authors

The number of citations of the publications were considered as a guide to which research papers have more relevance and impact in GitHub research. When searching for citations for each paper on Google Scholar [7], 16.30% of GitHub publications do not have citations, while research papers with 1 to 4 citations represented 27.17% of total number of papers. In total, 60.87% of the papers have less than 10 citations. Few publications [115],[39],[62] reached above 100 citations and only one work [32] achieved more than 300 citations. These four papers are considered the basis in the evolution of research in GitHub. Figure 4.4 shows the distribution of the amount of papers cited:

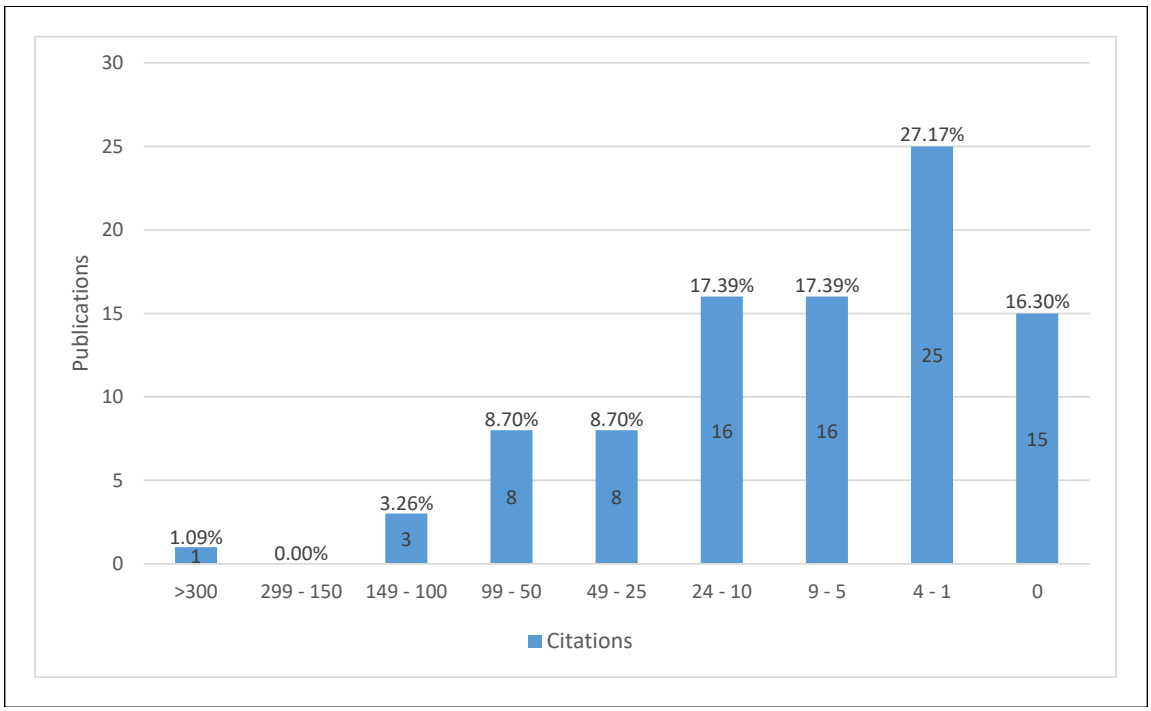


Figure 4.4: Citation statistics

The most active researchers focused on GitHub are presented in Table 4.3, by the number of publications that a researcher is either the first author or coauthor in. This metric may guide new researchers to follow popular researchers, the same way a GitHub developer does. Table 4.3 provides the authors with the highest number of published papers, this set of 16 authors represent 31.07% of the total number of authors focused on the impact of GitHub features on collaboration in software

[7] https://scholar.google.ca/

development:

Author	**Count**
Gousios, Georgios	11
Vasilescu, Bogdan	9
Damian, Daniela	8
Blincoe, Kelly	7
Vladimir, Filkov	7
Wang, Huaimin	7
Yu, Yue	7
Dabbish, Laura	6
Devanbu, Premkumar	6
Herbsleb, James	6
Singer, Leif	6
Cosentino, Valerio	5
German, Daniel M	5
Kalliamvakou, Eirini	5
Tsay, Jason	5
Yin, Gang	5

Table 4.3: Authors with more published research order by number of publications

In table 4.4, it can be observed that the majority of researchers (41.72%) have written one paper while the top-3 authors combined (mentioned in table 4.3) account for 8.28% of the publications on GitHub.

Only a small set of papers stand out for the number of citations. As mentioned before, the majority of papers have less than 10 citations. Table 4.5 includes the top-10 most cited papers in GitHub ordered by the number of citations and year:

When considering where GitHub research originates,it was found that the top-7 most active countries (found through the university affiliation of all authors) gather 53.26% of the total publications. Table 4.6 illustrates the group of countries with the largest number of publications:

Regarding publication venue, about 50% of the papers reporting GitHub research were published in 5 venues: MSR, ICSE, CSCW, APSEC and SANER. Table 4.7 includes the list of conferences with the most published papers on GitHub:

# Papers	# Authors	%	Cumulative
1	141	41.72%	41.72%
2	21	12.43%	54.14%
3	6	5.33%	59.47%
4	8	9.47%	68.93%
5	5	7.40%	76.33%
6	4	7.10%	83.43%
7	4	8.28%	91.72%
8	1	2.37%	94.08%
9	1	2.66%	96.75%
10	0	0.00%	96.75%
11	1	3.25%	100.00%

Table 4.4: Number of authors by n number of papers

Title	Citations	Year
Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository [32]	354	2012
The GHTorent Dataset and Tool Suite [39]	137	2013
The Impact of Social Media on Software Engineering Practices and Tools [115]	122	2010
The Promises and Perils of Mining GitHub [62]	106	2014
An Exploratory Study of the Pull-based Software Development Model [42]	97	2014
Network Structure of Social Coding in GitHub [118]	78	2013
Impression Formation in Online Peer Production: Activity Traces and Personal Profiles in GitHub [77]	77	2013
Creating a Shared Understanding of Testing Culture on a Social Coding Site [91]	70	2013
Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder [10]	66	2013
Influence of Social and Technical Factors for Evaluating Contribution in GitHub [120]	66	2014

Table 4.5: Top-10 most cited papers in GitHub order by number of citations and year

Country	**Paper Count**
USA	14
Canada	10
China	7
Netherlands	6
Brazil	4
France	4
China-USA	4

Table 4.6: Top-7 most active countries order by number of publications

Conference	**Amount**	**%**
Conference on Mining Software Repositories (MSR)	18	19.57%
International Conference on Software Engineering (ICSE)	9	9.78%
Conference on Computer-Supported Cooperative Work & Social Computing (CSCW)	5	5.43%
Asia-Pacific Software Engineering Conference (APSEC)	4	4.35%
International Conference on Software Analysis, Evolution and Reengineering (SANER)	4	4.35%
Computer Software and Applications Conference (COMPSAC)	2	2.17%
Conference on Extended Abtracts on Human Factors in Computing Systems (CHI)	2	2.17%
International Conference Software Maintenance and Evolution (ICSME)	2	2.17%
International Symposium on Empirical Software Engineering and Measurement (ESEM)	2	2.17%
Internal Workshop on Crowd-based Software Development Methods and Technologies (CrowdSoft)	2	2.17%
Joint Meeting on Foundations of Software Engineering (FSE)	2	2.17%

Table 4.7: Popular conferences for publication of GitHub research

4.2 Topics covered by publications on GitHub research

The *research questions* and *contributions* were extracted to analyze the theme of each paper and build abstract categories. Table 4.8 shows the themes that emerged from the coding of the extracted data, from the selected papers. In the last column, the papers of the category are ordered in descending order of number of citations. In cases of ties, the papers are presented in ascending order by year.

Classification	**Papers**
Mining GitHub	[39],[62],[40],[44], [45], [124], [63], [27]
Developers' roles	[128], [85], [88], [94],[78], [76], [133], [71], [126]
Collaboration and transparency	[32], [77], [91], [10], [49], [31], [80], [127], [60], [20], [61], [81], [14],[74]
Influencing developers	[141], [137], [107], [70], [8], [15], [9],[108]
Forking popular projects	[118], [13], [122], [58], [146], [37], [1], [105], [11], [57], [134], [3], [22], [6], [59], [55], [75], [96], [138]
Managing pull requests	[42], [120], [41], [121], [12], [129], [125], [43], [92], [95], [140],[48], [19], [93], [143], [145], [111], [52],[98] [65], [132], [142], [144], [82], [73], [97], [101], [104]
Tools	[56],[123],[114], [5]
Other	[54]

Table 4.8: GitHub classifications

4.2.1 Classification Description

For each theme, there is a sentence describing the category, and a summary of the contents of the category, as represented by the 3 most cited papers in that category.

Mining GitHub. This set of papers provides a glance of guidelines, perils and datasets to be considered when mining GitHub. Gousios *et al.* [39] introduced the GHTorrent dataset and tool suite to support researchers in conducting large-scale research studies using GitHub data, overcoming the restrictions of GitHub's API. The paper, as well as its follow up [40] introduced the schema, design, and implementation of the offline GitHub mirror, which became the dataset of choice

for most of the research papers on GitHub later on. As the number of papers using GHTorrent as the dataset for mining collaboration information grew, Kalliamvakou *et al.* [62] identified potential risks that researchers may face if they don't recognize the specific ways in which the data is organized and how it may affect the conclusions they draw. The authors in [62] caution that mined data should be accompanied by qualitative information to ensure that the conclusions about the degree and nature of collaboration in GitHub projects accurately reflects reality.

Developers' roles. This collection of papers defines the team constituents by classifying the developers' activities and including factors that may either affect performance or contribution. Onoe *et al.* [85] classified a team based on the type of activities, specialization and frequency of activities of developers. The findings revealed that developers play different roles from technical (i.e. coding) to management activities (commenting, creating issues), but rarely their participation are balanced among their different projects. Similarly, Padhye *et al.* [88] defined the team structure as a combination of internal, external and mutant contributors (i.e. only forking the project). The authors in [88] mentioned that the size of external communities in popular scripting languages are comparable to core communities. On the other hand, well-stablished programming languages (e.g. Java, C) have a larger community who do not contribute back. As a complementary study, Vasilescu *et al.* [128] advised increasing gender and tenure diversity in the team structure to improve productivity and promote staffing changes to welcome new ideas from newcomers.

Collaboration and transparency. This category shows how transparency in all developers' activities and project content is used to improve collaboration. Pham *et al.* [91] examined how GitHub assist project owners to communicate testing culture and community's norms. The findings suggested that when contributing to projects that include test suites, newcomers feel "obligated" to include test cases in their contributions. In addition, the authors [91] found that transparency around project artifacts allows novice contributors to adopt easily to community's norms. Moreover, Dabbish *et al.* [32] exposed how developers make social and technical inferences and use this information to collaborate. The transparency assist to project owners to be aware of impacts of incoming contributions and contributors decide to which project contribute. In a similar study, Marlow *et al.* [77] investigated how impressions are formed based on activity history. Developers use available information to form impressions about general coding ability, project management and personality and those inferences can affect contributions and further collaboration.

Influencing developers. This collection exposes the influence of popular users well known as "rockstars" on their followers and watchers. Yu *et al.* [141] found that the social connections among developers form two visible patterns in the follow network in GitHub. In the first, when rockstars or core team are followed by a large number of followers, the projects attract more attention. In the second, some core developers establish external connections with other communities to either collaborate or obtain/share ideas. Similarly, Wu *et al.* [137] spotted that it is common-place for developers in the core team to follow each other. The authors claimed that followship from external contributors is not the result of collaboration in GitHub, but the result of previous connections which are created in external social platforms. With respect to followers, Sheoran *et al.* [107] pointed that overall contributors who were watchers are twice more likely to contribute than contributors who were not watchers before.

Forking popular projects. This group of papers are focused on finding common patterns in popular and successful projects as well as including the characteristics of forked projects. In 2012, Tsay *et al.* [122] conducted a quantitative study with 5,000 projects to define project success based on productivity. The authors found that reducing the need of coordination among developers through concentration of work in small number of developers leads to more contributions. In 2013, Thung *et al.* [118] used 100,000 projects to built project and developer networks to identify the most influential projects on the network. The authors found a long tail distribution where only small number of projects had most of the connections. Libraries, frameworks and language platforms [8] were found to be the most influential projects. Similarly, in 2013, Bissyand *et al.* [13] gathered 100,000 GitHub projects to investigate the programming languages in popular projects. The authors found that scripting programming languages are commonly used in GitHub projects (e.g. Javascript, Ruby, Python and Shell script). Bissyand *et al.* [13] revealed that programming languages affect the popularity of projects. Objective C, related to iOS applications and Ruby, as multipurpose scripting language, were the programming language that attracted more atttention.

Managing pull requests. This set of papers provides insight on which factors determine the acceptance of pull requests as well as the use of other related artifacts (i.e. issues attached to pull requests, gists used in pull requests). Gousios *et al.* [42] conducted the first exploratory study focused

[8] https://github.com/rubinius/rubinius

on pull requests to discover the factors affecting acceptance of pull requests. The most important factors included small number of lines in the code, and contributions relating to recently modified code. In [42] the authors did not find the inclusion of test cases as a factor to influence acceptance. A follow up study conducted by Tsay *et al.* [120] complemented the technical and social factors influencing acceptance. In contrast, this study included test cases as an additional main technical factor. In addition, Tsay [120] mentioned that social aspects such as high status in the community or previous contact with the project owner may increase the possibilities for the pull request to be accepted. Finally, Gousios *et al.* [41] presented a mixed methods study to include a complete guideline to increase the probability to have pull requests accepted. This study includes technical aspects that project owners consider when prioritizing and evaluating pull requests.

Tools. This category includes publications introducing tools for managing collaboration among the team. In 2015, Izquierdo *et al.* [56] introduced Gila, a visualization tool to facilitate the analysis of issues in a project based on label-based categorization. In the same year, Gousious *et al.* [123] introduced the design and prototype of a pull request prioritization tool called *prioritizer* which works as a priority inbox for recommending pull requests. A year later, Stanciulescu [114] introduced a Software Ecosystem (SECO) framework to extract the collaboration between contributors and artifacts and applied this model in an exploratory case study.

Other. This classification includes research with a topic than does not fit with any of the previous themes. Huang *et al.* [54] examined the effectiveness of conflict management strategies to alleviate differences between project owners and contributors.

4.3 Summary

In this chapter, the research statistics in the last 7 years of publication activity on GitHub were presented, to understand the growth trend. It was shown that mixed method studies have been increasing in number since 2013. In addition, the number of studies published in 2016 decreased in 6.52% in comparison to 2015. However, this decrease in the number of publications may be affected by when the searches were conducted for this report.

With respect to gathering data, GHTorrent and GitHub API are mentioned as the most important datasources. North America is leading the research on GitHub,

and 30 authors have contributed to more than two papers. Finally, this section was aimed to classify the vast body of the knowledge to find 7 emerging themes; the most popular ones relate to characteristics of pull requests accepted, what are considered popular projects, and the impact of transparency on collaboration. The following chapter will discuss possible risks in GitHub research and impact of social factors in collaboration. Finally, limitations encountered during the systematic scoping review are acknowledged.

Chapter 5

Discussion and Limitations

In this chapter, some discussion points are raised, relative to research on GitHub and the quality of the studies, as they surfaced from the review of the literature. In addition, two hypotheses are formulated on how social features integrated in GitHub may affect collaboration. At the end of the chapter, the possible limitations of our study are listed.

5.1 Discussion

Blockers to the trajectory of GitHub research

As reported in the previous chapter, research on GitHub presents a general increasing trend. The ratios between quantitative, qualitative and mixed methods studies has changed recently, signalling that the community is using qualitative evidence in addition to quantitative, to understand how GitHub's social aspects influence collaboration. The foundation on which the research on GitHub grew had been the existence of public data and the lack of restrictions on its collection and use. The rich collected data ranged from GitHub user profile data (user name, email) to pull request data (pull-request id, commits, author's email). In addition, initiatives like GHTorrent lowered the barriers to obtaining data through providing offline datasets.

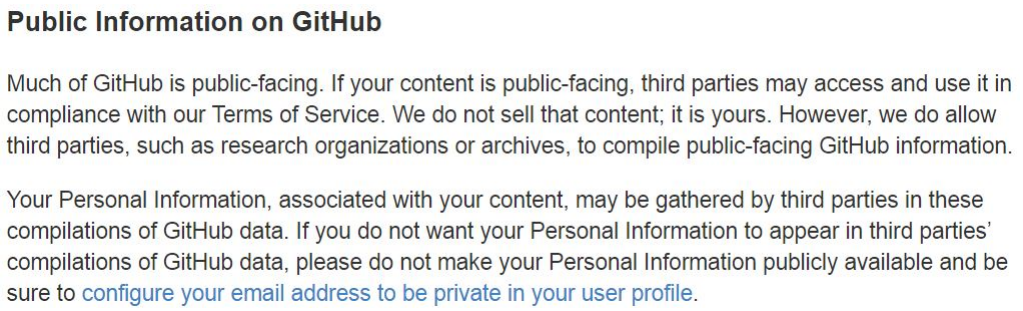
In 2016, the creation of issue 32[1] in GHTorrent showed some concerns by various developers, who requested that their emails be removed from GHTorrent data. As a result, there were community discussions about the public availability of personal data of GitHub developers. The GitHub users' emails, which were made publicly

[1] `http://www.gousios.gr/blog/Issue-thirty-two.html`

available by the users' choice – the default choice at the time – were included in the offline GHTorrent datasets. GitHub developers expressed disapproval of the use of their personal data for research purposes in the comments included in the issue 32[2].

As a result, in March 2016, GHTorrent excluded personal data such as emails and real names from the offline datasets. This restriction in personal data left the only option to researchers to obtain this information through the use of GitHub API.

In February 2017, GitHub Inc. presented a new Terms of Service (ToS) which includes restriction for research only when scraping data; researchers are required to publish their research as open access if they want to use non-personal information [3]. With respect to personal information, the ToS in the *GitHub Privacy Statement* section states the free use of email for research purposes as long as it is publicly available in the user profile:



Public Information on GitHub

Much of GitHub is public-facing. If your content is public-facing, third parties may access and use it in compliance with our Terms of Service. We do not sell that content; it is yours. However, we do allow third parties, such as research organizations or archives, to compile public-facing GitHub information.

Your Personal Information, associated with your content, may be gathered by third parties in these compilations of GitHub data. If you do not want your Personal Information to appear in third parties' compilations of GitHub data, please do not make your Personal Information publicly available and be sure to configure your email address to be private in your user profile.

Figure 5.1: GitHub Privacy Statement

The restrictions in the availability of personal data went further. In addition to the option "Keep my email address private" presented on GitHub to restrict the availability of emails from the user profile and GitHub API endpoints. In April 2017, GitHub launched a new feature [4] to protect even more the exposure of emails included in commits by adding the option "Block command line that expose my email".

Developers on GitHub have started restricting the availability of their personal data, this would bring up two issues that may impact research. First, without the ability to collect emails, researchers are far less likely to accurately link GitHub users to their activity without personal information, which may impact both qualitative and quantitative studies. Second, reaching out to GitHub users to invite them to participate in survey or interview studies becomes more difficult without access to

[2] https://github.com/ghtorrent/ghtorrent.org/issues/32

[3] `https://help.github.com/articles/github-terms-of-service/#5-scraping`

[4] `https://github.com/blog/2346-private-emails-now-more-private`

their email addresses; this can especially limit qualitative data collection. Given these possible difficulties for data collection and processing, it will not be surprising if the trend of GitHub-related published studies starts to decrease.

Replicability of GitHub Research

Replicability is a fundamental characteristic of high-quality research, intended to ensure the validity of findings and avoid research bias. In quantitative research, replicability is somewhat easily achieved by sharing the studied data as well as describing in detail the methods used to process it. On the other hand, qualitative research involves the interpretations and understanding of the researcher, making it difficult to replicate exactly [135]. In addition, qualitative data (such as interview transcripts) are more likely to be confidential and, as a result, make it impossible to share the raw data with other researchers.

In general, about 80% of the studies that ended up being included in the current systematic scoping review, neither made available the dataset used nor provides scripts or methods to extract information. In quantitative research, the inaccessibility to a dataset may create problems or raise questions when interpreting and comparing findings. For instance, Jarczyk *et al.* [57] conducted a quantitative study to analyze popular projects; they found that programming language does not affect the popularity of a project. Another quantitative study, however, by Bissyande *et al.* [13] reported that projects programmed in Objective C and Ruby tended to draw most of the interest, and concluded that programming languages may impact and help define project popularity.

When faced with opposing findings, a replication of the methodological steps on the used dataset by the researchers is a neccesity, to understand which variables made the difference. Without such references, drawing concrete conclusions becomes complicated, and researchers can be unsure which research results to use as knowledge in future studies. One proxy of confidence in a study's findings could be the number of citations it has received. Considering citations, [57] is cited by 5 papers, while [13] is mentioned by 34 authors. Another proxy could be the number of projects the study analyzed. For example, while Bissyande *et al.* [13] included 100,000 projects as their sample, Jarczyk *et al.* [57] considered 2,000 projects, but shared the scripts to obtain the data[5].

[5]https://github.com/wikiteams/supra-repos-x

Given that the majority of research studies do not make available their datasets or replication packages, alternative mechanisms should be proposed to evaluate research under these characteristics. An evaluation of research in a full systematic literature review would need to include additional information such as authors' track of research in GitHub to obtain authors' expertise.

Impact of GitHub's Social Features and Awareness

The categorization of the papers provided in the current review gives some insight on the impact of GitHub on the software development process. Open source communities are made of core developers, active and passive contributors, and the majority of the work is accomplished by a small percentage of developers (core developers) [38]. Previous research on SourceForge provided similar observations with respect to the structure of open source communities. A developer is more likely to get her contribution accepted (in GitHub's case, a pull request) through establishing prior connections with the core team and adhering to project programming standards [35]. Where GitHub has made a difference is that it transformed the traditional developer who had limited awareness, to a *social developer*[110] who has to manage a flood of news and information to support her awareness.

GitHub's social coding environment has incorporated social factors that affect the way developers interact. Previously in SourceForge the mailing list was the only source of awareness. In contrast, developers using GitHub are aware of the project activity even before contributing through the news feed. In GitHub, the developers' contribution and social connections may be more selective and aimed to build faster reputation. Several studies [46], [36], [51], [69] have recognized community recognition as one of the important motivators to participate in open source projects.

In the socio-technical analysis conducted by Ducheneaut et al. [35] in the Python community, the authors found that developers require not only technical skills to obtain community recognition and eventually be part of the core team, but also alliances through building social relationships which support their contributions during controversies. Are the social features and increased awareness in GitHub really providing assistance to newcomers to build easily these relationships and include themselves in the community? It seems that newcomers and external developers may become part of the core team in less time, through using GitHub's social features. However, this aspect hasn't been measured yet in existing research.

GitHub's Social Features may not Be Fully Used

GitHub not only offers a great deal of social features to improve awareness, collaboration and integration in the software development process, but also technical features such as pull-based development. Projects migrating to GitHub have increased the number of collaborators [80]. However, there is no clear evidence that developers may be taking advantage of these features to reduce the time for the first active participation in this social coding environment.

The needed time for first contributions in GitHub seems to be more when compared to the needed time in SourceForge, GitHub's predecessor. For instance, Von Krogh *et al* [130] found out that the average time between first comment and first commit on CVS was about 40 days while Sheoran *et al* [107] obtained the average of 257 days, minimum time among programming languages, for watchers to submit their first pull request. Currently, there is not available research to confirm that GitHub has not only increased the number of contributors, but also decreased the time for first contribution.

Could the large amount of information cause an overload that is delaying the first pull request? There is no doubt that developers on GitHub are exposed to more and richer information than before, and the time to process this information may affect the time to contribute for the first time. In a recent study, [43], most developers reported that they prefer to restrict their attention to news feeds when programming. More studies are needed to study the effect of including different *awareness modes* based on the developers activities to avoid deciding between programming and maintaining awareness.

5.2 Limitations

A possible bias in the results of this systematic literature review may be the selection of relevant papers. This threat of validity was mitigated by conducting a preliminary literature review to obtain the most common keywords in research focused on GitHub and defining inclusion and exclusion criteria validated by experts in the field. In addition, the selection of the digital research libraries could be a threat of validity. However, a preliminary search on the most recognized libraries was conducted in order to assess the range of publications related to the topic. Additionally, the most important digital libraries in Computer Science (IEEE Xplore and ACM Digital

Library) are included in the scoping.

Some important publications may not have been included because they are not hosted in the selected digital research libraries. To address this limitation, the snowball method was used to recover 25 additional studies. In addition, the date, September 2016, of conducting the search on electronic libraries may affect the number of publications for 2016.

A final threat to validity may be the subjective understanding of the research papers, which led to coming up with the specific classification. However, the annotated bibliography of the selected publications is included in the appendix section for the reader consideration.

Chapter 6

Conclusions

In this report a systematic review of studies on the impact of GitHub on software development was presented. By following a systematic methodology, 2,755 papers were identified and 92 publications were selected based on the exclusion/inclusion criteria and snowball method. The overview of what is the research conducted in GitHub and its categorization was presented.

The presented statistics provided the frequency of publication of GitHub-related research, and have given evidence of the different methodologies used, dataset availability, and impactful authors in GitHub-related research. In addition, the collection of research papers were categorized following grounded theory techniques [25] and summarized the most important findings. The resulting categories are: mining GitHub, developers' roles, collaboration and transparency, influencing developers, forking popular projects, managing pull requests, tools and other. In addition, the discussion section compares software development process before and after GitHub to acknowledge important differences, and enlisted some threats for research in the near feature.

This study is aimed to assist new researchers to gain familiarity with the field and recognize new directions to continue studying GitHub's impact on software development.

Appendix A

Annotated Bibliography

In this section, the summary of each paper considered as the dataset in this report is presented as annotated bibliography for further analysis. Each research paper includes classification, objective, questions, findings, methodology and datasets.

Bibliography

[1] K. Aggarwal, A. Hindle, and E. Stroulia, "Co-evolution of project documentation and popularity within github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 360–363.

Forking Popular Projects. This paper investigates the relation between project popularity and documentation updates. **Research Questions:**
RQ1.- How does the popularity of a project correlate with its documentation?
Popularity induces documentation change for some projects. Projects with consistent popularity have more contributors who had some contribution to the documentation.
RQ2.- Do different types of projects exhibit different documentation-evolution behaviours?
Library projects present very small initial documentation that starts increasing as the project's popularity arise. The framework projects include a lot of documentation before popularity.
Methodology: Quantitative (Cross-correlation and linear regression model).
Dataset extracted from MSR'14 Mining Challenge data set. 48 projects.

[2] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2010.

[3] M. Y. Allaho and W.-C. Lee, "Trends and behavior of developers in open collaborative software projects," in *Behavior, Economic and Social Computing (BESC), 2014 International Conference on.* IEEE, 2014, pp. 1–7.

Forking Popular Projects. This paper is aimed to discover trends that govern the developers' decision to select a popular repository to contribute to.

Research Questions:

What are the important project features that make a developer decide whether to participate or not in a project?

1) Technical information (e.g. primary programming language, complexity of the software, project documentation).

2) Popularity and level of activity of the project (e.g. number of watchers and time size).

3) Owner type and project's age: (e.g. project owned by organizations or users).

Strong correlation is found between the number of watchers, open issues and forks. The authors speculated that as the number of open issues increases, the number of watchers and forks increase as well.

After ranking selection, the authors found the following results: - Developers with 2 to 10 participation (newbie developers) look for project complexity (code size) as their top priority in addition to the project popularity (watchers and team size information).

- Developers with 11 to 50 participations consider the programming language, project's age and team size to participate in the project.

- Developers with more experience (51 to 200 participations) take into account team size, programming language and code size to participate in a repository.

Methodology: Quantitative (Descriptive Statistics, Kolmogorov-Smirnov test, Pearson correlation. Feature selection method to rank features based on Information Gain value).

Dataset extracted from GitHub.com (2,332,749 projects and 1,034,996 users).

[4] H. Arksey and L. O'Malley, "Scoping studies: towards a methodological framework," *International journal of social research methodology*, vol. 8, no. 1, pp. 19–32, 2005.

[5] R. Arora, S. Goel, and R. K. Mittal, "Supporting collaborative software development over github," *Software: Practice and Experience*, 2016.

Tools. This paper proposes a novel collaborative software development tool (called COG) over GitHub to provide real-time information about arising direct and indirect conflicts among collaborative developers.
For using COG, developers need only to connect with COG server once and continue working as usual in the Eclipse IDEs, configured for GitHub. Collaborating developers are connected to a central server that stores information about current state of the project hosted over GitHub and on individuals developers' workspace. The server also stores activity information of individual collaborating clients. The authors conducted a case study of conflict identification to demonstrate the effectiveness of the tool.
Methodology: Quantitative(Dependency graphs).
Dataset: None.

[6] J. Aué, M. Haisma, K. F. Tómasdóttir, and A. Bacchelli, "Social diversity and growth levels of open source software projects on github," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 41.

Forking Popular Projects. This paper investigates the relation between project growth as a proxy for success, and social diversity .
Research Questions

RQ1.- How does gender diversity relate to the growth of open source projects?
The Spearman's correlation resulted in less than 0.065 showing no correlation.
RQ2.- How does geographical diversity relate to the growth of open

source projects?
The Spearman's correlation was 0.08 that was higher than gender diversity, but no relevant.
Methodology: Quantitative(Compound Annual Growth Rate (CAGR), Blau index and logistic regression).
Dataset extracted from GHTorrent. Used data provided by the social diversity studies of github teams research.

[7] L. Augustin, D. Bressler, and G. Smith, "Accelerating software development through collaboration," in *Proceedings of the 24th International Conference on Software Engineering*. ACM, 2002, pp. 559–563.

[8] A. S. Badashian, A. Esteki, A. Gholipour, A. Hindle, and E. Stroulia, "Involvement, contribution and influence in github and stack overflow," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2014, pp. 19–33.

Influencing developers. This paper analyzes the members' core contributions, activities and influence in GitHub and Stack Overflow to understand the similarities and differences of the members' contributions in the two platforms. **Findings:**

- As developers become increasingly engaged with the GitHub platform (committing code, contributing to more issues), they accrue more followers.
- More popular users are engaged more in the commits, issues and comments.
- Popularity is not gained through development alone. Other factors such as communications, commenting behaviour or consistency of the activities.
- The activity in one network cannot predict the other in general.
- More active GitHub users are engaged more in answering than asking questions.
Methodology: Quantitative(Spearman correlation, pairwise correlations and Pearson correlation).

Dataset extracted from GHTorrent. 255,375 users.

[9] A. S. Badashian and E. Stroulia, "Measuring user influence in github: the million follower fallacy," in *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*. ACM, 2016, pp. 15–21.

Influencing developers. This papers is focused on understanding how the popularity can influence over different programming languages.

Research Questions:

RQ1.- What does influence mean in GitHub?

- The top 30 list based on number of watchers are owned by individuals who are popular people in SE.

- The top 30 accounts based on number of forks are owned mostly by organizations and specifically they are libraries and frameworks.

The authors refer to the number of forked projects as the most "real influence" metric in GitHub.

RQ2.- How broadly does influence extend? Do the highly reputed users exert influence focused within a theme, such as programming language?

The influence is spread among different programming languages. The number of forked projects is not language dependent but developer dependent.

Methodology: Mixed(Spearman correlation, qualitative analysis of similarities and differences of influential users).

Dataset extracted from GHTorrent. 216 users.

[10] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder," *IEEE Software*, vol. 30, no. 1, pp. 52–66, 2013.

Collaboration and Transparency. This paper provides an insight about how GitHub has been used by developers and the future for GitHub.

Research Questions

RQ1.- In addition to following developers and watching projects, are there other mechanisms for forming relationships?
The URL structure indicates who is the primary maintainer of the project. GitHub put a face to the name of the project's main developer. In addition, user's profile can provide personal information. *RQ2.- How do GitHub help community members find trustworthy content?*
The star is the main feature to show trending repositories. GitHub provides the README file to show exactly how the project works. Projects with good README files tend to be most used.
RQ3.- How did GitHub achieve a critical mass of users on GitHub?
Git is so distributed that there is no centralized place to collaborate. GitHub is the hub where you can host all your Git repositories and work on them together. Big projects such as Rails attracted people. *RQ4.- What future trends do you see that are important to GitHub?*
GitHub on mobile devices such as iPads to cover the ways that people consume content. *RQ5.- If GitHub had 100 times more users and 10 times more resources, where GitHub would go?*
Use GitHub for more activities beyond Sw Development such as Journalism, science through sharing scientific papers for replication.
Methodology: Qualitative (Interview).

Dataset: None.

[11] M. Biazzini and B. Baudry, "May the fork be with you: novel metrics to analyze collaboration on github," in *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*. ACM, 2014, pp. 37–43.

Forking Popular Projects. The authors proposed a methodology to efficiently analyze commits in forks related to the same project.
Research Questions

RQ1.- Are there commits related to a given project that are dispersed in forks other than the project mainline?
Created forks are not kept up-to-date and no new commits are added

to their upstream repository.

RQ2.- Are there differences in the collaboration patterns of multi-repository projects which we can track by analyzing their distributed commit history?

The amount of commits linking forks and mainline is balanced. Some forks are heavily involved, while other very little. Maybe some of these branches are used to develop alternative solutions and they are not shared with other forks.

Methodology: Quantitative(visualization with Circos Software)

Dataset extracted from GitHub API. 342 projects.

[12] T. F. Bissyand, D. Lo, L. Jiang, L. Rveillre, J. Klein, and Y. L. Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2013, pp. 188–197.

Managing Pull Requests. The purpose of this paper is to investigate the adoption of issue trackers among projects with different number of contributors, source code and followers.

Research Questions

RQ1.- What is the proportion of projects that receive issue reports and how can projects be differentiated in that respect?

Projects with smaller code bases are less likely to have issue reports. The median number of developers is 2 for projects without issues while the number is increased to 5 for projects with issues. Projects with reported issues tend to be older, have more lines of code, have more number of developers and have more popular owners.

RQ2.- How many issues are tracked in projects whose trackers are used?

Most projects have small number of reported issues. 86% of the projects have less than 50 issues. Less than 8% of the projects have more than 50 issues in their tracking systems.

RQ3.- What is the number of occurrences of category tags in issue reports? What are the most frequently appearing categories?

Only less than 30% of issue reports in the dataset are tagged. The

two most common tasks are bug and feature. In addition, web applications receive more reported issues which represents 12% of tagged issues.

RQ4.- Who enter issues into issue tracking systems? How many of them are project team members?

Only one third of the developers have written some issue reports for the projects. Additionally, issue reporters often contribute to the code base with 58% of them. Development teams should encourage issue reporters to become active contributors.

RQ5.- What is the relationship between utilization of issue tracking system and project success (i.e. number of forks and watchers)?

The success of a project is proportional to the number of developers that watch the project. There is a strong correlation between the number of issues and the number of watchers and forks.

RQ6.- Does the size of user communities impact the time-to-fix rates of bugs?

The time-to-fix intervals of issues is only slightly impacted by the number of issue reporters.

Methodology: Quantitative (Mann-Whitney-Wilcoxon (MWW) test and Spearman correlation).

Dataset collected from GitHub API. 96,199 projects.

[13] T. F. Bissyand, F. Thung, D. Lo, L. Jiang, and L. Rveillre, "Popularity, interoperability, and impact of programming languages in 100,000 open source projects," in *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, July 2013, pp. 303–312.

Forking Popular Projects. This paper investigates popularity , interoperability and impact of various programming languages in GitHub.

Research Questions

RQ1.- How popular are the various programming languages in terms of adoption in real-world software projects?

- Popularity based SLOC: C ansi (60.835%), Javascript (11.18%),

c++ (8.19%). The authors suggested the predominant use of C is due to OS kernels such as Linux Kernel with above 10,000,000 lines of C code.

- Popularity based on the number of projects on GitHub shows a different list: JavaScript (27.87%), Ruby (19.86%), Python (15.22%).

RQ2.- How many projects are written in more than one programming language and what is the degree of interoperability of each language towards the others?

JavaScript, Shell and Ruby appeared to be used together with most of the programming languages. JavaScript interoperates regularly with PHP and Ruby. C and C++ interoperate the best with each other.

RQ3.- Is there a correlation between the programming language used and the projects success?

- Objective C with 91 followers in average, followed by JavaScript with 65 follower and Ruby with 60 followers are the programming languages with more followers and consequently interest among the projects.

- Measured by number of forks projects in Objective C with 12 forks average, followed by Ruby with 10 forks average and Earlang projects with 9 forks.

The authors suggested this preference for Objective C projects due to iOS (iPhone)applications.

RQ4.- What is the correlation between the programming language used and the number of issue reports?

The authors did not find a clear correlation between programming languages and the number of issues for language.

RQ5.- How does the programming language correlate with the size of the development team?

The top-3 languages (Ansi C, C++ and TCL) included in the dataset have the highest average team size. This results can be affected by highly distributed projects such as Linux.

Methodology: Quantitative (Descriptive statistics and MWW test to assess statistical significance between distribution of projects).

Dataset extracted from GitHub's API. 100,000 projects.

[14] K. Blincoe and D. Damian, "Implicit coordination: A case study of the rails oss project," in *IFIP International Conference on Open Source Systems*. Springer, 2015, pp. 35–44.

Collaboration and Transparency. This paper is focused on how implicit coordination can be measured in modern collaborative development environments by a cases study of Rails project. The authors report how and why features that support implicit coordination are used.

Research Questions:

RQ.-How are the features that enable implicit coordination being used on modern software development environments?

Both issue subscription and following other users are widely adopted. The main reason for subscribing to issues is to obtain information on dependencies (authors called implicit coordination because they reduce the need of direct communication). 48.1% of contributors are subscribed to at least one issue. Developers use issues to be aware of dependencies, issue status, project status, general interest, awareness for future.

Methodology: Qualitative - Online Survey to 7,492 GitHub users and interviews with 14 contributors.

Dataset: GHTorrent. Selected projects with the most code contributions in the GHTorrent 2014-04-02 dataset for Rails. 2,437 closed issues, 7,935 contributors.

[15] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian, "Understanding the popular users: Following, affiliation influence and leadership on github," *Information and Software Technology*, vol. 70, pp. 30–39, 2016.

Influencing developers. This paper studies the motivation behind following others and the influence of popular users on their followers.

Research Questions:

RQ1.- Why do GitHub users follow others and who are the most

followed users?
Getting project updates(27.2%), discovering new projects and trends(24.8%), no benefit(18.8%), learning(11.2%), following friends and co-workers(7.3%), collaboration or sharing code (5.8%).
RQ2.- Are GitHub users influenced by the users they follow?
- Popular user activity on a project precedes the follower's activity for 90.5% of the new projects they star or contribute to.
- When popular users star, fork, contribute to or create a new project, a large percentage of their followers (24.7%, 10.2%, 23.9% and 17%) will star that project.
- 12.5% of the followers will contribute after the popular user star a project and the percentage of followers increase to 13.7% after the popular user contributed to that project.
- The popular user's rate of contribution does not impact their rate of influence and this is an indication that popularity may be more influential than actual contribution.
Methodology: Mixed (survey to 4,000 most active GitHub users (800 responses), chi-squared test, Mann-Whitney test).
Dataset extracted from GHTorrent: 199 users with 101,688 followers.

[16] A. Booth, A. Sutton, and D. Papaioannou, *Systematic approaches to a successful literature review.* Sage, 2016.

[17] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," *arXiv preprint arXiv:1606.04984*, 2016.

Excluded paper. This research is aimed to find the main factors that impact the number of stars of GitHub projects, including programming language and application domain.
Research Questions

RQ1.- How popularity varies per programming language, application domain, and repository owner?
The top-5 languages with more stars are JavaScript, Python, Java, Objective-C and Ruby. Regarding to application domain, the highest

median number of stars are system software, web libraries and frameworks and documentation. The authors considered programming language and application domain a factor may impact on popularity. Repositories owned by organizations have higher popularity than those own by users.
RQ2.- Does popularity correlate with other characteristics of a repository, like age, number of commits, number of contributors, and number of forks?
There is not correlation between number of stars, age, commits and contributors. However, popularity is highly correlated with the number of forks.
RQ3.- How early do repositories get popular?
Around 40% of repositories received 10% of their stars in the first days of the initial release. After this period, for half of the repositories the growth rate tends to stabilize.
Methodology: Mixed (Kruskal-Wallis test, Mann-Whitney test, Spearman?s rank correlation test, KSC algorithm and surveys (5 responses out of 17 developers, 25 answers out of 60 developers)).
Dataset extracted from GitHub API , 2,500 repositories and 97,948 stars.
Dataset available at https://goo.gl/73Sbvz.

[18] H. Borges, M. T. Valente, A. Hora, and J. Coelho, "On the popularity of github applications: A preliminary note," *arXiv preprint arXiv:1507.00604*, 2015.

Forking Popular Projects. *Excluded paper*
This paper introduces a framework to evaluate popularity of projects and a collection of popularity growth patterns.
Findings:

Growth patterns describing evolution of number of stars over the time:
1)*Sustainable growth*: The number of stars received is enough to maintain without variations in popularity. 22% of projects in GitHub belongs to this category.

2)*Fast growth*: Projects maintained an improvement in their popularity compared with the previous week at least during 90% of weeks under analysis. 5% of projects matched this category.
3)*Slow growth*: Projects received few stars on each week and they are not enough to maintain their position and decreased their popularity. A minimum 0.5% of projects in GitHub follow on this growth.
4) *Viral growth*: Projects with a massive growth due to word-of-mouth propagation in social networks. Only 2% of the projects present this growth and the majority are implemented on JavaScript. The majority of project show a sustainable growth with some of them presenting a fast growth.
Methodology: Quantitative(Descriptive statistics, Spearman correlation).
Dataset was not mentioned in the paper. 2,138 projects.

[19] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 550–554.

Managing Pull Requests. This paper is focused on how and which labels are used in GitHub projects.
The GitHub issue-tracking system includes seven generic labels to tag the issues: bug, duplicate, enhancement, help wanted, invalid, question, wontfix. However, it provides the capability to create custom labels.
Research Questions:

RQ1.- How many labels are used in GitHub?
There are only 3% of project containing labelled issues. The label mechanism is scarcely used in GitHub.
RQ1.1.- How many labels are used per project?
The majority of the projects uses one or two different labels. About 60% of the issues are labelled. About 40% of GitHub users are involved in at least one labelled issue. Projects with less than 100

issues use regularly label mechanism.
RQ1.2.- What are the most popular ones?
The most popular labels in decreasing order are: enhancement, bug, question, feature, documentation, wontfix and task. There are some labels that are used together. For example, patterns of label pairs are: bug-question, enhancement-question, wontfix-bug, wontfix-enhancement.
RQ2.- Does using labels influence the evolution of the project?
On average the percentage of solved labelled issues increases with the number of labels used in the project. The effort to categorize the issues is beneficial for the project at the cost of taking time to labelled the issues. Projects with a complex label management tend to involve more people in discussions.
Methodology: Quantitative(Pearson correlation).

Dataset extracted from GiLA (tool which generates visualizations of issues based on label-categorization). GiLA relies on GHTorrent.

[20] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in github: The role of prior social links and language experience," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 817–828.

Collaboration and Transparency. This paper is focused on developer's behaviour when joining a project.
Research Questions:

RQ1.- Do prior social connections matter when developers join new projects? That is, are they more likely to join projects in which there are developers they have already collaborated with in the past?
Over 90% of developers joined projects having past social connections.
RQ2.- How does the presence of past social connections, and their strength, influence initial developer productivity in both familiar (past language experience) and unfamiliar project environments (no past language experience)?

Having prior experience either with the project's programming language or prior social links will increase between 3.7% and 6.2% of productivity for a newcomer. On the other hand, having stronger social connections decreases the probability of being more productive in the initial period by 2%.
RQ3.- What is the effect of past experience and prior social connections on how productive a developer will be overall in a project?
In the long-term contributors social links and prior experience are very important. Developers with previous social connections have much higher probability of performing better with an increase over 29.5% and 54.3%. Social connections is very important factor to recruit more contributors.
Methodology: Quantitative(negative binomial regression).

Dataset extracted from GHTorrent data dump (2014/11/1) 1,255 developers and 58,092 projects.

[21] D. Celińska *et al.*, "Who is forked on github? collaboration among open source developers," Tech. Rep., 2016.

Excluded paper. This paper is focused on finding which characteristics of the repository owners motivates other developers to fork a project.
Research Findings:

1) Developers with excellent programming skills are likely to be highly rewarded in the community, but their projects may discourage other developers from collaboration.
2) Having a project repository written in one of the most popular languages increases the probability of finding new collaborators.
3) People providing valid e-mail addresses and urls to their personal sites reduce the cost of obtaining information about them.
Methodology: Quantitative(Directed graphs, binary logit model and linktest).
Dataset extracted from GitHub through web-scrapping technique.

553,370 active users with at least one repository.

[22] F. Chatziasimidis and I. Stamelos, "Data collection and analysis of github repositories and users," in *Information, Intelligence, Systems and Applications (IISA), 2015 6th International Conference on*, July 2015, pp. 1–6.

Forking Popular Projects. This papers is aimed to understand GitHub user behaviour and projects success factors through providing seven success rules for GitHub projects.

Success Rules:

1) If Open issues= near to 26 && user has made some action in GitHub in the past 4 months && user has a number of followers near to 8, then Project ¿ 1000 downloads with probability 99%

Zero open issues probably reveal an inactive projects while a large number of open issues means either a project went inactive or that is of low quality.

2) If Project's age is near to 36 months && project user has worked on in it in the last 4 months && projects has number of collaborators near to 2 && project owner follows near to 8 other users, then Project ¿ 1000 downloads with probability 81%

Mature projects with small development teams are more likely to be successful. 3) If project age is near to 24 months && user has made some action in GitHub in the past 4 months && project owner has public repositories near to 12 && project owner has number of followers near to 13 and followees near to 8, then Project ¿ 1000 downloads with probability 78%.

Project's maturity, user activity, user number of repositories and user social activity are factors to maintain a project alive.

4) If project age is near to 24 months && user has made some action in GitHub in the past 4 months && project owner has public repositories near to 12 && project owner has number of followers near to 13 and followees near to 8 && number of collaborators near to 2, then Project ¿ 1000 downloads with probability 78%.

5) If (project has wiki && project is 36 month old (mature) && project user has near to 8 followees, then Project ¿ 1000 downloads

with probability 77%
6) If project is 36 months old (mature) && owner is not hirable, then project ¿ 1000 downloads with probability 77%
The characteristics of successful projects include:
Maturity, high activity, support to other users, active owners, owners with small amounts of other projects, small development teams, owner with small amount of followers and followees and owner is not hirable.
Methodology: Quantitative(Apriori and K-Means algorithms).

Dataset extracted from GitHub API. 192,732 projects and 10,153 users.
Data available at http://sweng.csd.auth.gr/githubData.zip

[23] S. Christley and G. Madey, "Analysis of activity in the open source software development community," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on.* IEEE, 2007, pp. 166b–166b.

[24] H. L. Colquhoun, D. Levac, K. K. O'Brien, S. Straus, A. C. Tricco, L. Perrier, M. Kastner, and D. Moher, "Scoping reviews: time for clarity in definition, methods, and reporting," *Journal of clinical epidemiology*, vol. 67, no. 12, pp. 1291–1294, 2014.

[25] J. M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[26] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Three metrics to explore the openness of github projects," *arXiv preprint arXiv:1409.4253*, 2014.

Excluded paper. The authors proposed three metrics to explore the openness of a project: the distribution of the project community, the rate of acceptance of external contributions, the time it takes to become an official collaborator of the project.
Research Metrics:

RM1: Community composition: How the community of the project is composed in terms of project and non-project members?
In average only 13% of the community is willing to contribute to the

code. In general, the number of external contributors could also be regarded as low.
RM2: External contribution analysis: How many external contributions are accepted? How long does it take to evaluate an external contribution?
On average, 59.47% of pull requests are accepted and it takes around 231.70 days to address them. *RM3: Time to become collaborator. How long does it take to become collaborator?*
The median value is almost 5 months. Some projects have a larger value (more than a year) which may indicate reluctance to grant management permission to external contributors.
Methodology: Quantitative(Descriptive Statistics).

Dataset extracted from GHTorrent (MSR2014 Challenge). 108,718 projects, 499,485 developers, 150,362 issues, 78,955 pull requests and 555,325 commits.
Results available at http://atlanmod.github.io/openness/metric-2-external-contribution-analysis.html

[27] V. Cosentino, J. Luis, and J. Cabot, "Findings from github: methods, datasets and limitations," in *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM, 2016, pp. 137–141.

Mining GitHub. This paper studies how research papers have mined GitHub to understand situation regarding analysis and tackle potential perils. The authors applied systematic searches, prunning of non-relevant works and comprehensive forward and backward snowballing processes and identified 93 relevant papers. They pointed at the trade off between using up-to-date data (GitHub API) and curated data from GitHub (GHTorrent). Regarding to the dataset size most of the research is using small-medium size datasets with less than 100 projects. Only one third of the papers make available their dataset or the code to collect the data and replicate the research.
Methodology: Systematic Literature Review.

Dataset: GitHub Research with "GitHub" as keyword avail-

able at https://www.dropbox.com/sh/1lauk9huugckvwt/AACqYNDOs6YbscEUvxD7tscZa?dl=0

[28] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005.

[29] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: Theory and measures," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 123–148, 2006.

[30] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/libre open-source software development: What we know and what we do not know," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 7, 2012.

[31] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Leveraging transparency," *IEEE software*, vol. 30, no. 1, pp. 37–43, 2013.

Collaboration and Transparency. The authors inspect how developers managed and contributed to projects and how they used the social features such as reviewing their feeds, watching projects and following other users.

Research Questions:

RQ1.- What you can see in a transparent environment?

-Infer the activity in projects by the number of commit events in the feed.

-Infer participation by following, watching and commenting events. They are indicators that the community cared about that person, project or action.

-Infer quality and community interest by number of watchers and forks.

RQ2.- How Transparency affects the way work is done?

-Watching commits in among different forks to recruit skilled and committed contributors.

-Forks can show certain user trends when the fork is used with another piece of software.

-The visibility of changes allowed the project owner to discover why

something was no longer working.
- Benefits in learning (e.g. learn from other developers' actions by watching how other people coded, find the hottest new projects through what others were watching) - Problems in Transparency (e.g. transparency broke down when developers needed information and they could not observed due to flooding of notifications, developers reported problems with information overload when watching several repositories or following many developers). Methodology: Qualitative

(Interview with 24 project owners).
Dataset: None.

[32] ——, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1277–1286.

Collaboration and Transparency. This paper examines the value of transparency for large-scale distributed collaborations and communities of practice. **Research Questions:**

RQ1.- What inferences do people make when transparency is integrated into a web-based workspace?
-Technical inferences (e.g. recency and volume of activities signs of commitment and interest, number of followers represents status in the community, number of watchers and forks indicates of high quality). - Social inferences (e.g. project owners recruit developers based on visible information about developers, project owners make decisions about what code to accept in the project after making inferences about the quality of code based on its style, efficiencyand sometimes submitters' competence).
- Learning through observing (e.g. GitHub users learn from watching how someone else coded).
- Reputation management (e.g. developers managed their self-image to promote their work. Heavy users of GitHub are aware of the audience for their actions).
RQ2: What is the value of transparency for collaboration in

knowledge-based work?
- Transparency allowed users to access the content of a project and be aware of the changes in the project. Awareness and visibility supported direct feedback and interaction between project owners and users (micro-supply chain).
- Developers work independently until there was information developer could not directly observe. Thus, they use external communication channels to solve potential problems.
Methodology: Qualitative (Interviews and Grounded approach).Interviews with 24 GitHub users.
Dataset: None.

[33] B. Dagenais, H. Ossher, R. K. Bellamy, M. P. Robillard, and J. P. De Vries, "Moving into a new software project landscape," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1.* ACM, 2010, pp. 275–284.

[34] B. De Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?" in *Proceedings of the 2009 ICSE Workshop on cooperative and human aspects on software engineering.* IEEE Computer Society, 2009, pp. 36–39.

[35] N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, pp. 323–368, 2005.

[36] Y. Fang and D. Neufeld, "Understanding sustained participation in open source software projects," *Journal of Management Information Systems*, vol. 25, no. 4, pp. 9–50, 2009.

[37] K. H. Fung, A. Aurum, and D. Tang, "Social forking in open source software: An empirical study." in *CAiSE Forum.* Citeseer, 2012, pp. 50–57.

Forking Popular Projects. This research is aimed to understand how forking is used in contributions.
Research Questions

RQ1.- How is forking utilised to facilitate OSS development?
Only 14% of forks contributed back to the master fork. The contributions were classified as defect fixes (43%), code enhancement (12%), documentation (7%), integration (2%).
Methodology: Quantitative (Descriptive Statistics).

Dataset extracted from GitHub API, 9 JavaScript projects and 7789 forks.

[38] R. A. Ghosh and V. V. Prakash, "The orbiten free software survey," *First Monday*, vol. 5, no. 7, 2000.

[39] G. Gousios, "The ghtorent dataset and tool suite," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, May 2013, pp. 233–236.

Mining GitHub. This paper presents the dataset details and construction process through showing the data schema. In addition, the challenges that emerge from its use are mentioned.
Dataset extracted from GitHub API.

The data is available at http://www.ghtorrent.org

[40] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, June 2012, pp. 12–21.

Mining GitHub. This paper presents the design and initial implementation of an off-line GitHub mirror and demonstrates how the generated datasets can be queried and processed.
The data showed that the majority of developers reside in the US (43%) or Europe (20.3%).
Methodology: Quantitative(Off-line mirror of GitHub API).
Data set extracted from GitHub API.

[41] G. Gousios, A. Zaidman, M. A. Storey, and A. v. Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in

2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, May 2015, pp. 358–368.

Managing Pull Requests. This paper focused on the role of the pull-request integrator, challenges when evaluating pull requests and decision making process to evaluate them.

Research Questions

RQ1.- How do integrators use pull-based development in their projects?

Integrators use pull requests for doing code reviews, resolve issues and half of the integrators use pull requests to discuss new features.

RQ1.1.- How do integrators conduct code reviews?

75% of projects conducted code reviews on all contributions. Similar percentage of the integrators use inline code comments in the pull requests to do code review. Delegation in evaluating pull requests occurred in 42% of the integrators by using @-mentioned.

RQ1.2.- How do integrators merge contributions?

Integrators prefer to maintain the commit metadata by using the GitHub interface in 79% of the cases to merge a pull request while Cherry-picking and textual patch were avoided in most of the cases. In addition, integrators reported the use of CI to merge or the use of scripts to automate merges.

RQ2.- How do integrators decide whether to accept a contribution?

1) Source Quality and code style matching the project with its documentation. 2) The technical fit of the pull requests.

3) Importance of proposed change with respect to bug fixes or project priorities.

4) Test cases included in the pull requests.

RQ3.- What factor do the integrators use to examine the quality of contributions?

In general, the integrators evaluate the quality manually. Project owners search for changes matching the project's current style, understandable with good documentation and providing clear added value to the project with minimal changes. Pull request contains a clear description, commit organization and commit messages. Test

cases cover the changes. Contributor's reputation.

RQ4.- How do the integrators prioritize the application of contributions?

1) Pull request age, many integrators prefer first-in, first-out approach before applying other prioritization criteria. 2) Urgency by the contribution fixes i) a security issue, ii) a serious new bug, iii)a bug that other projects depend upon.

3) One fifth of the integrators do not prioritize.

RQ5.- What key challenges do integrators face when working with the pull-based development model?

Integrators are struggling to maintain quality considering the total volume. Social challenges include motivating contributors to keep working on the project, reaching consensus through pull requests.

Methodology: Mixed(Pilot survey 21 answers out of 250 emailed integrators. Survey obtained 749 answers out of 3,150 integrators. K-modes clustering algorithm and descriptive statistics).

Dataset extracted from GHTorrent.

[42] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 345–355.

Managing Pull Requests. This paper explores the characteristics of pull-based development.

Research Questions

RQ1.- How popular is the pull-based development model?

Between 2012 and 2013, 14% of the active repositories used pull request and shared repositories are equally used among projects. Pull request usage is increasing, but the number of repositories using pull requests is decreasing.

RQ2.- What are the lifecycle characteristics of pull requests?

The lifetime of pull requests include two states: merged or closed. 60% of pull requests include less than 20 lines of code and are merged or discarded in less than 1 day. However, the 80% are merged within

4 days and 30% of them in one hour. They include in average 3 comments and the code review affects the time to merge the pull request. Including test cases do not affect the time or decision to be merged. Pull request do not receive special treatment based on their origin (coming from contributor or core team). Across GitHub, more than 70% of external contributions are merged.

RQ3.- What factors affect the decision and the time required to merge a pull request?

The decision to merge a pull request depends on whether the pull request modify a recent modified code. The time to merge depends on the developer's track record, the size of the project, test coverage and the openness to external contributions.

RQ4: Why are some pull requests not merged?

27% of unmerged pull requests are closed due to concurrent modifications and 16% are closed due to no interesting changes.

Methodology: Quantitative (Descriptive statistics, Classification algorithms (Random Forest, Logistic Regression, Naive Bayes, Support Vector Machine, decision trees and AdaBoost with decision trees).

Dataset extracted from GHTorrent. 291 projects and 166,884 pull requests.

[43] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 285–296. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884826

Managing Pull Requests. This paper is focused on the work practices of pull request contributors and the challenges they face to have their pull request accepted.

Research Questions:

RQ1.1 What do contributors do before and after coding a PR?

Developers increase their awareness through looking up for open issues related to their changes, checking similar pull requests pro-

cessed recently and discussions. After coding, most developers do not validate similar work has been done in the meanwhile.
RQ1.2: How do contributors assess the quality of their PR?
More than 60% of developers ran automated testing. About 20% of contributors carried out code review by themselves.
RQ1.3 How do contributors communicate about an intended change?
60% of the contributors do not communicate their changes to the core team. Mainly the developers contact the core team by opening an issues describing the problem and fix or by opening a new PR.
RQ2.- What are the challenges of contributing in social coding sites using the pull-based development model?
Social challenge (e.g. lack of response to obtain feedback about the PR, explain rationale), code challenge (e.g. understanding the code base of the project and architecture without enough documentation) are the most popular challenges.
Recommendations to reduce barriers:
1) Improve guidelines on getting started on code conventions, contribution process and communication with the project owners.
2) Promote empathy for new contributors.
3) Faster response for feedback on their work.
4) Clear roadmap to know in advance the direction of the project.
5) To do list with recommendations for newcomers.
Methodology: Mixed(descriptive statistics and survey (760 out of 4,617 prospective respondents)).
Dataset extracted from GHTorrent. 3,400 projects.

[44] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean ghtorrent: Github data on demand," in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 384–387.

Mining GitHub. This paper presents the data dumps on demand offered by GHTorrent. This research lowers the barriers to mine GitHub data.
Methodology: Quantitative.

Dataset extracted from GitHub API.

[45] G. Gousios and A. Zaidman, "A dataset for pull-based development research," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 368–371.

Mining GiHub. This research presented a dataset which included 900 projects and 350,000 pull requests with statistical tool set for further analysis. The selected features included pull request, project and developer characteristics. The authors presented an innovative heuristic to detect pull request merged not indicated in GitHub. Those heuristics were able to identify 24% of merged pull requests.
Methodology: Quantitative (descriptive statistics and datamining)

Dataset extracted from GHTorrent.

[46] A. Hars and S. Ou, "Working for free? motivations of participating in open source projects," in *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE, 2001, pp. 9–pp.

[47] A. E. Hassan, "The road ahead for mining software repositories," in *Frontiers of Software Maintenance, 2008. FoSM 2008.* IEEE, 2008, pp. 48–57.

[48] V. J. Hellendoorn, P. T. Devanbu, and A. Bacchelli, "Will they like this? evaluating code contributions with language models," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 157–167.

Managing Pull Requests. This paper analyzes how reviewers consider conformance to project's code style to be one of the top priorities when evaluating code contributions on GitHub.
Research Questions

Authors defined a PR as debated if this included at least 3 comments based on Gousios research who found that a PR on average receives 2.77 comments. They found that 21% of PR received no comments before being decided, 38% received feedback but no more than two comments, and 41% was debated.

RQ1.- Are rejected PRs less natural than accepted ones?
The more different is the code in the PRs in relation to the code base, the more chances to be rejected. PRs without comments have significantly lower entropies than reviewed ones.
RQ2.- Are more debated PRs less natural?
A larger number of comments is correlated with a greater probability of rejection among PRs with very high entropy.
RQ3.- Does reviewing affect the entropy of debated PRs?
PRs increase in entropy during code review. In general, the contributors of debated PRs were asked to add novel code to their PRs (e.g. , test cases).
RQ4.- Does contributions' naturalness grow with experience?
The PRs from contributors with prior experience have lower entropy.
Methodology: Quantitative(Language Models, entropy, t-test and MWW test).
Dataset extracted using the GitHub API. 22 projects with 7,500 pull requests and over 13,500 commits.

[49] B. Heller, E. Marschner, E. Rosenfeld, and J. Heer, "Visualizing collaboration and influence in the open-source software community," in *Proceedings of the 8th working conference on mining software repositories*. ACM, 2011, pp. 223–226.

Collaboration and Transparency. This paper identifies patterns related to the effect of geographic distance on developer relationships and social connectivity.
Research Findings:

Heavy development between the US and Europe. Some South American countries (Brazil, Argentina and Uruguay) connect more with Europe than North America. Most collaboration occurs between developers near each other.
Methodology: Quantitative(Linked geo-scatter maps, small multiple displays, and matrix diagrams).
Dataset extracted from GitHub API. Dataset including 50,00 users and 40,860 repositories.

Tool available at https://github.com/emarschner/gothub

[50] H. Hemmati, S. Nadi, O. Baysal, O. Kononenko, W. Wang, R. Holmes, and M. W. Godfrey, "The msr cookbook: Mining a decade of research," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on.* IEEE, 2013, pp. 343–352.

[51] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[52] M. Hilton, T. Tunnell, D. Marinov, D. Dig, K. Huang, *et al.*, "Usage, costs, and benefits of continuous integration in open-source projects," Corvallis, OR: Oregon State University. School of Electrical Engineering and Computer Science, Tech. Rep., 2016.

Managing Pull Requests. This paper carries out an in-depth analysis to understand which CI systems developers use, how developers use CI. In addition, the use of pull requests is included.
Research Questions : *Note: Only research questions related to pull requests are included.*
RQ1.- What percentage of open-source projects use CI?
The authors found that 40% of all projects use CI.
RQ11.- Do projects with CI release more often?
Projects using CI release twice faster than those not using CI.
RQ12.- Do projects which use CI accept more pull requests?
The authors found that pull requests submitted in projects using CI are less likely to be merged than pull request without CI usage.
RQ13.- Do pull requests with CI builds get accepted faster?
The use of CI can make integrating pull requests faster. The median time for a pull request to be accepted is 1.6 hours sooner when using CI.
Methodology: Mixed (survey (442 respondents out of 4,508 developers), descriptive statistics, MWW and Fisher?s Exact test).
Dataset extracted from GitHub API. 826 projects and 653,404 pull

requests.
More details about statistics related to the questions can be found at http://cope.eecs.oregonstate.edu/CISurvey/

[53] J. Howison and K. Crowston, "The perils and pitfalls of mining sourceforge," in *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*. IET, 2004, pp. 7–11.

[54] W. Huang, T. Lu, H. Zhu, G. Li, and N. Gu, "Effectiveness of conflict management strategies in peer review process of online collaboration projects," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 2016, pp. 717–728.

Other. This study examines the effectiveness of three strategies (rational explanation, constructive suggestion, and social encouragement) to manage conflicts.
Findings:
- If contributors argued against any reviewers during the PR inspection, they are 16.8% more likely to leave the project.
- Providing rational explanations has no significant effects on the contributor's tendency of leaving the project.
- Responding to the contributors' argument by providing constructive suggestions, the contributor would be 21.7% less likely to leave the project regardless the PR acceptance. The suggestion to fix the PR's issues can help the PR-contributor to modify the PR. As a result, it is more likely that this PR is eventually accepted and conflict is probably solved.
- When dealing with conflicts, social encouragement might be interpreted as just a way to say that their work is not good enough to be accepted.
- Whether the person responding to the argument is project's administrator or not has no significant effect on the contributor's tendency to leave the project.
Methodology: Mixed(Linear SVM model, interviews, survival analysis, Cox proportional-hazards regression model).

Dataset extracted from GitHub API: 170 projects.

[55] J. C. Izquierdo, V. Cosentino, and J. Cabot, "Attracting contributions to your github project," *The Journal of Object Technology*, 2015.

Forking Popular Projects. This paper investigates whether the collaboration facilities provided by GitHub influence the development advance (commits) on the project.
Research Questions:

RQ1: Are collaboration facilities being used in GitHub?
91.90% of projects have never received a pull request and 91.70% have never received an issue. 98.47% of the total projects have only between 0-10 commits from pull requests. 67.40% of projects have 0 watchers and 75.94% projects have never been forked. The authors suggest that the use of GitHub is far from what it would be expected as a social coding site.
RQ2: Is there a relationship between the project success and either the (a) the collaboration facilities or (b) the project interest attributes?
92% of projects included a description file (i.e. readme) with a link to wikis (46%) and/or external websites (50%). They included precise information on the process to follow for all those willing to contribute to the projects (e.g. how to submit a pull request, the decision process followed to accept a pull request or an issue).
Methodology: Quantitative(Spearman correlation).

Dataset extracted from GitHub Archive: 2,126,093 projects.

[56] J. L. C. Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, and J. Cabot, "Gila: Github label analyzer," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 479–483.

Tools. The authors proposed a visualization tool to facilitate the analysis of issues in a project based on label-based categorization.
The tool provides three visualizations to analyze three different

characteristics of labels in GitHub:
1)Label usage identifies the most used labels.
2)User involvement shows the most active users working on each label-based.
3)Typical Label timeline provides the evolution of issues under labels. The paper includes some sample visualizations and the GiLA architecture.
Methodology: Quantitative(Visualization).

Dataset extracted from GHTorrent.
Tool NOT available at http://atlanmod.github.io/gila
Video available at http://youtu.be/qZ1PFBTcs2A

[57] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, "Github projects. quality analysis of open-source software," in *International Conference on Social Informatics*. Springer, 2014, pp. 80–94.

Forking Popular Projects. This paper investigates how project quality and the characteristics of team members is correlated.
Research Questions:

RQ1.- What makes a good GitHub Project?
- Almost 80% of the project popularity is explained by project features and the most significant is the number of forks.
- The project owned by companies and organizations are more popular.
- Programming language has little effect on the projects popularity. However, projects based on CSS styles were more likely to be popular.
RQ2.- What affects the Quality of a Support?
Having developers making many contributions and owning many repositories negatively affect the number of bugs fixed. The authors suggested a piece of advise: try getting into the project focused developers who will concentrate all their efforts on it.
Methodology: Quantitative(Statistical regression and survival analysis (Kaplan-Meier estimates of survival time for issues), binomial

regression).
Dataset extracted from Google BigQuery online tool. 2,000 repositories and GitHub API.
Available script at https://github.com/wikiteams/supra-repos-x

[58] J. Jiang, L. Zhang, and L. Li, "Understanding project dissemination on a social coding site," in *2013 20th Working Conference on Reverse Engineering (WCRE)*, Oct 2013, pp. 132–141.

Forking Popular Projects. The paper is focused on understanding how social relationships between users are used to disseminate projects, attract contributors and increase popularity in GitHub projects. Popularity is defined as the number of developers who participate in the project.

Research Questions

RQ1.- What are the characteristics of social graphs?
Only 12.5% of users have reciprocal relationships between them, while 87.5% of user pairs are connected one-way. In addition, 55% of users are not followed by any of their following in GitHub.
RQ2.- Do social graphs in GitHub support project dissemination?
The minority of the projects attracts many developers. 84.9% of projects have less than 50 contributors , and only 15.1% more than 50 contributors.
RQ3.- How long after the creation of a project do developers participate in it?
About 45% of new developers participate in projects within 10 days and more than 54% of contributors participate in projects after the first 10 days. The growth rate continues without change around 50 days.
RQ4.- How widely and quickly do projects spread in GitHub?
Across all popularity level, between 15% and 22% of followers eventually become contributors of projects.
RQ5.- Do projects spread along social links? What fraction of a project's contributors discover projects through social relationship?
The majority of projects have much more audiences than contrib-

utors. Compared with other contributors, project creators are the most influential in recommending projects.
RQ6.- How long does this dissemination process take?
67% of projects have the diffusion delay more than a day and 27% of projects have the diffusion delay more than a week.
Methodology: Quantitative (Dijkstra spanning tree. Topological structure of social graphs).
Dataset extracted from GHTorrent from January 25th to May 15th 2012. GHTorrent and GitHub API to collect social relationships. 2,665 projects, 747,107 developers and their 2,234,845 social links.

[59] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang, "Why and how developers fork what from whom in github," *Empirical Software Engineering*, pp. 1–32, 2016.

Forking Popular Projects. This study explores whether developers fork repositories for making contributions or building other competitive projects, how developers fork repositories.
Research Questions

RQ1.- Why developers fork repositories?
Submitting pull requests is the main reason for forking a repository. 85.3% of developers forked to fix bugs and 45.1% of developers forked in order to add features. Less than 10% of developers forked to add documentation. *RQ1.1.- How developers find these repositories?*
66% of developers found a repository to fork through external links (blogs, Hacker News, Reddit, Twitter, Facebook). 65.4% used search engine to find them and 42.6% knew about a repository from a friend. Interestengly, GitHub's recommendation is rarely followed by developers.
RQ1.2.-Do developers care about repository owners?
35.2% of developers care about repository owners while 60.5% of respondents do not care about repository owners. Mostly the developers care about the code and repository activeness rather than the owner.

RQ2.- What kinds of repositories do developers fork? Do developers prefer to fork repositories written in a particular programming language?
Developers are likely to fork repositories written in their preferred programming language, JavaScript is the most popular language for 21.9% developers. The second is Ruby, followed by Java, Python and PHP.
RQ3.- From whom do developers fork repositories?
68.4% of developers fork all their repositories directly from creators. Repository forking does not split developers work on different competing and incompatible versions of repositories. 60% of developers update their forks and 31.3% of them submit pull requests.
RQ3.1.-What are some characteristics of attractive owners of repositories that get forked?
Developers prefer to fork repositories from a minority of attractive owners (the top 5%). Attractive owners contain a higher percentage of organization accounts and have more followers than unattractive owners.
Methodology: Mixed(cumulative Distribution Functions (CDFs) and two rounds of surveys (124 replies out of 1,000 developers, 162 responses out of 3,000 developers, MWW test).
Dataset extracted from GitHub API. 236,344 developers and 1,841,324 forks.

[60] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. M. German, "Open source-style collaborative development practices in commercial projects using github," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 574–585.

Collaboration and Transparency. This paper analyzes how developers in commercial organization use GitHub's features and how collaboration is impacted by their use.
Research Questions

RQ1.- What practices do commercial software projects follow in con-

junction with GitHub's features to collaborate? What is the effect on their collaboration?
GitHub use was primarily for collaboration(62%) and solo projects (37.5%). The common reasons for adopting GitHub are to contribute to OSS or share code (28%) and because the developers used git already (22%).
Effects on development:
A. Working together through independent work (e.g. Fork & pull request workflow is used to isolate development and perform code code before merge it in). B. Reduced communication and coordination needs (e.g. Use of pull request to focus their communication and coordination needs through code reviews and merges). C. A touch of self-organization (e.g. 67% of commercial projects reported using self-assignment to divide tasks among the group members). D. Challenges in collaborating with non-technical members.
Methodology: Qualitative (Initial Survey (240 responses from 1,000 GitHub users) and 30 interviews).
Data extracted from GitHub API.

[61] E. Kalliamvakou, D. Damian, L. Singer, and D. M. German, "The code-centric collaboration perspective: Evidence from github," Citeseer, Tech. Rep., 2014.

Collaboration and Transparency. This paper analyzes how collaboration through coordination, communication, awareness, task division and conflict resolution is benefiting the software development process.
Research Questions:

RQ1: How do DVC-based workflows support collaboration?
- Independent development minimizes coordination needs by forking repositories and branching.
- Visible progress and status increase awareness by providing notifications and integration with chat client.
- Code Reviews highlights coordination needs by pull requests. - Self-organization improves task division by using the public issue

tracking. *RQ2: How does GitHub support collaboration?*
62% of respondents used GitHub primarily for collaboration and 38% used it primarily for solo projects. GitHub highlights when there is need to coordinate activities between members. The tagging functionality alerts developers when they have been mentioned in connection to an issue or commit, drawing their attention when necessary to act upon something. The submission of pull requests signals the need for a developer to review the code and merge it.
Methodology: Qualitative(surveys (240 responses out of 1,000 participants) and interviews with 35 participants).
Dataset extracted from GitHub API.

[62] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 92–101.

Mining GitHub. This paper is aimed to understand the characteristics of GitHub repositories and main features in GitHub. This research shows potential risks and rules of thumb to leverage the data extracted in GitHub. **Research Questions:**

RQ1: What are the promises and perils of mining GitHub for software engineering research?
Peril I: A repository is not necessarily a project. The commits included in the pull-request do not appear in the main repository, they only appear in the repository where the commit was originated.
Advice: The data miner must consider activity in both the base repository and all associated forks.
Peril II: Most projects have very few commits. The most active 2.5 % of project represent the 97.5% of the commits.
Peril II: Most projects are inactive. The median number of days a project is active is 10 days. 32% of projects are active one day.
Advice: Select projects with recent commits and pull requests.
Peril IV: A large portion of repositories are not for software devel-

opment. 63% of the projects are related to software development, experimental (12%) and storage (8.3%) of the projects. Advice: Researcher should review the description and README file of the project.
Peril V: Two thirds of projects are personal. Advice: Researcher should consider the number of committers in a project.
Peril VI: Only a fraction of projects use pull requests and the use distribution is very skewed. Advice: Researcher must consider the number of pull requests.
Peril VII: If the commits in a pull-request are reworked, GitHub records only the commits that are the result of the peer-review, not the original commits. Advise: Researcher must not rely on the commits reported by GitHub.
Peril VIII: Most pull requests appear as non-merged even if they are actually merged. Advice: Research should consider the use of heuristics to detect merged pull requests.
Peril IX: Many active projects do not conduct all their software development in GitHub. Advice: Projects with high number of committers who are not registered GitHub users should be avoided.
Methodology: Mixed analysis including qualitative analysis - Interviews and Survey (240 respondents out of 1,000 users), manual inspection of 434 GitHub repositories and descriptive statistics.
Dataset extracted from GHTorrent available in Jan 2014.

[63] ——, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, pp. 1–37, 2015.

Mining GitHub. This paper includes an extended version of promises and perils.
Perils added:

- Not all activity is due to registered users: Only 1.1% of committers are not registered users.
- Only the user's public activity is visible: GitHub data only is coming from public repositories.
- GitHub's API does not expose all data: For instance, the created_at

field related to watchers does not correspond to the date win which the user became watcher. Researchers may not even be able to have direct access to information. Then, getting additional qualitative input through surveys can help to obtain comments from participants to have a solid source.
- GitHub is continuously evolving.
- The paper includes a comparison between perils identified in other research considering SourceForge and GitHub.
Methodology: Mixed(Surveys and interviews, manual analysis. Descriptive statistics).
Dataset extracted from GHTorrent.

[64] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. sn, 2007.

[65] R. Kikas, M. Dumas, and D. Pfahl, "Issue dynamics in github projects," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2015, pp. 295–310.

Managing Pull Requests. This research is focused on how issues are handled. In specific, how the rate of issue creation, the amount of pending issues and their average lifetime evolve over the course of time.
Research Questions:

RQ1.- What is the issue arrival rate and how does it change over time?
For the first month, a project in GitHub receives on average 19.7 new issues, but one year later, it receives 10.3 opened issues. The monthly rate of opened issues for project decreases over time on average.
RQ2.- How do opened and pending issue members evolve over time?
The median of issue lifetimes varies between 2 to 4 days, thus more than 50% of issues get closed at least 4 days later and in many cases earlier.

RQ3.- What is the average issue lifetime and how does it change over time?
The median issue lifetime is 3.1 days for issues opened in month 0, 4.1 days for issues opened in month 10, and 1.78 for issues opened in month 30. There is slight increase from month zero to month 10 and then slight decrease in the following months.
Methodology: Quantitative(Descriptive Statistics).

Dataset extracted from GHTorrent. 4,024 projects with 967,037 total issues.

[66] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[67] B. Kitchenham, P. Brereton, and D. Budgen, "The educational value of mapping studies of software engineering literature," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, vol. 1. IEEE, 2010, pp. 589–598.

[68] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on software engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[69] K. R. Lakhani and R. G. Wolf, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," 2003.

[70] M. J. Lee, B. Ferwerda, J. Choi, J. Hahn, J. Y. Moon, and J. Kim, "Github developers use rockstars to overcome overflow of news," in *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2013, pp. 133–138.

Influencing developers. This paper explores the effects of social awareness and transparency on developers' contribution to others' projects in GitHub.
Research Questions

RQ1.- Do rockstars' really have a larger influence on their followers

compared to regular developers?
Rockstars' followers performed more subsequent actions on the same project than the followers of regular developers.
RQ2.- Do certain actions by rockstars affect their followers differently?
When a rockstar contributes directly to a project through open issues, pull requests or comments, the followers are performing more actions in the same project compared to when they watch or fork a project.
RQ3.- Does a project's age affect a rockstars' influence on followers?
There is a positive relationship between rockstars' actions on older projects and more subsequent actions by followers.
RQ4.- Does the amount of activity by a rockstar on a project influence followers?
When rockstars perform more actions on their owned projects, they attract more followers to perform actions on the same project.
RQ5.- Do followers use rockstars as guides to projects?
After a rockstar performs any action on a project, a follower also contributes to the project by doing an unrelated action. After a rockstar opens a new issue or comments on an existing issue reported in a project, a follower also commits on the same issue thread.
Method: Quantitative (Descriptive statistics).

Dataset extracted from the GitHub's API. 544 developers and 5,657 projects.

[71] S. Li, H. Tsukiji, and K. Takano, "Analysis of software developer activity on a distributed version control system," in *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 2016, pp. 701–707.

Developers' Roles. The authors proposed a method to analyze the developers' characteristics in terms of strong development areas, contribution, initiative, support and leadership in MVC (Model-View-Controller) projects.
The model includes the following features to search:

-Contribution: Contribution level of developer creating new source code files and modifying other developers' source code files.
-Initiative: The method assigns "initiative" to a developer who often take the initiative in creating and commit configuration files, libraries and management needed for the creation of the system environment.
-Support: A developer performs supportive activities that include modification of other developers' source code files with specific threshold.
-Leadership:It is defined as the mix of the interaction of the developer in Support and initiative classification. For example, High support and low initiative refers to low leadership, but high support and high initiative corresponds to high leadership.
The authors applied their method of feature extraction and suggested the success of their approach by showing the characteristics among sample projects.
Methodology: Quantitative.

Dataset extracted from GitHub API. 4 projects.

[72] A. Lima, L. Rossi, and M. Musolesi, "Coding together at scale: Github as a collaborative social network," *arXiv preprint arXiv:1407.2535*, 2014.

Excluded paper. This paper analysed analyzed interactions in GitHub from different angles: social ties among collaborators, collaboration patterns and contributors' activities with respect to their followers.
Findings:

Followers and Collaborators Networks
The average degree of followers is 3 and onnly 9.6% of users have reciprocal relation. Developers contributing to the same repository do not necessarily follow each other.
Activity, Social Presence and Indirect Rewards

People with higher number of followers are commonly more active. Developers who follow many other developers or watch many repositories are less likely to be more active than those who do not.

Geography of Collaboration

Developers tend to interact with closer developers. Repositories with low number of collaborators tend to have them concentrated around one or more key locations rather than scattered around the globe.
Methodology: Spearman correlation, bipartite graphs.

Dataset extracted from GitHub archive. 345,625 users and 5.68 million repositories.

[73] J. Liu, J. Li, and L. He, "A comparative study of the effects of pull request on github projects," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, June 2016, pp. 313–322.

Managing Pull Requests. This paper explores the differences in projects that use and not use Pull Request in their development process.

Research Questions

RQ1.- Are there any differences in project features between projects using Pull Request and those projects not using Pull Request?
On average, projects using pull requests have a larger number of followers and receive wider attention (number of watchers, forks, stars, contributors' followers and owner's followers)In addition, more contributors are engaged with projects using pull requests. *RQ2.- Are there any differences in bug fixing efficiency between the projects using Pull Request and those projects not using Pull Request?*
The percentage of the bugs taking more than 15 days to be fixed is larger in projects using Pull Requests. The larger bug fixing time in projects using Pull Request may be caused by higher percentage of bugs that takes a long time to be fixed.
Methodology: Quantitative(Spearman correlation, factor analysis and MannWhitney-Wilcoxon test).
Dataset obtained using GitHub API. 461 projects (277 projects using pull requests and 184 not using Pull Requests), 224,581 issues.

[74] J. Longo and T. M. Kelley, "Github use in public administration in canada: Early experience with a new collaboration tool," *Canadian Public Administration*, vol. 59, no. 4, pp. 598–623, 2016.

Collaboration and Transparency. This paper explores the use of GitHub and implications for collaboration in the Canadian public administration.

Research Findings:

- A small number of government departments are experimenting with GitHub and little activity among the groups. For instance, deparments with strong scientific mission (e.g. Natural Resources Canada, Environment Canada, Agriculture Canada) were developing applications for service delivery (e.g. Industry Canada, Revenu Quebec) or promoting open data initiatives (e.g. TBS Open Government initiative, City of Ottawa). The majority of accounts are just created, but not used for collaboration in practice.
- The authors concluded that if individual public servants were recognized for their contributions in collaborative efforts, and given the freedom to contribute to organizational objectives, their incentives to contribute might increase.

Methodology: Survey to 180 GitHub users(32 responses) Semi-structured interviews with 5 Government of Canada public servants. Dataset exracted by internet-based searches and access to GitHub (46 organization acounts related to provincial or territorial governments).

[75] J. F. Low, T. Yathog, and D. Svetinovic, "Software analytics study of open-source system survivability through social contagion," in *Industrial Engineering and Engineering Management (IEEM), 2015 IEEE International Conference on*, Dec 2015, pp. 1213–1217.

Forking Popular Projects. This paper analyses the patterns in GitHub projects to understand how projects survive.

Research Findings:

-The number of forks a project has indicates interest from developers outside the base repository. Having outside help can be important

when a project grows, generating issues and receiving more feature requests.
- Projects with strong interaction between developers and users have better chances of surviving.
Methodology: Quantitative(AdaBost, Random Forest and Support Vector Regressor).
Dataset obtained from GHTorrent (Data retrieved on Dec 10,2014). 2,615 projects.

[76] P. Loyola and I.-Y. Ko, "Population dynamics in open source communities: an ecological approach applied to github," in *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 2014, pp. 993–998.

Developers' Roles. This paper provides a model to understand the population of contributors over time in OSS projects. The approach is based on the use of biological mutualistic models that model the dynamics between contributors and repositories in an OSS ecosystem.
Findings:
High degree of specialization: The parasite is forced to develop sophisticated functionalities to adapt to the host.
Constant interaction leads to stable relationship. A parasite may evolve to become less harmful to its host (developers reducing bugs). Almost 90% of the repositories had mutualistic behaviour. The set of low mutualistic behaviour was related to non-popular projects. Ruby and Javascript projects present highly correlation with this behaviour.
Methodology: Quantitative(Lokta-Volterra equations for mutualism, Root Mean Squared Error (RMSE)).
Dataset extracted from GitHub API. 26 seed repositories.

[77] J. Marlow, L. Dabbish, and J. Herbsleb, "Impression formation in online peer production: activity traces and personal profiles in github," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013.

Collaboration and transparency. This paper investigates how impressions are form around users' expertise based on history of activity across projects, how the available information can influence the openness to contributions?.

Research Questions:

RQ1.- When do people seek out information about unknown others in an online peer production community?

For discovery (i.e. 55% of respondents when receiving a new follower, they look at their profile), informing interaction (e.g. maintainers would look at people's profile when they receive a pull requests), skill assessment. *RQ2.- What information do they use to form impressions?*

General coding ability (e.g. amount of activity, frequency of commits, number of projects owned, languages used), project-relevant skills (e.g. types of visible activity) and personality-interaction style (e.g. past discussion posts and threads).

RQ3: How do interpersonal impressions influence evaluations of others' contributions?

When the submitted pull request is uncertain, project owners evaluated both code-based factors and person-based factors.

Methodology: Qualitative (Interviews to 18 project owners).

Dataset extracted from GitHub API.

[78] N. Matragkas, J. R. Williams, D. S. Kolovos, and R. F. Paige, "Analysing the'biodiversity'of open source ecosystems: the github case," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 356–359.

Developers' Roles. The authors present an analogy between ecological communities and Open Source Software communities. They investigate if the OSS communities become more productive when increasing member diversity.

Research Questions

RQ1.- Do GitHub communities exhibit diversity and structure? If

so, how does this structure look like?
GitHub communities have a structure that is made of three main types of users: core developers, active users and passive users.
RQ2.- Is the structure of GitHub communities similar to the structure of other open source software communities hosted in other forges?
The three layer structure revealed is similar than the onion-shaped structure of OSS projects. Only small percentage of the members are core developers.
RQ3: Does the size of a community affect its structure?
The structure is not affected by the project size. As projects get larger, the percentage of core developers remains stable.
Methodology: Quantitative(Cluster analysis, k-means clustering algorithm).
Dataset extracted from MSR challenge, GHTorrent 21 projects and 23,774 users.
Code available at https://github.com/ossmeter/msr14-challenge

[79] N. Mays, E. Roberts, and J. Popay, "Synthesising research evidence," *Studying the organisation and delivery of health services: Research methods*, pp. 188–220, 2001.

[80] N. McDonald and S. Goggins, "Performance and participation in open source software on github," in *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2013, pp. 139–144.

Collaboration and Transparency. The authors conducted qualitative studies to understand how the increased awareness influences the participation, performance and success in GitHub.
Research Findings:

RF1: Developers infer success on OSS projects by the number of contributors.
RF2: GitHub lowers barriers of participation, some projects have increased the level of participation since moving from other platforms to GitHub.
RF3: Discussion of pull requests provide better visibility and allows

public discussion to direct the decision. As a result, the process of integrating new features is more transparent.
RF4: Developers consider number of commits, number of forks and stars as indicators of the success of the project.
Methodology: Semi-structured interviews with 10 lead and core developers on three open source projects.
Dataset: None.

[81] I. Mergel, "Open collaboration in the public sector: The case of social coding on github," *Government Information Quarterly*, vol. 32, no. 4, pp. 464–472, 2015.

Collaboration and Transparency. This paper analyzes the collaboration ties between contributors to US federal government repositories.
Research Questions:

RQ1.- How are government agencies collaborating on software code to improve government operations?
Government agencies are using GitHub for sharing software code for web platforms and open data platforms, research data and algorithms for scientific papers, technological solutions, health data sets, Geospatial databases, intelligence datasets. Most government-to-government interactions are focused on large-scale projects. The most active organizations are related to national priority such as national security.
RQ2.- how and why are government agencies sharing code, who reuses or contributes to existing open source projects shared on GitHub, and what types of code is shared on GitHub?
Many agencies reuse code already developed in other parts of the government to learn from existing innovations. They reuse libraries and utilities that allow an agency to adapt the code to their own needs. Similar to open source communities, the content is evaluated as well as developer reputation. Discovering issues, either error or improvements is less frequently observable in the U.S. federal

government at this stage.
Methodology: Mixed(Node degree centrality measure, semi-structured interviews).
Dataset extracted from GitHub API: 357 organizations (192 from U.S. , 107 are government organizations from other countries and 58 were civil organizations working on government projects).

[82] L. Murta, A. Plastino, *et al.*, "Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 960–965.

Managing Pull Requests. This paper investigates the factors that lead to rejection of internal contributions.
93.42% pull requests were accepted and 6.58% were rejected.

Rules:
1) When submiting a pull request for the first time and modifies many files, his contribution will be rejected 37% of the time.
2) When submitting a pull requests for the first time including many commits, his contribution will be rejected 39% of the time.
3) The lack of experience, the volume of commits and modified files increases 7 times the chances of rejection.
Methodology: Quantitative(Descriptive statistics and manual inspection).
Dataset extracted from GHTorrent. 20,140 pull requests of seven projects.

[83] K. Muslu, C. Bird, N. Nagappan, and J. Czerwonka, "Transition from centralized to distributed vcs: A microsoft case study on reasons, barriers, and outcomes," in *Proceedings of the International Conference on Software Engineering*, 2014.

[84] V. Myllärniemi, M. Raatikainen, and T. Männistö, "A systematically conducted literature review: quality attribute variability in software product lines," in

Proceedings of the 16th International Software Product Line Conference-Volume 1. ACM, 2012, pp. 41–45.

[85] S. Onoue, H. Hata, and K.-i. Matsumoto, "A study of the characteristics of developers' activities in github," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 2. IEEE, 2013, pp. 7–12.

Developers' Roles. This paper analyzes the types of developers based on actual developers' activities.

Findings

1) *Type of activity* (e.g.coding, commenting and issue handling): The majority of the activities are related to commenting followed by coding.

2) *Specialization of contributions*: Top contributors contribute differently to target and other projects. Regularly the developers focused their attention for development activities to a small set of projects, comment activities to different set of projects.

3) *Workdays:* Some developers work mostly on weekdays, while others work mainly on weekends.

4) *Frequencies of Activities:* Some developers produce 300 events in few weeks while others took months for 300 events.

Methodology: Quantitative (Descriptive Statistics).

Dataset extracted from GitHub API. Collected data on August 14 2013. Dataset consisted of actions related to 2 projects and 19 developers.

[86] M. Ortu, G. Destefanis, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli, "How diverse is your team? investigating gender and nationality diversity in github teams," PeerJ Preprints, Tech. Rep., 2016.

Excluded paper. This paper is focused on investigating the the impact of gender and nationality diversity on the productivity (i.e. the average time required to resolve an issue) and collaboration quality of a team)

. **Research Questions:**

RQ1.- Are gender or nationality diversity linked to the issue fixing time of a team?
The results showed that gender diversity is link with lower issue fixing time. Country diversity did not have a significant impact.
RQ2.- Are gender or nationality diversity linked to the overall politeness of a team?
The results showed that country diversity is very dominant with a tendency to lower politeness. On the other hand, gender diversity is not significant in the model.
Methodology: Quantitative(modularity algorithm, logistic regression).
Dataset extracted from GHTorrent: 33,673 issues, 13,872 developers, 8,040 projects and 1,176 different teams.

[87] S. Otte, "Version control systems," *Computer Systems and Telematics, Institute of Computer Science, Freie Universität, Berlin, Germany*, 2009.

[88] R. Padhye, S. Mani, and V. S. Sinha, "A study of external community contribution to open-source projects on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 332–335.

Developers' Roles. This paper analyzes the levels of participation from different communities perspectives: core, external and mutant developers.
Research Questions

RQ1.- What is the distribution of relative sizes of communities (CORE, EXTERNAL and MUTANT) in the root projects considering (1) number of users in each community and (2) number of commits contributed by each community?
Most of the times, the CORE COMMITTERS are in the same size than EXTERNAL COMMITTERS. In few projects only CORE COMMITTERS are contributing while in others, the EXTERNAL COMMITTERS contributing more than CORE COMMITTERS.
RQ2.- Is the distribution of relative sizes of communities impacted by the main programming or scripting language used by the root

project?

The number of commits by CORE-COMITTERS always dominated the number of commits in comparison with the other two communities. However, in some cases MUTUAL COMMITTERS are higher than EXTERNAL-COMMITERS. For example, C++, CSS, Java, JavaScript, PHP and Ruby.

RQ3.- Are the communities that are contributing to these open-source projects geographically diverse or concentrated?

In general, the United States dominance represented more than 50% of CORE COMMITTERS and EXTERNAL COMMITTERS. Germany (about 10%)is the second country for both the CORE and EXTERNAL communities.

RQ4: What is the nature of external contribution (e.g. bug fix, feature enhancement or documentation) and do the maintainers of root projects have a preference of either type in deciding whether or not to incorporate such external contribution or to reject it?

Only 45% of pull requests were merged. 67% of pull requests related to bugs were merged. Similarly, 60% of pull requests related to documentation were accepted.

Methodology: Quantitative (Ring-based visualization, descriptive statistics).

Dataset extracted from GHTorrent contains 89 projects, 23,237 users, 548,299 commits.

[89] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering." in *EASE*, vol. 8, 2008, pp. 68–77.

[90] K. Peterson, "The github open source development process," Technical report, Technical report, Mayo Clinic, Tech. Rep., 2013.

Excluded paper. This paper shows how GitHub has influenced aspects of traditional OSS development, such as developer hierarchies and issue close velocity.

Research Questions:

RQ1.- Are GitHub project primarily focused around a small set of

core Committers?

Large percentage of developers committed small amounts (0 - 5%)of the total number of repository commits (external developers). Developers contributing 95 - 100% of the total commits are small (internal developers). However, the notion of a small set of core committers is strongly supported.

RQ2.- How is GitHub changing the OSS process?

National Center for Biomedical, National Cancer Informatics Program, NASA and the White House are some organizations that have moved development to GitHub. GitHub moved the open source community away from a permission culture. Prior to GitHub, contributing to an open source project was very difficult and often involved asking permission.

RQ3.- Can GitHub be used for more than code artifacts?

The CTS2 OMG Specification used GitHub issue tracker to track specification changes. Elicitation, Analysis and Negotiation were natural fits for the issue tracker. Changes in specification can be tracked through including the issue number in the commit log.

Methodology: Quantitative(Descriptive statistics).

Dataset extracted from GitHub API: 1,000 repositories Scripts available at https://github.com/kevinpeterson/github-process-research

[91] R. Pham, L. Singer, O. Liskin, F. F. Filho, and K. Schneider, "Creating a shared understanding of testing culture on a social coding site," in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 112–121.

Collaboration and Transparency. This paper is focused on how transparency on GitHub influences developer's testing behavior.

Research Questions

RQ1.- How do project owners asses incoming contributions from contributors?

Project owners treated pull requests differently depending on whether they trusted the developer or not. Perceived size of the

changes influenced the project owner's need for tests. The target of the changes (i.e. if the changes were considered a core functionality, then test cases were required).

RQ2.- What are the internal and external motivations for engaging in testing efforts on social coding sites?

Contributors reported feel obliged to perform the QA for projects with testing setup or existent tests on the projects gave contributors the impression of informal guidelines. In addition, including test cases in pull requests can increase the value the value of the contribution.

RQ3.- What challenges and risks related to testing arise from engaging on social coding sites?

The constant turnover of contributors makes important to communicate testing culture efficiently. Currently, there are no integrated tools provided by GitHub to lower resting barriers such as setting up a server for CI.

RQ4.- How developers cope with those challenges and risks related to testing?

Project owners used automated CI services to deal with high volume of contributions. Provided testing frameworks with suitable explanation. Provided access to learning resources and supported contributors when having difficulties writing tests.

RQ5.- What impact does the participation on social coding projects have on testing practices?

Projects lower the barrier for contributions by deferring testing to a later stage of the project. Young projects need to get quick contributions to gain traction while accumulating technical debt in the future.

Methodology: Qualitative (Interviews and Questionaires).

Dataset extracted from the GitHub Archive. Interviews to 100 users. 158 out of 1,000 answered questionnaires. Second interview included to 62 users. Second questionnaire was sent to 4,000 users and 569 answered.

[92] R. Pham, L. Singer, and K. Schneider, "Building test suites in social

coding sites by leveraging drive-by commits," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1209–1212. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486964

Managing Pull Requests. This paper is focused on analyzing the impact of drive-by commit phenomenon related quality assurance improvements.

Research Questions:

RQ1.- What are the preconditions for DBC?

The exposure for developers and projects to be searchable and browsable attract and gather large group of developers. This high exposure combined with a project's large periphery and GitHub's low barrier mechanism are generating the new mode of contributing to a project: drive-by commit.

RQ2.- Which circumstances support or prevent DBCs?

Support: User interface should be easy to learn and use, low complexity of the change, clear defined requirements, not need of set up any environment are factor supporting DBC. Prevent: Actions causing a follow-up duties is perceived as extra-work.

RQ3.- How can the DBC mechanism be applied in the domain of testing?

The users proposed to implement an interface to lower the barriers of participation through filtering projects by technology, domain or skills.

Methodology: Mixed (descriptive statistics and interface sketches).

No dataset: Analysis of interview data of our previous work: Creating a shared understanding of testing culture on a social coding site.

[93] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 112–123.

Managing Pull Requests. This paper investigates how common casual contributions are and what are their characteristics. **Research Questions**

RQ1.- How common are casual contributors in OSS projects?
48.98% of contributors are casual contributors. This contributors performed a single contributions and these contributors are responsible for only 1.73% of the total contributions. Similar findings included in "An exploratory study of the pull-based software development model". *RQ2.- What are the characteristics of a casual contribution?*
Casual contributions are no trivial: 28.64% are related to documentation, 30.20% of them fix bugs, 18.75 % included new features. Only 24% of these bugs had a GitHub issue related. Casual contributors are usually active in other projects and they need to implement a feature from specific project. *RQ3.- How do casual contributors and project maintainers perceive casual contributions?*
Project maintainers and casual contributors consider these contributions to be beneficial in the software development process. The motivations behind contributors include giving back to community and gaining reputation and prestige.
Methodology: Mixed (Descriptive statistics and two surveys, first with 197 out of 760 users responding and a second survey with 64 out of 608 answering).
Dataset obtained using GitHub Archive. (October 6, 2015). 275 popular, non-trivial projects.

[94] G. Pinto and F. Kamei, "The census of the brazilian open-source community," in *IFIP International Conference on Open Source Systems*. Springer, 2014, pp. 202–211.

Developers' Roles. This paper focuses on identifying Brazilian open-source communities, characteristics and motivations to collaborate on GitHub.
Research Questions:

RQ1.- Who is the Brazilian open-source contributors?

The Brazilian community is mainly integrated by hobbyst between 20 to 20 years who are students with a oepn-source experience between 2 to 5 years. The majority of them perform around 30 commits per year and only 20.35% of them perform 80.32% of the Brazilian contributions.
RQ2.- Do the Brazilian contributions to open-source increase over the time?
There is an increment of 15% in number of contributions from October 2012 to August 2013. This increase is due to an increase of contributors performing few contributions instead of the increase of contributions by individual.
RQ3.- Why do Brazilian programmers contribute to OSS?
The most common reason is to help the community and improve the software they use.
Methodology: Survey (1,039 responses)

Dataset extracted from GHTorrent (includes 11,411 users) and GitHub API to extract location.
Answers and dataset available at: http://bit.ly/Brazilian_OSS

[95] M. M. Rahman and C. K. Roy, "An insight into the pull requests of github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 364–367.

Managing Pull Requests. This research provides a comparative study between successful and unsuccessful pull requests.
Findings:

The most commonly PR discussions are recursion and refactoring (18.35%), database query execution (11.12%),arrays and functions (11.2%) and OOP paradigm (16.29%).
The projects using Scala, C, C#, R and PHP programming languages have more successful pull requests than failed ones whereas JavaScript, Java, Python, Ruby and C++ have more rejected pull requests. Framework and IDE domains have he higher number of pull request accepted among the application domains.

Developers between year and a half and 4 years of working experience in the project are the most productive. Surprisingly, developers with further experience are found either less productive or making a number of unsuccessful pull requests each month. The authors suggested that those developers might be involved in management activities rather than development.
Method: Quantitative (Latent Dirichlet Allocation (LDA)).

Dataset extracted from GHTorent, 68,916 pull requests in 78 projects with 20,142 developers.

[96] A. Rastogi and N. Nagappan, "Forking and the sustainability of the developer community participation – an empirical investigation on outcomes and reasons," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 102–111.

Forking Popular Projects. This paper analyzes the influence of forking on the sustainability of the community participation in the original project.
Research Questions

RQ1.- How forking influences the sustainability of the developer community participation in the original project hosted on GitHub?
58% of the projects observe no affectation in participation after forking. The 33% and 9% of the projects observed an increase and decrease in participation.
RQ2.- Do projects ported to GitHub behave differently relative to projects created in GitHub?
50% of imported projects observed no affectation in participation after forking. The 30% and 19% a decrease in participation after forking.
RQ3.- How do project characteristics at the time of forking influence the forkability?
-Maturity: In small projects, an increase in the maturity of a project by a year increases the likelihood of being forked by 60%.
-Community Size: In small projects, one unit increase in the contrib-

utors' count increases its forkability by 370%, but only increases to 110% and 40% for medium and large projects respectively.
RQ4.- How do project characteristics at the time of forking influence the sustainability of the developer community in the original project?
-Maturity: The increase in the maturity of the master repository reduces the likelihood of decline in participation by 23% in medium size projects. -Community Size: The increase in the community size of master repository increases the chances of decline participation by 313% in small projects.
Methodology: Quantitative(Mann-Whitney U test, logistic regression and ANOVA analyses).
Dataset obtained from GHTorrent (January 2014). 877,688 projects.

[97] A. Rastogi, "Do biases related to geographical location influence work-related decisions in github?" in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 665–667.

Managing Pull Requests. This paper investigates how the geographical location can influence the evaluation of pull requests in GitHub.
Findings:

Developers from United Kingdom, Canada, Japan, Netherlands and Switzerland have higher chances of getting their pull requests compared to the United States. Countries with the lowest chances to have their pull request accepted are Germany, Brazil, China and Italy. When submitters and integrators belong to the same geographical zone there is a higher probability to have their pull request accepted. Finally, the authors considered that not only technical skills, but also communication skills, behaviour, trust and pro activeness are critical in the pull requests acceptance decisions.
Methodology: Mixed(Logistic Regression and surveys (2,532 responses out of 14,979 prospective respondents)).
Dataset extracted from the research "A dataset for pull request

research" - GHTorrent. 70,740 pull requests.

[98] P. C. Rigby, A. Bacchelli, G. Gousios, and M. Mukadam, "A mixed methods approach to mining code review data: Examples and a study of multi-commit reviews and pull requests," *The Art and Science of Analyzing Software Data. Morgan Kaufmann*, 2015.

Managing Pull Requests. The authors explained how they used quantitative and qualitative methodologies to triangulate their findings on code reviews. They included in their study projects managed by Microsoft with CodeFlow, Google-based with Gerrit code review, OSS code review using mailing lists and GitHub projects using pull requests for code review.

Research Questions:

RQ1.- How many commits are part of each review?
Gerrit project had the largest percentage of multi-commit reviews with 63% while Rails projects on GitHub had the smallest with 29%. In general, single commit reviews dominate on most projects.

RQ2.- How many files and lines of code are changed per review?
Regarding to single commits, the median of line churned varies from 8 to 40. With respect to multi-commit reviews, the median of line churned varies from 43 to 420 lines. Authors suggested that multi-commit changes implement new features or involve complex changes.

RQ3.- How long does it take to perform a review?
GitHub projects perform reviews at least as quickly as Linux and Android projects. Single commit reviews happen in a median between 2.3 hours and 3.3 days. Multi-commits take from 2 to 10 times longer than single commits.

RQ4.- How many reviewers make comments during a review?
The number of comments vary depending the number of commits. Then, the number of comments per review increases on multi-commit reviews. Linux sees as much larger increase from 2 to 6 comments.

In addition, authors analyzed the causes of rejected pull requests, they found that 27% of unmerged pull requests are closed due

to concurrent modifications and 16% are closed as a result of the contributor not having identified the direction of the project and 10% od contributions are rejected due to quality requirements.
Methodology: Mixed (Qualitative: descriptive statistics - Quantitative: manual coding, managers' survey, programmers' survey, semi-structure interview, observing, card sorting, affinity diagram, grounded theory).
Dataset extracted for Linux Kernel, Android, Chrome Rails Katello, Wildfly in a range of 2 years to 4 years.

[99] G. Robles, "Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 171–180.

[100] G. Robles and J. M. Gonzalez-Barahona, "Geographic location of developers at sourceforge," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 144–150.

[101] Y. Saito, K. Fujiwara, H. Igaki, N. Yoshida, and H. Iida, "How do github users feel with pull-based development?" in *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, March 2016, pp. 7–11.

Managing Pull Requests. This paper focused on analyzing responses from a large-scale survey to investigate the use of pull-base development)
Research Questions
RQ1.- How do developers feel with using GitHub and pull request?
Almost none developer felt GitHub and pull request as troublesome. Some developers mentioned the need of unifying Git and GitHub (e.g. including rebasing and resolve conflicts). *RQ2.- How unique practices are used in pull-based development?*
50% of developers use WIP (Work-in-progress) Pull request and perform commit refactoring. A developer beginner who feels Git as difficult tends not to use both WIP pull request and commit refactoring.

Methodology: Qualitative(large survey).

Dataset extracted from GHTorrent. 333 projects with at least 5 pull requests and extracted 13,152 developers. Receiving 1,552 respondents.

[102] J. Saldaña, *The coding manual for qualitative researchers.* Sage, 2015.

[103] R. Salo, T. Poranen, and Z. Zhang, "Requirements management in github with a lean approach."

Excluded paper. This paper present a semi-formal guideline for managing requirements in agile software development projects using GitHub issues. The guideline is focus on complementing the four areas of Requirements Management: change control, version control, requirements tracing and requirements status tracking by utilizing the issue tracker. As GitHub is considered programmer-friendly platform, handling RM in the same place can lower barriers for developers to participate and take a more active role in it.
Methodology: Qualitative.

Dataset: None.

[104] E. A. Santos and A. Hindle, "Judging a commit by its cover: correlating commit message entropy with build status on travis-ci," in *Proceedings of the 13th International Workshop on Mining Software Repositories.* ACM, 2016, pp. 504–507.

Managing Pull Requests. This paper is focused on the analysis of commit messages as proxy for code quality. The authors compared the failed and successful commits included in pull requests with regards to the "unusualness" of their commit message.
RQ1.- How can we measure unusualness of a commit?

Not all commit messages are typed even non-merge commits can be automatically generated.
RQ2.- Is the unusualness of a commit message related to the quality

of the code commited?
There is a slight positive correlation between "unusualness" of a commit message with build failure. However, the results indicated that is infeasible for the average developers to validate a commit by only reading the message.
Methodology: Quantitative(language processing, n-gram language models and cross entropy of commit messages).
Dataset extracted from Boa (September 2015). 120,822 commits from 2,679 projects.

[105] S. Schulze and A. Wasowski, "Forked and integrated variants in an open-source firmware project."

Forking Popular Projects. The authors focused on understanding whether forking is useful for the community and to what extent it provides advantages and disadvantages.
Research Questions

RQ1.- What are the main reasons for creating forks?
75% of forks changed the configuration of the source code. 43% of forks are development forks are used to add new functionality. 32% of forks are bug-fix forks are used to fix defects.
RQ2.- How do ongoing project development and maintenance benefit from existence of many forks? To what extent do forks retrieve changes from their origins?
Only 15% of all forks have synchronized at least once with the main repository. Only 34% of the forks are active and most forks do not retrieve new updates from the main repository.
RQ2.1 To what extent do forks contribute changes to the their origins?
58% of all commits in the main repository are coming from pull requests.
RQ3.- To what extent known drawbacks of cloning in-the-small (e.g., difficulties in propagation changes) apply to forking? Are there any new challenges?

Decentralization of information in many forks is a challenge in fork-intensive development. Propagation of bug-fixes is a problem for forking even though git offers facilities for selective download of patches from upstream.
Methodology: Mixed(Descriptive statistics and survey).

Dataset extracted from GitHub's public API. 1 project (Marlin) and survey applied to 336 users.
Dataset available at https://bitbucket.org/modelsteam/2015-marlin

[106] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on software engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[107] J. Sheoran, K. Blincoe, E. Kalliamvakou, D. Damian, and J. Ell, "Understanding watchers on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 336–339.

Influencing developers This paper analyzes the behaviour of watchers and their contributions to the projects they watch.
Research Questions:

RQ1.- Do watchers become contributors?
4.7% of watchers later became contributors. However, while the rate of watchers who become contributors is small, this amount corresponds to 20.7% of the total population of contributors. The watchers start contributing after a year of watching while start commenting after 9 months.
RQ2.- Do watchers behave differently than other contributors?
Contributors who were not first watchers are most likely to start commenting on an issue. Contributors who were watchers first are more likely to report an issue. In addition, watchers are more likely to have sustained contributions than other contributors. Watchers contribute for 16.4 weeks on average whereas other contributors contribute for only 7.5 weeks on average. A pull request is the first contribution of 43.9% of the watchers whereas only 4.2% of contributors who were not first watchers submit a pull request.

RQ3.- Does a project's programming language impact how its watchers behave?
Projects using C++ receive more comments on commits as a first contribution. Contributors on CSS projects are more likely to submit issues as first contribution. Watchers wait 36.8 weeks on average to contribute on CSS projects while waiting 74.7 weeks to contribute on Ruby projects.
Methodology: Quantitative (Pearson correlation, Man-Whitney tests and Chi-squared test).
Dataset extracted from GHTorrent. 72 repositories, 55,265 users.

[108] Z. Shoroye, W. Yaqub, A. A. Mohammed, Z. Aung, and D. Svetinovic, "Exploring social contagion in open-source communities by mining software repositories," in *International Conference on Neural Information Processing*. Springer, 2015, pp. 120–127.

Influencing developers. This study is focused on the best time to start a project in GitHub and the relationship between followers a developer has and the number of followers that commit to a particular project after the developer.
Research Questions:

RQ1: When is the best time to start a new open-source project?
The best time to start the project on GitHub would be from June to August across most of the years (the commit activity continues increasing during this time).
RQ2: Who should be given the incentive to start new open-source project?
Developers (followers) are likely to participate in projects where their followees have earlier participated. Then, coding can be considered as contagious in the follower network of GitHub.
Methodology: Quantitative(Bootstrap method, goodness-of-fit test, Spearman and Pearson correlation).
Dataset extracted from GHTorrent: 16.7 million repositories and 3.4

million users.

[109] M. C. O. Silva, M. T. Valente, and R. Terra, "Does technical debt lead to the rejection of pull requests?" *arXiv preprint arXiv:1604.01450*, 2016.

Paper excluded. This paper focused on the identification of technical debt in pull requests that can lead to rejections.

Research Questions:

RQ1.- How often pull requests are rejected?
39.4% of pull requests were rejected.
RQ2.- Does Technical debt lead to the rejection of pull requests?
30.3% of the rejected pull requests are due to Technical debt.
RQ3.- What are the most common types of technical debt that lead to the rejection of pull requests?
The technical debt encountered were:
- Design (39.34%): Problems in architecture violations, bad programming practices, violations of object-oriented design principles.
- Test (23.70%): Need for implementation or improvement of submitted tests.
- Project convention (15.64%): Names of variables, method size, and identation.
- Performance (9.48%): Any part of code that can delay or hinder the system's performance.
RQ4.- Which types of technical debt spur more discussion?
The security and project convention debts had the highest number of comments per pull request with 7.3 comments.
Methodology: Quantitative(Descriptive statistics and manual inspection).
Dataset extracted from GitHub API related to 7 projects, 1,722 pull requests.

[110] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual assessment in the social programmer ecosystem: An empirical

investigation of developer profile aggregators," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 103–116.

[111] D. M. Soares, M. L. de Lima Júnior, L. Murta, and A. Plastino, "Acceptance factors of pull requests in open-source projects," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1541–1546.

Managing Pull Requests. This paper includes association rules to find patterns that influence on the acceptance of a pull request.
Association Rules: A) C#, C, TypeScript, Scala and Go increase the chances of occurrence of a merge.
B) The greater the contribution (in terms of commits), the smaller the chances of merge.
C) Pull requests of external collaborators have 13% less chances of being accepted while core members increase in 35% the chances to have their pull request merged. When the pull request is the first made by a developer, the chance of merge is decreased in 32%.
D) Pull request with a commit without adding new files made by core team who had submitted pull request before has 40% more chances to be merged.
Factors affecting pull requests are: Programming language, number of commits, files added, external developer, and first pull request.
Methodology: Quantitative(Association rules).

Dataset extracted from GHTorrent(MSR?14 Mining Challenge): 72 projects and 61,592 pull requests.

[112] D. Spinellis, "Git," *IEEE software*, vol. 29, no. 3, pp. 100–101, 2012.

[113] D. Spinellis and C. Szyperski, "How is open source affecting software development?" *IEEE Software*, vol. 21, no. 1, p. 28, 2004.

[114] Ş. Stănciulescu, D. Rabiser, and C. Seidl, "A technology-neutral role-based collaboration model for software ecosystems," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2016, pp. 512–530.

Tools. This paper introduces a Software Ecosystem (SECO) model to describe collaboration between contributors and artifacts. **Research**

Questions:
RQ1.- Is the expressiveness of the Software Ecosystem (SECO) collaboration role model sufficient to address the challenges arising in software ecosystem environments?
Five collaboration role concepts are sufficient to describe the relations between users, features and repositories.
RQ2.- Is the SECO collaboration role model useful for revealing redundant development efforts?
The models offered a simple view of artifacts and their relations in the SECO which increased awareness of current developments and limiting concurrent development of features or existing ones.
Methodology: Mixed approach (Extraction of meta-data, categorization of collaboration roles based on the steps to create the SECO model).
Dataset extracted from GitHub's public API. 1 project (Marlin) and survey applied to 336 users. Dataset available at https://bitbucket.org/modelsteam/2015-marlin

[115] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP workshop on Future of software engineering research.* ACM, 2010, pp. 359–364.

First paper. This study is the first to mention GitHub as a "social coding environment". GitHub is described as a tool that combines social networking with the Git distributed source-control. A tool that developers use to follow other developers and watch specific projects.

[116] M. Sweet and R. Moynihan, "R: Improving population health: The uses of systematic reviews," 2007.

[117] Y. Takhteyev and A. Hilts, "Investigating the geography of open source software through github."

Excluded paper. This paper conducted an empirical study of the geography of Open source software development . **Research Findings:**

Users and contributions by region:

United States accounts for the largest share of the registered user accounts by 39%. The second largest country is United Kingdom with 7% followed by Germany with 6%. When considering groups, North America and Western-North Europe account for 43% and 26% of the users. Repositories associated with North America account for 59% of repository-watching.

Users and contributions by local cluster:

The top five local clusters are San Francisco, London, New York, Tokyo and Boston that account for 20% of users and 25% of the contributions.

Distance and ties: Contributors located in the same cluster have more contributions on average. 41% of the contributions are in the same cluster.

Methodology: Quantitative(Descriptive statistics, clustering).

Dataset extracted from GitHub API. 70,414 developers.

[118] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in github," in *Software maintenance and reengineering (csmr), 2013 17th european conference on*. IEEE, 2013, pp. 323–326.

Forking Popular Projects. This paper is aimed to understand how developers and projects are related to each other.

Research Questions:

RQ1.- How strong are the relationships among projects?

Project networks are more interconnected than human networks, they only require one common developer to establish a connection between two projects.

RQ2.- How strong are the relationships among the developers?

The majority of developers share projects with few developers. The average shortest path in GitHub is significant lower than the one

presented on Facebook. This suggests that social coding promotes more collaboration among developers.
RQ3.- Which projects are the most influential?
The most influential projects in GitHub are libraries, programmer utilities and scripts and languages implementations.
RQ4.- Which developers are the most influential?
The top-1 developer is part of the core team of Rails and working on 81 projects.
Methodology: Quantitative (Statistics and Page Rank to calculate popularity).
Dataset obtained by GitHub API. 100,000 projects and 30,000 developers.

[119] D. Tranfield, D. Denyer, and P. Smart, "Towards a methodology for developing evidence-informed management knowledge by means of systematic review," *British journal of management*, vol. 14, no. 3, pp. 207–222, 2003.

[120] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th international conference on Software engineering*. ACM, 2014, pp. 356–366.

Managing Pull Requests. This research is focused on the task of evaluating pull requests. The authors analyzed the association of technical and social factors with the probability of contribution acceptance.
Research Findings:

The inclusion of test cases was positively associated with pull request acceptance. The acceptance likelihood increased by 17.1% when tests were included.
Increase of acceptance by 187% when the submitter follows the project manager. Prior interaction was also positively associated with acceptance, increasing acceptance likelihood by 35.6% per unit. The likelihood of acceptance decreases by 54.6% with each unit of comment count.

Being a collaborator on a project increases the likelihood of contributions being accepted by 63.6%.
The older the project, the less likely it is to accept pull requests, with acceptance decreasing by 18% per unit of project age. Popularity has a negative effect on acceptance with projects 35.2% less likely to accept pull requests per unit of increase in stars.
Method: Quantitative(Logistic Regression).

Dataset extracted from GitHub API and GitHub Archive dataset. 12,482 projects, 659,501 pull requests, 95,270 GitHub developers.

[121] ——, "Let's talk about it: evaluating contributions through discussion in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 144–154.

Managing Pull Requests. This papers is focused on how developers evaluate and discuss pull requests by analyzing pull-requests discussions and interviews with GitHub developers.
Research Questions

RQ1.- What are the different kinds of issues raised around code contributions?
Project appropriateness (e.g. implemented features were not in the project scope), value proposition request (e.g core members request submitters to provide use cases or test cases), disapprove of the solution approach (e.g. core members question design decisions), suggest alternative solution (e.g. some core members and third parties implemented an alternative solution to the submitter's implemented solution). *RQ2.- How do participants try to influence the decision process in code contributions?*
Audience involvement (e.g. third parties pressure to core members to influence evaluation decision), use of @-mention feature (e.g. to request feedback from more qualified developers), submitter asks core team about evaluation status (e.g. waiting periods from 18 days to 2 months).
RQ3.- What are the different outcomes for proposed code contribu-

tions?

Regularly highly discussed pull requests were rejected. However, the core team would often still implement the appropriate solution.

RQ4.- Is discussion different when the submitter has prior experience with a project?

When the solution seemed suspect, regardless of the submitter's experience on the project, the developers questioned the approach of the solution.

Method: Qualitatitive (Interviews with 47 GitHub users and qualitative analysis of comments).

Dataset extracted from GitHub API and GitHub Archive.

[122] J. T. Tsay, L. Dabbish, and J. Herbsleb, "Social media and success in open source projects," in *Proceedings of the ACM 2012 conference on computer supported cooperative work companion*. ACM, 2012, pp. 223–226.

Forking Popular Projects. This paper conducted an analysis of the success of projects with respect to social and technical characteristics of developers.

Findings

- *Developer Attention:* Projects with contributors who focus on relatively few projects tend to gather more developer attention, but less work contribution.

- *Work Contribution:* Projects where effort was centralized with a few developers tended to receive much more work contribution. Work concentration is connected with more work contribution.

- *High Task Focus*, which reflects that developers focus primarily on a project, many indicate a high degree of specialization.

- *Low Task Focus* may indicate less specialization , meaning the project is easier to many people to contribute to.

Methodology: Quantitative(Negative binomial regression).

Dataset extracted from GitHub API.

[123] E. v. d. Veen, G. Gousios, and A. Zaidman, "Automatically prioritizing pull requests," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 357–361.

Tools. This paper presents the design of a pull request prioritization tool. This prioritizer works like a priority inbox for PRs, recommending the top pull requests the project owner should focus on. The tool offers an alternative view to GitHub's interface which allows developers to sort open pull requests based on different criteria.
Methodology: Quantitative(Logistic regression, Naive Bayes and Random Forest).
Dataset extracted from GHTorrent.

[124] B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of github teams," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 514–517.

Mining GitHub. The authors present a large data set of social diversity attributes of programmers in GitHub teams which includes 23,493 GitHub Projects. In addition, the researchers included case studies to demonstrate the usefulness of the dataset.
The case studies included the following statistics among the teams:

-Gender: Only 171 (1%) projects are all-female projects and the remaining are all-male projects. 208 projects include a perfect gender balance throughout their history.
- Country: Team country diversity is widespread among 62.7% projects during at least one of their quarters.
- Tenure: The distribution presented in the data is skewed indicating that most projects have low tenure diversity. On average, teams with less experienced developers have less tenure diversity.
- Diversity and Productivity: There is a small (2.5%) positive effect on productivity by gender and tenure diversity for teams larger than 10 developers.
Methodology: Quantitative(descriptive statistics, creation of dataset).

Dataset extracted from GHTorrent with 23,493 projects.
Dataset available at https://github.com/bvasiles/diversity

[125] B. Vasilescu, S. van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. J. van den Brand, "Continuous integration in a social-coding world: Empirical evidence from github," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 401–405.

Managing Pull Requests. This paper explores to what extent GitHub developers use continuous integration using pull requests and commits.
Findings:

- Commits are more popular than pull requests. Similar findings reported by "An exploratory study of the pull-based software development model." The shared repository model is more popular among Java projects while Python and Ruby projects have more contributors submitting pull requests.
- The majority of the projects programmed in Java, Ruby and Python are configured to use Travis-CI with about 92%. However, 45% of projects have no associated configuration and Java projects are not using CI in about 70% of projects.
- Builds corresponding to push commits are much likely to succeed than builds corresponding to pull requests.
Methodology: Quantitative(X test, Stouffer test).
Dataset extracted from GHTorrent and Travis-CI API. 223 GitHub projects.

[126] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov, "The sky is not the limit: multitasking across github projects," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 994–1005.

Developers' Roles. This paper is aimed to measure the rate and breadth of developer's multitasking and how this developer behaviour can affect their productivity.

Research Questions:

RQ1.- What are the trends of, reasons for, and effects of multitasking and focus switching on developer productivity in GitHub?
Developers contribute on average in 2.7 projects per days. During a week, the developers contributed on average in 6 projects. Developers increase multitasking as they participate in more projects per week.
RQ2.- Are there limits on multitasking (and what are they) before productivity is impacted?
47% of the developers have the idea that multitasking in different projects increases the chances of each project to succeed. 40% of contributors experience solving more issues when multitasking in different projects. Higher average daily multitasking is related to an increase in productivity peer week as long as developers do not take more than 4 projects peer week.
Methodology: mixed(regression modeling, ANOVA analyses, Spearman correlation, WMW test and user survey (128 answers out of 851 prospective respondents).
Dataset extracted from GHTorrent. 58,092 projects and 1,255 users.

[127] B. Vasilescu, V. Filkov, and A. Serebrenik, "Perceptions of diversity on git hub: A user survey," in *Cooperative and Human Aspects of Software Engineering (CHASE), 2015 IEEE/ACM 8th International Workshop on.* IEEE, 2015, pp. 50–56.

Collaboration and Transparency. This paper analyzes how teamwork and individual attributes are perceived by developers collaborating on GitHub and how those perceptions influence their works.

Research Questions:

RQ1.- What do people perceive constitutes a team?
Two-thirds of respondents considered themselves part of a team when working on a repository. 53% of respondents indicated that people who contribute code frequently are part of the team.
RQ2.- How does team composition change with time?

Turnover occurs at the periphery, Most of the members are active for a month at most. Some motivations to contribute are: people need to use the code or desire to learn, build skills and move on to other open source projects when they are comfortable, also availability of the GitHub user, changes in employment. In commercial projects (18.7% respondents), changes in team composition are mostly due to HR practices.

RQ3.- Do individuals recognize differences among others on their team? Which differences are more prominent?

Almost three quarters of the respondents are aware of the programming skills of the other team members. Almost half of the respondents were aware of the gender and real name of their teammates.

RQ4.- What mechanisms contribute to increased awareness of diversity attributes among team members?

1)In-person interactions: 52% of respondents know their team members in persons. 2)GitHub-enabled interaction: Team members learn about their programming skills from each other's code contributions, issues and pull requests, they use comments on commits and pull requests to make inferences. *RQ5.- How is diversity perceived to influence collaboration?*

Positive Effects (62%): Diverse viewpoints often lead to lively discussions and new ideas and perspectives to solve problems. Negative Effects (30%): Improper contributions by newcomers. Differences in ideology between team members, it can be time consuming with little return value.

Methodology: Qualitative(Surveys 816 responses out of 4,500 users).

Dataset extracted from GHTorrent.

[128] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 3789–3798.

Developers' Roles. This paper analyzes how gender and tenure

diversity relate to team productivity and turnover.
-Productivity is measured by the number of commits recorded in either the main repository or any of its forks in a given quarter.
-Turnover is measured as the number of contributors in a given quarter with respect to previous quarter.
Findings:

Gender diversity has a positive effect on productivity: The effect of gender diversity is positive , highly significant, and stable over all team size ranges. As gender diversity increases, team productivity increases.
Tenure diversity has a positive effect on turnover: Tenure diversity is significant and contribute significantly and positively to the turnover. Tenure diversity effects are positive: more diverse team experience more staffing changes.
Methodology: Mixed(survey (816 responses out of 4,500). Multiple linear regression models, ANOVA analyses).
Dataset extracted from GHTorrent, 873,392 GitHub contributor.

[129] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 805–816.

Managing Pull Requests. This paper is aimed to understand the effects in projects using CI.
Research Questions:

RQ1.- How is the productivity of teams affected by CI?
The larger the team, the more likely core developers would submit PRs and they would be accepted. Older projects accept fewer and reject more PRs from this external contribution, the older the project, the fewer external PRs the project receives. In popular projects, the core developers are less likely to submit PRs. In addition, CI is also associated with external contributors having fewer PRs rejected.
RQ2.- What is the effect of CI on software quality?

The older the project, the fewer bug reports it receives from core developers. The more popular the project, the fewer bugs are reported by core developers. The more forks the projects has, the more external contributors are reporting bugs. Core developers in projects using CI are more likely to find bugs than teams not using CI.
Method: Quantitative(negative Binomial Regression).

Dataset extracted from GHTorrent dump dated 10/11/2014. GitHub API 246 projects, 182,056 pull requests.

[130] G. Von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Research Policy*, vol. 32, no. 7, pp. 1217–1241, 2003.

[131] P. Wagstrom, C. Jergensen, and A. Sarma, "Roles in a networked software development ecosystem: A case study in github," 2012.

Excluded paper. This paper identifies the different roles among projects in GitHub and the transition between one role to other.
Research Questions:

RQ1: How has the basic user/developer dichotomy in an open source project evolved with addition of social data and more robust tracking of code contributions?
-Lurkers , issuers, independent, aspiring, external contributor, internal collaborators. *RQ2: What are the more nuanced roles of developers in a modern networked open source project?*
The nomad coders make small contributions across a set of projects. About 43% of rockstars also execute other roles. It is impossible for an individual to be a Nomad as well as rockstar or code warrior. Surprisingly, it is more common to be rockstar/prodder or rockstar/steward than a rockstar/code warrior. There are some nomads who also serve as stewards on a project. Those individuals close many issues and handle many pull requests without writing new code for the project.
RQ3: With the addition of social data and tracking of relationships

across an entire ecosystem, how has the understanding of participation across open source projects evolved?
There is little overlap of external contributor role across projects. Although some projects are related, most projects have their own core groups. There were no users that were aspiring developers on multiple projects which might indicate that developers did not attempt to contribute to multiple projects simultaneously.
Methodology: Quantitative(Descriptive Statistics).

Dataset extracted from GitHub API.
Source code available at https://github.com/pridkett/gitminer

[132] W. Wang, G. Poo-Caamaño, E. Wilde, and D. M. German, "What is the gist?: understanding the use of public gists on github," in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015, pp. 314–323.

Managing Pull Requests. This paper explores what Gists are and how they are used in GitHub.
Research Questions:

RQ1.- What do gists look like?
The gists are used by small proportion of users (14%). 86.8% of Gists contain a single file and 98.5% contain at most 4 files. The most common files included in Gists are images, HTML files, JSON data and CSS. The median size of Gists is 920 bytes, 22 lines, 18 SLOC. They provide collaboration through forking (5.1%)them. 63% had a single change and most of the Gists (92.8%) had 2 to 5 commits.
RQ2.- How are users using gists?
The regular use of Gits include code sharing, syntax highlighting, embedding Gits in Web Pages and to-do list. "Draft" a Pull-request, issue or comment on GitHub by using Gists.Gists are rarely forked and the majority of Gists never change.
Methodology: Mixed (Qualitative analysis of a small sample of Gists and quantitative analysis of Gists from large portion of GitHub users).

Dataset extracted from GHTorrent, 750,000 users and 762,000 Gists Dataset not available at http://turingmachine.org/2015/gists (broken link).

[133] Z. Wang and D. E. Perry, "Role distribution and transformation in open source software project teams," in *2015 Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2015, pp. 119–126.

Developers' Role. This paper identifies the patterns of team structures and role transformations in order to understand the dynamic role distributions and role transformations among the teams.

Research Questions:

RQ1.- Are there significant differences between the team constituents of different FLOSS projects? Can FLOSS projects be classified in term of the role distribution?

In the largest projects, the ratio of watchers is the largest too, but the ratios of other roles are significantly lower than the others project size. Popular projects attract mass of developers as watchers, but it is difficult for these watchers to make deeper contributions to the projects. In the smallest project, the ratio of watchers is the lowest , but the ratio of the other roles are significantly higher than the other projects. There are 30% forkers who are willing to observe more details of the project instead of watching. The ratios of committers and members are higher.

RQ2.- What kinds of role transformations are more frequent than others and what are less frequent? Do different FLOSS project teams exhibit distinct role transformation patterns?

In large projects, the changes from watcher to forker, issue reporter to commenter are higher than the others projects. In small projects is more difficult to attract watchers, but if some of them are involved, they are more willing to communicating with other members and contribute to the code. The smallest projects require less Time of Activeness to go into inner roles.

RQ3.- If a developer simultaneously undertakes multiple roles in one

FLOSS project, are there any underlying correlations among the contributions he made in different roles?
The changes in the roles rely on the developer's working habits and they are not enough significant to find any correlated pairs in the roles.
Methodology: Quantitative(feature vector, k-means algorithm and Chi-Square test).
Dataset extracted from GHTorrent (January 2012 - June 2013). 89 projects and 192,910 developers

[134] S. Weber and J. Luo, "What makes an open source code popular on git hub?" in *2014 IEEE International Conference on Data Mining Workshop*, Dec 2014, pp. 851–855.

Forking Popular Projects. This paper investigates the relationship between code popularity and other features such as programming language, documentation and code volume.
Findings:

The most important features for popular projects are: readme size, with statement usage and Travis CI configuration size. Popular projects were found to have larger READMEs. In addition, 95% of popular projects have non-empty READMESs. 22% of popular project have TravisCI configuration files. In conclusion, popular projects have more documentation.
Methodology: Quantitative (Decision trees (rank)).

Dataset extracted from GitHub API. 13,000 projects.

[135] G. Winter, "A comparative discussion of the notion of'validity'in qualitative and quantitative research," *The qualitative report*, vol. 4, no. 3, pp. 1–14, 2000.

[136] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. ACM, 2014, p. 38.

[137] Y. Wu, J. Kropczynski, P. C. Shih, and J. M. Carroll, "Exploring the ecosystem of software developers on github and other platforms," in *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing.* ACM, 2014, pp. 265–268.

Influencing developers. This paper analyzes how developers follow others and to what extend the follow feature on GitHub is similar or different to social media functions.

Research Question:

RQ1.- How do users find the developers that they follow, if not through direct co-participation in productive activities?

Developers make social connections with others on platforms such as Twitter and Hacker. Developers used Twitter to learn about ideas, Hacker News to be aware of trending projects, Quora and StackOverflow to ask and answer technical questions. All developers mentioned that their interactions on these platforms result in followship or collaboration.

Method: Mixed (Quadratic Assignment Procedure (QAP) correlation, Pearson correlation and survey).

Dataset extracted from GitHub archive. Homebrew project with 6,554 developers. Survey to 5 developers.

[138] K. Yamashita, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi, "Magnet or sticky? measuring project characteristics from the perspective of developer attraction and retention," *Journal of Information Processing*, vol. 24, no. 2, pp. 339–348, 2016.

Forking Popular Projects. This paper is focused on understanding how projects retain and attract contributors by using population migration metric called Magnet (personnel attraction) and Sticky (retention) metrics. The magnet metric indicates the number of new developers attracted to a project. The sticky metric indicates the number of existing developers that stay with the project.

Research Questions

RQ1: What are the typical distributions of projects from magnet and sticky perspectives?
23% of contributors remain with the same project irrespective of size (total of developers) and new developers tend to join popular and famous projects. Larger projects attract and retain larger number of new contributors than smaller projects.
RQ2: How do the magnet and sticky values change over time?
53% of terminal projects eventually decay into a state of have fewer than ten contributors. On the other hand, 55% of attractive projects keep their popularity. Furthermore, stagnant projects (projects struggling to retain existing developers while failing to attract new ones) are more likely to decay than fluctuating projects (projects that successfully attract new developers but tend to lose existing ones).
Methodology: Quantitative(descriptive statistics based on sticky and magnet metrics).
Dataset extracted from GHTorrent 16,652 projects.
Tool used is available at https://github.com/bvasiles/ght_unmasking_aliases

[139] Y. Yoshikawa, T. Iwata, and H. Sawada, "Collaboration on social media: Analyzing successful projects on social coding," *arXiv preprint arXiv:1408.6012*, 2014.

Excluded paper. This paper investigates the keys to success of projects based on the the number of commits, number of stars and number of pull requests.
Research Findings:

- Project with larger number of internal members have higher activity (commits), popularity (stars) and sociality (pull requests). However the work effectiveness of each developer decreases significantly when the number of internal members exceeds 60 developers.
- Projects that consistently tackle pull requests from external developers are more likely to exhibit higher popularity (stars) and sociality

(pull requests)
- Successful projects are more likely to exhibit strong connectivity between the internal members.
- Increasing transparency by core developers submitting pull requests would lead to higher popularity (stars) and sociality (pull requests).
Methodology: Quantitative(Kendall rank correlation coefficient, Latent Dirichlet Allocation(LDA), linear regression).
Dataset extracted from GitHub API and GitHub Archive: 317,077 projects, 41,720,139 commits, 5,164,934 stars, 1,903,595 pull requests, 1,381,121 developers.

[140] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 367–371.

Managing Pull Requests. This paper analyses which factors affect pull request evaluation latency in projects using Continuous Integration in GitHub.
Findings:

Pull requests with more discussion, commits and more lines of code are associated with longer evaluation latencies. The effect of submitters track record, reputation and social connection to project members are highly significant. Pull requests made by the core team members, rockstars, developers close to project integrators are associated with shorter evaluations. Pull Requests touching recent modified parts of the system are regularly accepted.
The priorities assigned by integrators affect the evaluation time. The integrators' workload and availability (e.g. pull requests submitted outside business hours and Fridays tend to be closed later).
The use of CI allows to find integration errors faster. Integrators wait for automatic testing phase to end (median 39 minutes) before proceeding to do a team code review and close the pull request.
Methodology: Quantitative(multiple linear regression models and ANOVA analyses).

Dataset obtained by GHTorrent using the 10/11/2014 dump file and GitHub API. 40 projects and 103,284 pull requests.

[141] Y. Yu, G. Yin, H. Wang, and T. Wang, "Exploring the patterns of social behavior in github," in *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*. ACM, 2014, pp. 31–36.

Influencing Developers. This paper explores the growth of project and users in GitHub as well as analyzing behaviour among developers.

Research Questions:

RQ1.- What are the differences between the growth mode of GitHub and traditional OSS communities, and is there any sociological theory that supports the special growth of GitHub?

Two main reasons why majorities are involved in GitHub:

- *Effect of Leader*: Popular developers with high reputation using GitHub and some outstanding projects such as Rails that involved a lot of developers and attract to many attention.

- *Herd Behaviour:* Large number of users joined GitHub because they found other developers talking about GitHub.

RQ2.- Whether or not the social connections among developers form some distinctive behaviour patterns in GitHub and if it is true, what are these patterns?

The more developers are followed by external users, the faster their project growing. The social connections follow a star-pattern and hub-pattern. In the star-pattern, a popular developer is followed by a large number of users, but he almost never pay any attention to others (effect of leader). In the hub-pattern, the core developers have connections only with internal users. Another pattern, a developer not only follow internal developers, but also make connections with other communities.

Methodology: Quantitative (follow-networks and Gini coefficient).

Dataset extracted from GHTorrent. 1,838,805 users.

[142] Y. Yu, G. Yin, T. Wang, C. Yang, and H. Wang, "Determinants of pull-based development in the context of continuous integration," *Science China Information Sciences*, vol. 59, no. 8, p. 080104, 2016.

Managing Pull Requests. This paper investigates the factors that affect acceptance and latency in pull requests in the context of Continuous Integration (CI).

Research Questions:

RQ1.- How do social, technical and process-related factors influence pull-request acceptance and latency?

The project age and hotness area are significant factors to predict acceptance. Submitting test cases with the pull-requests would moderately increase the acceptance.

The more controversy in a pull-request, the more likely it is to be rejected.

Latencies factors:

- Pull-request churn, commit size, complexity, and length of discussion account for 63.8% in increase on the evaluation latency. - Pull requests including commits with small pieces will be processed quicker.

RQ2.- What is the effect of CI on pull-request acceptance and latency?

When the pull-request have failed CI testing, the rejection would increase by 89.6%. Integrators are willing to wait for the CI outcomes to assist them in making final decisions.

RQ3.- What is the difference of traditional predictors affected by CI-related variables involving?

The effect of commit size would indicate an agile principle to separate the code into small pieces and commit more frequently.

Methodology: Quantitative(Multiple Linear Regression, ANOVA analyses, Variance Inflation Factors (VIFs)).

Dataset extracted from GHTorrent and GitHub API. The dataset contains 103,284 pull-requests collected from 40 different projects in GitHub.

[143] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "A exploratory study of @-mention in github's pull-requests," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2014, pp. 343–350.

Managing Pull Requests. This paper analyzes the use of @-mention in pull requests in order to understand the impact of @-mention and its benefits.

Research Questions

RQ1.- To what extent is @-mention used in the pull request paradigm?
In total, only 12% of pull requests have @-mention. 79% @-mentioned is used by internal developers to locate the suitable reviewers that external developers can not recognize.

RQ2.- What kind of differences are there between the pull-requests with and without @-mention?
1)PRs with @-mention are slightly larger with 5.4 commits in average while without @-mention is decreased to 4.1 commits.
2)PRs with @-mention are more likely to have more comments (7.2 comments in average) while without @-mention the comments are reduced to 1.
3)PRs with @-mention are more likely to need more time to deal with (350.8 hours) while without @-mention is 101.5 hours.

RQ3: How well do @-mention in the pull-requests support the developers for communication?
1) @-mention helps developers to find and respond pull requests (37.2 hours) in comparison with PRs without @-mention with 78 hours for the first comment in the PR.
2) Considering pull requests reviewed by the same number of reviewers, the pull-requests with @-mention need shorter handling time.

RQ4.- Does the specific location of @-mention impact the processing of pull-request?
1) The processing time for PRs with @-mention in their bodies is 77.6 hours to handle it.
2) PRs with @-mention in their review comments increased to 171.4% to handle it.

3) PRs with @-mention in their issue comments continue increasing to 450.4 hours to handle it.
Pull-requests with @-mention used in their bodies accelerate the evaluation of the PRs.
Methodology: Quantitative (Statistics tests used included MMW test, Z test and Cliff's delta).
Dataset extracted from GHTorrent and GitHub Achive. 3,587 projects and 744,684 pull requests.

[144] Y. Zhang, H. Wang, G. Yin, T. Wang, and Y. Yu, "Exploring the use of@-mention to assist software development in github."

Managing Pull Requests. This paper explores the use of @-mention in GitHub.
Research Questions:

RQ1.- To what extent is @-mention used in the issues of GitHub?
Most @-mention are used in the pull requests with nearly half of the @-mention. The frequency of each issue using @-mention is low. Most @-mention are used in issue's comments rather than issue's description body. The @-mention is more likely to be used in the conversation between developers.
RQ2: What kind of differences are there between the issues with and without @-mention? Does using @-mention influence the solving process of issues?
Issues with @-mention are more likely to have more comment than issues without @-mention . The @-mention is used to promote discussion. Issues with @-mention are more likely to need more time to deal with than issues without @-mention. However, @-mention tends to be used in difficult issues. Issues with the same difficulty using @-mention are more likely to be solved faster than without @-mention.
Methodology: Quantitative(Mann-Whiney-Wilcoxon test, Z test and Clif's delta).
Dataset extracted from GitHub Rest API. Rails Repository, 10,094

developers and 21,273 issues.

[145] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "Investigating social media in github's pull-requests: a case study on ruby on rails," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*. ACM, 2014, pp. 37–41.

Managing Pull Requests. The authors conducted a quantitative analysis of @-mention in pull-requests of the project Ruby on Rails.
Research Questions

RQ1.- What is the popularity of pull-requests in GitHub?
60% of commits were coming from pull requests.
RQ2.- What is the current situation of @-mention used in the pull-requests paradigm?
@-mention is not widely used in the pull requests and @-mention is more likely to be used in comments than in pull-request's body. The pull-requests with @-mention have 3.9 commits, 6.7 comments and 3.3 participants. On the other hand, pull requests without @-mention include 5.5 commits, 1 comment and 1.5 participants.
Methodology: Quantitative (descriptive statistics).

Dataset extracted from GitHub Archive and GHTorrent. 9,129 pull requests, 43,526 commits, 21,772 stars and 7,980 forks .

[146] J. Zhu, M. Zhou, and A. Mockus, "Patterns of folder use and project popularity: a case study of github repositories," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, p. 30.

Forking Popular Projects. This paper is focused on how folders are used and what different uses may have.
Research Questions:

RQ1.- What are the most commonly used folders in the projects?
Folders are related to programming languages and application domains (i.e. com is popular among Java programs and css is popular

for web applications). Almost half of the projects use folder src, and one third use lib and test. Standard folder structure (i.e. test, doc and examples).

RQ2.- Are they associated with the chance of the project code being forked?

The number of forks depends on the presence of testing, documents and example folders.

Methodology: Quantitative (Frequencies and regression models).

Dataset extracted from GitHub API. 146,124 projects.