SECRET LIFE OF SOFTWARE CODE

Celal Karakoç
Department of Computer Science
Technical University of Delft
Student number: 4342933
c.karakoc@student.tudelft.nl

ABSTRACT

Pull-based development is a widely adopted distributed development model for contributing to open source software (OSS) projects. In the pull-based development model, a variety of factors have an influence on the decision making of the pull-requests. This literature review looks at and summarises all the factors, of around 80 papers, that have a correlation with the acceptance rate. We have set up a table containing a summary of all factors, how they were obtained within their respective papers and we note down the outcomes per study. The results mainly provide insight on the correlation of the factor with the acceptance rate, summarised from one or more papers, into a single paragraph. The conclusion that we reached on the outcome on a per-factor basis can be seen in appendix A.

1 Introduction

Pull-based development is a distributed development model for developing large-and-small-scale software projects. The development model works with the usage of forking one single central repository and after forking the repository, the contributors then contribute code, be it a new feature, bug fix or a small patch, to their forked repository and create pull requests to merge their work in the main repository[1]. A pull request is a platform to discuss the work and request changes if necessary or even deny the contribution all together. This model allows for a distributed workflow, which in turn makes sure there can be a multitude of contributors working on the feature at the same time. There are some websites that allow you to work in this model, such as Github, Gitlab and Bitbucket. Furthermore, several third-party applications (applications not part of the primary service), e.g. hooks, can be added on the aforementioned services, to which more data and thus more stability and reliability can be added towards the process. These hooks are mainly continuous integration services, whom are automated systems that check whether code-guidelines are followed, tests all passed and sufficient tests are written. The integrator can use these hooks easily and as a time-savior to immediately check for sufficient testing and code-style errors, or as a quick barrier for obviously bad code.

There exist other models for software development. One adopted by some popular OSS projects over the years is the mailing list[2]. This model has its development through a series of email threads, where people can discuss implementation and possible new features. Another model is the shared repository. This model has the core developers sharing the read and write permission of the repository with contributors, making it so contributors can clone the repository to their local machine and push changes and even create branches in the central repository.[1] Both the pull-based development model and the shared repository model are examples of Distributed Version Control Systems, which allows distributed development among the branches.

PDF-XChange Editor Click to BUY NOW! www.pdf-xchange.com

The pull-based development model and in extension all the other DVCS have not been around for too long. The development process had to evolve from one point. In earlier development strategies, centralized version control systems were in place. In 2009, the most commonly used version control system was this centralized version control. This caused the project to have only one single canonical source repository where developers would work against through a checkout taken from the repository [3]. This however had the disadvantage that the code was not owned by contributors, but rather small patches had to be submitted through only one repository. With the upcoming of DCVS, every contribution became its own project in the forked repository, causing the contributors to not need to write access to the main repository, allowing better development flow in the process. Even before 2009 there have been multiple other iterations of version control systems in place. In fact the first system that used version control to work collaboratively was made by Dick Grune in 1985. This system had a Client/Server model, which allowed the version history of the project to be stored in a central server whilst the developers worked on a copy of the files on their own machines [4].

Pull-based development has the distinct advantage that the code has the protection of the core developers, where the contributors only change their own forked repository and make pull requests from that forked repository. This protects the main repository from bad code, since every change has to be reviewed by a core developer first. Pull-based development also enables contributors and integrators to easily peer-review the code when a pull request is made. This *code review*, makes it such that questions can be asked and feedback can be given before any merge to the main repository occurs. Said question and feedback generally occur on the comments of the pull request and help solve misunderstandings.

Furthermore, unlike the mailing lists, the pull-based development model doesn't require one to send their patches through mailing lists. With this, the model also appears to generate more awareness for the issues at hand. The pull-based model appears to be more robust and a general step forward.

Lastly, pull-based development can also attract more contributions from casual contributors[1]. Since the effort required to fork the repository is very low, it forms a low barrier for people to start contributing to a repository.

Within this literature review, we will mainly look at the factors that contributed towards the pull-based development model. For each factor, we will report how the factor is measured, the outcome regarding the decision making and how the study is conducted. The full results can be seen in appendix A. With this paper, we will try to answer the following research questions:

[RQ_{main}] What factors influence pull request decision making?

This question will be split up into four sub-questions, mainly:

[RQ_{sub1}] What factors related to the developer influence pull request decision making?

[RQ_{sub2}] What factors related to the project influence pull request decision making?

[RQ_{sub3}] What factors related to the process influence pull request decision making?

[RQ_{sub4}] What factors related to the code influence pull request decision making?

2 Aims

This literature review assesses the influence of factors on the acceptance or rejection of the code contribution, within the pull-based development model. We broadly categorize all factors within 4 main groups: developer, project, process and code. Our aim is, by looking at the influence on the pull-based model regarding factors of the developer (i.e. gender, country, track-record), the factors within a project (i.e. popularity of the project, programming language), the factors surrounding the process (i.e. comments, length of discussion, age of a pull-request) and the factors regarding code (i.e. code style, comments, amount of blank lines), we can derive an answer for our research question or a set of indications of the future of pull-based development and the direction it is moving to.

3 Methodology

3.1 Search Terms

As a starting point, we used as initial search-terms: pull-based development, pull-request, GitHub and open source. These were our main terms, and we combined them with zero or more of the following sub-terms: model, software, accepted, rejected, review, merged, through google scholar's boolean search queries. The more papers we read, the more specific the search terms became based on the titles and content of the papers. With snowballing through the papers we searched the specific titles of the papers.

PDF-XChange Editor Click to BUY NOW! www.pdf-xchange.com

3.2 Time frame

The division of the search was also based on a division of a time-frame. We only selected articles within the last 10 years, namely 2009-2019.

3.3 Sites

The base search engine we used was scholar.google.com. With that, we found and got our resources and papers from mainly these sites: researchgate.net, ieeexplore.ieee.org, arxiv.org, dl.acm.org, scholar.google.com.

4 Factor Selection

The factors selection was based on the separation of the pre-and-post pull-based development cycle. With this, we mean the area of factors before a pull request, during a pull request and after a pull request. The factors are primarily looked at in the outcome of acceptance rate or rejection rate. Furthermore, the division is split into the following four categories: developer, project, code, process.

4.1 Developer characteristics

Developer characteristics look at the different influences developers have on the pull-based development model. We found the distinction between integrators and contributors and separate, wherever necessary, the factors based on these two categories. We use the term developer as a umbrella term, including integrators and contributors. We also group together similar characteristics from several papers (i.e. country of the developer and geographical location of the developer), wherever it makes sense or the meaning is retained. Firstly, we quantify the innate factors about the developers and their environments, like gender type of developer and geographical location. Furthermore, we quantify the factors surrounding the skill of the developer, like track-record, experience and expertise. Lastly, we look at social factors regarding the developer. Social factors include the relation between contributors (propensity to trust), prior interaction, affiliation, activeness and level of participation. All factors regarding developers can be seen in appendix A table 1.

4.2 Project characteristics

Project based characteristics are all the project related factors. Note that the popularity of the project is measured differently in every paper, some papers measure the rating of the project, some measured the followers of a project. We mapped all these factors in the table as popularity of the project. Firstly quantify the factors that are software related in the project (i.e. programming language), then we quantify at non-software related factors (i.e. age and popularity). All factors regarding project can be seen in appendix A table 2.

4.3 Process characteristics

Process based characteristics look at the factors that are related to the process a pull request or patch goes through during pull-based development. The process looks at the pull request characteristics itself (like the number of comments on a pull request and the age), while later looking at the substance of the development process (i.e. use of automated systems and conformation to the project standards). All factors regarding process can be seen in appendix A table 3.

4.4 Code characteristics

Code based characteristics look at the factors that are related to the source code of the pull-requests. It looks at what is actually inside of the pull-request, at what the developers want to merge. The code related factors include among other things code style, size of commit and comments. Some factors were extracted from multiple papers, while others only have one paper related to them. We have mapped all the factors in a table, this can be seen in appendix A table 4.

5 Developer

In total, 34 papers were used to indicate the factors related to the developer. In figure 1 a distribution can be seen of how the studies were conducted in these papers. Do note, however, that some papers used several means of conducting the studies. In regards to the developers, datamining of large-scale projects (either through GHTorrent (20.7%) [5] or other means (53.4%)) is how most of the data is collected (74.1%). After that, singular or small-scale projects make for a sizeable amount of the collected data (18.5%). Small-scale projects whose patches were datamined and looked through include the Linux Kernel [6] and Mozilla project [7]. Finally, two large-scale surveys (7.4%) [8, 9], with 750 and 2500+ developers respectively, were used to attribute outcomes for the factors.

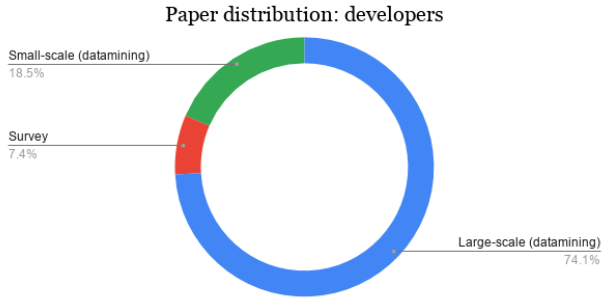


Figure 1: How the studies were conducted in the papers regarding developers.

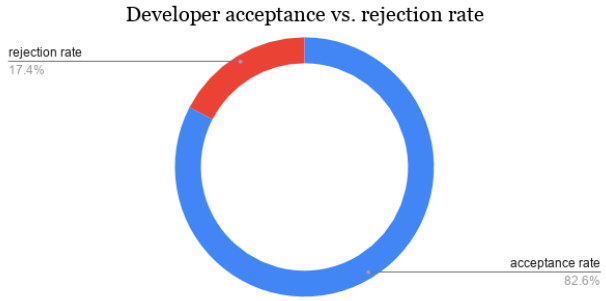


Figure 2: How the outcomes were reached in the studies of the papers regarding the developers.

The outcomes of our literature review, as seen in A.1, indicate that most of the factors have a positive correlation with acceptance rate, see figure 2. A total of 82.6% of the papers shows that the acceptance rate increases in correlation to the factor, while 17.4% of the papers show an increase in the rejection rate.

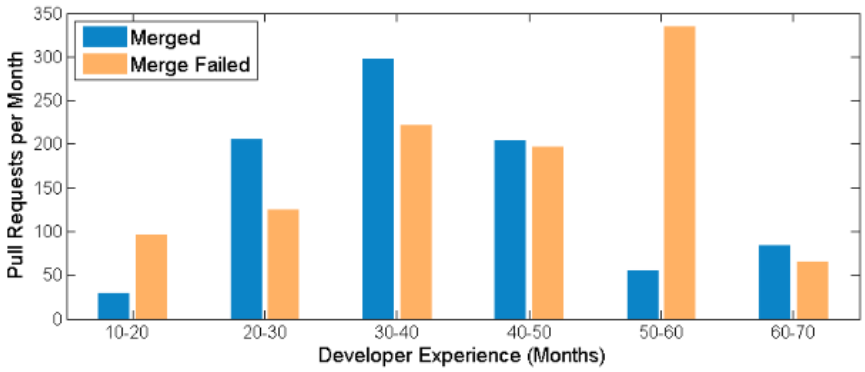


Figure 3: Pull requests vs. experience [10]

In total, we have 11 different factors for the developers, which are a categorisation of similar specific factors. This means that factors such as, country of the developer and geographical location of the developer have been put together into an overarching factor.

Gender of the developer, separated between male and female, has been acquired through parsing the about data of users in Google+. This data indicates that a problem of trust exist and that the genders of the users are truly correct cannot be a 100% trustworthy or even verifiable. With that in mind, it is shown that female developers have a higher chance of getting accepted if they do not identify their gender. Furthermore, pull requests made by men were more likely to be accepted than those made by women [11].

Experience of the developer is a widely used factor regarding the acceptance rate. With experience many papers look at different aspects, but the main ones that keep reappearing are the number of submitted changes [12], the number of commits (per time span) [13], the amount of interaction with the project [14] or a mix of these factors. It is noted that developer experience plays a major role during code review and leads to a greater acceptance rate and a faster response time [12]. Also, the contribution is more likely to not be re-opened and accepted with an experienced developer

[13, 15, 14]. A general pattern regarding experience can be seen in figure 3. Within this figure, contributors with 20 to 50 months of experience are found the most productive and have the highest acceptance rates [10]. This is measures as merge successes as opposed to merge failures. The more inexperienced developers (10 to 20 months) also seem to have the most merge failures. Generally speaking, the pull request seems to have a greater acceptance rate with an experienced contributor. [13]

Affiliation of contributor looks at the contributor and integrator affiliation. An integrator is more likely to accept the contribution if the contributor is affiliated, either with the organization, project or the integrator. [12, 16, 17]. If the contributor follows a core contributor, the acceptance rate is increase [16]. Being a collaborator on the project increases the likelihood of acceptance by 63.6%, according to Tsay [17]. As can be seen from this, having an affiliation with the organization, project or the developer indicates a positive correlation with acceptance rate.

Track record of the developer is an indicator that, surprisingly, was not ranked that high as a priority in the survey [8]. Despite that, if a contributor has submitted more pull requests previously, than a contributor's pull request is more likely to be accepted [18, 19]. Thus, a known contributor, has a higher chance of acceptance [8].

	Model 1	Model 2
(Intercept)	2.82 (0.14)***	2.61 (0.14)***
Control variables		
proj_months_existence	−0.01 (0.00)***	−0.01 (0.00)***
proj_watchers	−0.00 (0.00)***	−0.00 (0.00)***
log(proj_ncloc + 1)	−0.06 (0.01)***	−0.06 (0.01)***
proj_external_contribs	−0.01 (0.00)***	−0.01 (0.00)***
proj_test_loc_per_lloc	0.00 (0.00)***	0.00 (0.00)***
log(dev_followers + 1)	0.06 (0.01)***	0.07 (0.01)***
dev_commit_access	0.06 (0.07)	0.05 (0.07)
dev_followed_integrator1	0.11 (0.03)**	0.10 (0.03)**
dev_watched_project1	0.04 (0.03)	0.05 (0.03)
log(dev_prev_pull_requests + 1)	0.17 (0.01)***	0.17 (0.01)***
dev_success_rate	0.01 (0.00)***	0.01 (0.00)***
dev_months_participation	−0.00 (0.00)***	−0.00 (0.00)***
log(pr_comments + 1)	−0.25 (0.01)***	−0.24 (0.01)***
log(pr_changed_loc + 1)	−0.06 (0.01)***	−0.06 (0.01)***
log(pr_changed_files + 1)	0.01 (0.02)	0.01 (0.02)
pr_test_inclusion1	0.26 (0.03)***	0.26 (0.03)***
geo_country_switzerland	0.38 (0.11)***	0.46 (0.11)***
geo_country_netherlands	0.26 (0.09)**	0.36 (0.09)***
geo_country_japan	0.25 (0.08)***	0.34 (0.08)***
geo_country_united kingdom	0.13 (0.04)**	0.20 (0.04)***
geo_country_canada	0.12 (0.07)	0.22 (0.07)**
geo_country_belgium	0.09 (0.12)	0.18 (0.12)
geo_country_spain	0.08 (0.10)	0.15 (0.10)
geo_country_australia	0.05 (0.07)	0.14 (0.07)
geo_country_india	0.02 (0.07)	0.12 (0.07)
geo_country_france	0.02 (0.06)	0.11 (0.06)
geo_country_russia	−0.06 (0.07)	0.04 (0.07)
geo_country_sweden	−0.21 (0.09)*	−0.10 (0.09)
geo_country_germany	−0.25 (0.04)***	−0.16 (0.05)***
geo_country_brazil	−0.27 (0.06)***	−0.19 (0.07)**
geo_country_italy	−0.31 (0.08)***	−0.21 (0.09)*
geo_country_china	−0.39 (0.09)***	−0.27 (0.10)**
geo_same_country1		0.18 (0.03)***
AIC	49231.59	49198.20
BIC	49534.09	49509.87
Log Likelihood	−24582.80	−24565.10
Deviance	49165.59	49130.20
Num. obs.	70740	70740

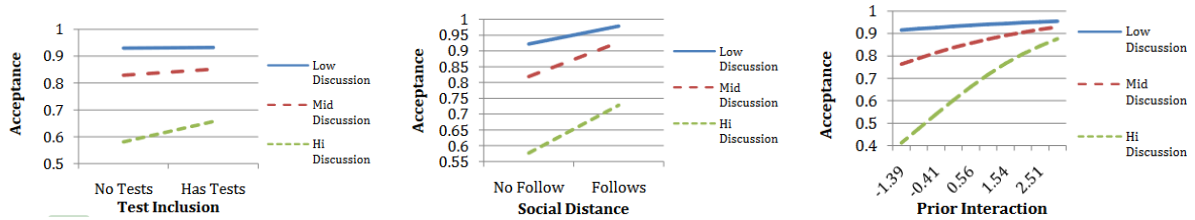
***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

Figure 4: Logistic regression models of factors influencing pull request acceptance [20]

Geographical location indicates the country of the developer or regions close to each other. It holds true that when contributors and integrators are either from the same geographical location [9] or from the same country [20] the chance of pull requests getting accepted is higher than when they are from different geographical locations or countries. It is worth to note, however, that the papers for this factor only looked at 17 locations based on GitHub as a primary source, see figure 4. Being in the same country increases the chances of pull request acceptance by a factor of 1.2 in comparison to the submitter and integrator being from different countries [20].

Level of participation is looked at through the number of patches of the contributors. This patch life cycle indicates that casual integrators (< 20 patches) have a lower acceptance rate and a greater rejection rate as compared to core integrators (>100 patches) [21]. Thus, core developers enjoy a higher acceptance rate than the peripheral developers [22].

Expertise of the integrator means the field or area in which the developer has a lot of skill in (i.e. specializes in). The pull requests made by a contributor is more likely accepted if the contribution is in the expertise of the integrator [23]. In terms of delivery time, the rejection time is faster for contributions from core developers, while the acceptance time is faster for causal developers [21].



(a) Interaction plot of test inclusion and contribution contention

(b) Interaction plot of social distance and contribution contention

(c) Interaction plot of prior interaction and contribution contention

Figure 5: Acceptance in relation to test inclusion, social distance and prior interaction [17]

Activeness of the contributors indicates the effect that the amount of discussion has in the decision making of a pull requests. This can be seen in relation to test inclusion, social distance and prior interaction in figure 5. Generally speaking, pull requests with many associated comments indicate uncertainty and may signal controversy [24]. Based on this, pull requests with many associated comments are much less likely to be accepted [24, 17]. It should also be noted that if the reviewer of the pull request is an inactive developer, the pull request has a higher chance of staying open [25].

Submitter's prior interaction looks at the previous pull requests submitted by the developer. Submitting the first pull request is a factor that has the greatest influence in the rejection of the contribution [26]. When the first contribution of a developer occurs via pull request, the chances of rejection are 3.38 times higher. Also, a developer with more prior interaction submitting a pull request leads to a higher acceptance rate, increasing 35.6% [17]. Generally, it is indicative that the more prior interaction the developer has, the bigger the acceptance rate becomes. Thus, there is a positive correlation between the previous pull requests and the acceptance rate [27, 17, 26].

Relation between contributors is looked through 2 lenses. Propensity to trust and social distance, see the table A.1. A high propensity to trust (agreeableness) leads to a higher pull request acceptance rate [28]. Furthermore, social distance has the strongest influence on likelihood of acceptance as compared with other pull requests factors [17]. Social distance increased the acceptance rate when the submitter follows the project manager. [17]

Type of developer indicates whether the developer is either a volunteer or employee in our case. In the case of the developer being a volunteer the rejection rate of the pull request increases [29].

Our main observation is that experience and the contributor 'sticking around' (i.e. continuing after the first (failed) pull request) indicates a giant hurdle to overcome and increases the acceptance rates drastically. For the rest, it should be noted that some of the results were taken from a survey, which made contrasting points, as it is opinion-based, in regards to the data, like in the case of the track record of the developer. Multiple factors increasing the acceptance rate are rather logical, but it is a good thing that the data also shows this. It is, however, a weird phenomenon in figure 3 that the data of 50 to 60 months indicates such a stark difference between success and failure in the acceptance of the pull request. Finally, it should also be noted that most of our papers got their data either from GitHub or from other papers, whom made use of data from GitHub. This, with the fact that the ratio between datamined data and survey is pretty low in regards to the developer, could suggest a bias within our results.

6 Project

In total, 16 papers were used to indicate the factors. In figure 6 a distribution can be seen of how the studies were conducted in these papers. Do note, however, that some papers used several means of conducting the studies. In regards to the projects, datamining of large-scale projects was done through datamining (through small-scale projects or other datamining options) (93.3%). We think that this has to do with the the data that is needed. Most of the data of the projects on distributed version control systems such as GitHub, which means that mining has to be done most of the times. Finally, one medium-scale surveys (6.7%), with 118 developers respectively, were used to attribute outcomes for the factors.

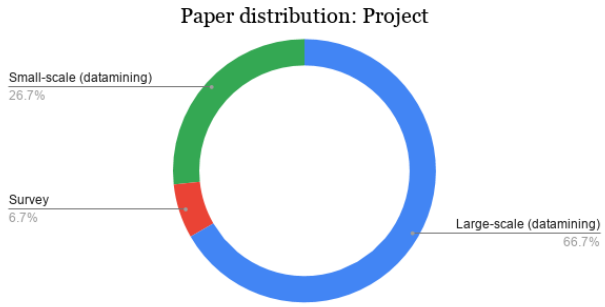


Figure 6: How the studies were conducted in the papers regarding projects.

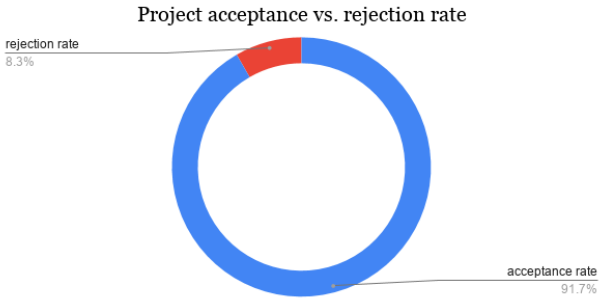


Figure 7: How the outcomes were reached in the studies of the papers regarding the projects.

When we take a look at all the findings of the paper for each factor we see that 91.7% leads to an increase in the acceptance rate of the pull request if the factor is included or of the factor is incremented. 8.3% leads to an decrease in the acceptance rate.

In total, we have 5 different factors for projects, which also contain some of the pull request factors like number of files changed, number of commits.

Pull request system, is described as a piece of software that is used to create pull request instead of a mailing list or an issue tracker. This system makes it easier for a developer to create and review a pull request. This factor leads to a higher acceptance ratio and a quicker review time. [30]

Project age and maturity, is a measure of the lifetime of a project. During examination it was observed that this factor has a high correlation with the programming language. Increasing this factor leads to a lower acceptance rate. [10, 17]

			Model I	Model II	Model III	Model IV
			Pull Request Level		Pull + Submitter Level	Pull + Submitter + Repo Level
	Factor	Variable	Odds Ratio	Odds Ratio	Odds Ratio	Odds Ratio
Pull Request Level		(Intercept)	2.934 ***	2.898 ***	2.845 ***	3.925 ***
	Technical Contribution Norms (H1)	Test Inclusion	1.059 ***	1.023 *	1.114 ***	1.171 ***
		Commit Size	0.849 ***	0.834 ***	0.736 ***	0.738 ***
		Number of Files Changed	1.165 ***	1.152 ***	0.970 ***	0.927 ***
	Social Connection (H2)	Social Distance	1.345 ***	1.461 ***	3.636 ***	2.870 ***
		Prior Interaction	1.423 ***	1.362 ***	1.207 ***	1.356 ***
	Highly Discussed Contributions (H3)	Comments	0.481 ***	0.480 ***	0.414 ***	0.454 ***
	Decision-Making for Highly Discussed Contributions (H4)	*Test Inclusion x Comments*		1.057 ***	1.092 ***	1.106 ***
		Commit Size x Comments		1.101 ***	1.166 ***	1.169 ***
		Files Changed x Comments		1.017 ***	1.043 ***	1.035 ***
		Social Distance x Comments		0.806 ***	0.792 ***	0.796 ***
		Prior Interaction x Comments		1.106 ***	1.246 ***	1.142 ***
Submitter Level	Status in General Community (H5)	Followers			1.060 ***	1.181 ***
	Status in Project (H6)	Collaborator Status			3.904 ***	1.636 ***
Repo Level	Project Establishment (H7)	Repository Age				0.820 ***
		Collaborators				0.954 **
		Stars				0.648 ***
AIC:			633600	630879	506850	461077

Figure 8: Multi-level mixed effects logistic model for pull request acceptance [17]

Programming language, looks at the language that the pull request is written in. Multiple papers found this factor, however not every programming language is taken into aspect. We found some correlation between languages that have

a higher acceptance rate (i.e. C, C#) and language that have a lower one (i.e. C++, Java). However, some languages showed a higher rate in one research and a lower rate in another research. So for some language we could say that it increases or decreases the acceptance rate, but for others we cannot say this. [10, 31]

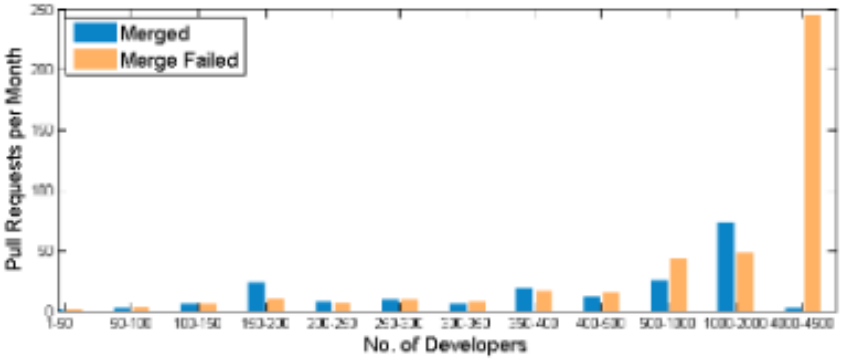


Figure 9: Pull requests vs. Number of Developers [10]

Number of developers has an effect on the success or failure rate of the pull requests. As a factor, it is mainly looked through the means of 'grow' in the developer numbers in time and the effect of said grow. It is known that the more developers participate in the projects, the more pull requests there are. The increase in the pull request also increase the rate of unsuccessful pull requests exponentially. Thus, projects with a large developer crowd may make an excessive number of failed pull requests, as can be seen in figure 9. However, the average number of pull requests does not increase comparatively with higher participation [10, 27, 17]. It does increase almost regularly against increased participation of the developers.

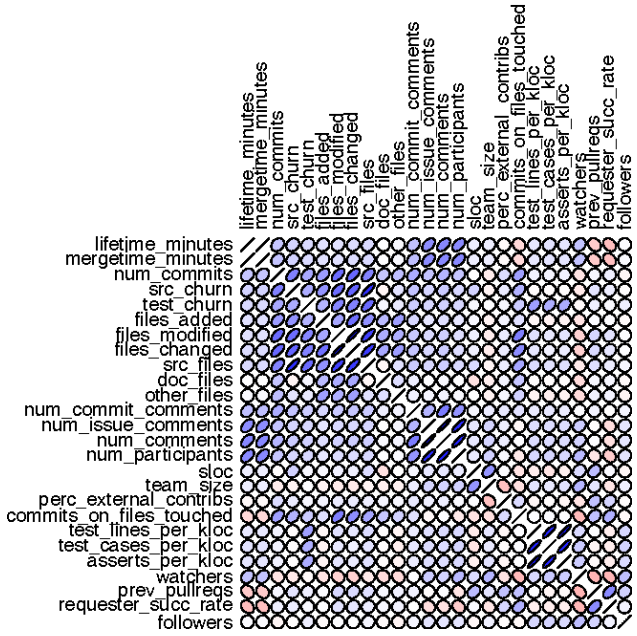


Figure 10: Cross-correlation (Spearman) among all dataset features. Blue color (or right slant) indicates positive correlation, red color (or left slant) is negative correlation. The darker the color, the stronger the correlation. [27]

Popularity of the project, is measured by different means, some paper looked at the rating of the project and some took the amount of followers. Both papers were contradiction each other. Some found an increase if the project was more popular and some found a decrease in the acceptance rate. [27, 17]

Our observation about pull request factors is that the amount of modified files in a pull request have a huge influence on the acceptance rate of the pull request. Also, the purpose of the pull request and the number of commits have a

PDF-XChange Editor Click to BUY NOW! www.pdf-xchange.com

influence, on its acceptance rate. We find that the 'perfect' pull request are small in changes but big in impact. The main observation about project is that the lifetime of a project has the biggest influence on the pull request acceptance. Older project tend to have a lower acceptance rate that new project. We think that this is inline with the complexity of the project. If time increases, on average more features and more lines are added to a project. Most of the time this also increases the complexity of the project.

7 Process

In total there were 17 papers were used in determine all the factors that were influenced by the process of the projects. Figure **??** shows how the studies were conducted in terms of the process. The amount of papers does not correlate with the amount of factors found. This is because there were also papers used in order to validate results from other papers or even contradict the papers.

Most studies of which were used to find the factors were using a form of datamining (55.6%), like the GHTorrent [5], to determine the results from their studies. However an interesting observation is that compared to the other characteristics analyzed during this literature survey (22.2%), more papers used surveys in order to get their results. The rest of the papers conducted their research by looking at singular projects (22.2%).

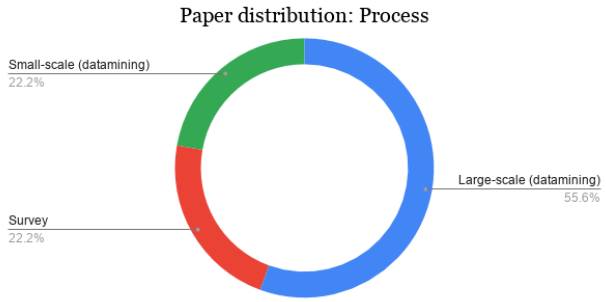


Figure 11: How the studies were conducted in the papers regarding process.

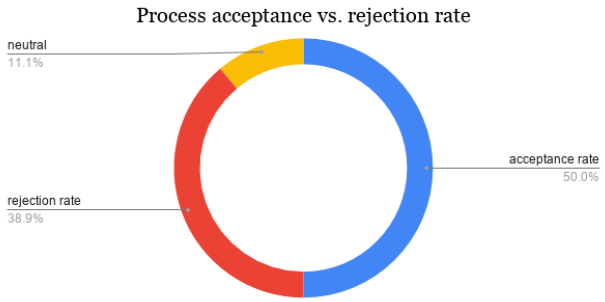


Figure 12: How the outcomes were reached in the studies of the papers regarding the process.

The results of this part of the survey shows that the factors that were found do not necessarily have a trend in them in terms of being favourable for acceptance or not. 50% of conclusions about the factors found lead to an increased likelihood for the contribution being accepted, while on the other hand 39.1% of the conclusions talk about a negative influence of the factor in terms of accepting the contribution. The rest of the papers (11.1%) show a neutral stance. In total there are 10 factors found. The amount of conclusions differs from the amount of factors because there are papers that contradict each other in terms of their conclusion and both of these conclusions were added in the survey. There are a few factors that will be discussed in the next part because of their relevance.

Number of comments on pull request are a significant factor in the process that determines whether or not a contribution gets accepted or not. Comments on the pull request serve as a discussion point where developers can review and share thoughts about the contribution. The consensus is that if there are more comments on the pull request, the contribution is more likely to be accepted [13, 32]. However, there is also a study that claims the exact opposite, mainly that contributions with more comments are less likely to be accepted[17].

The **type of comments on the pull-requests** influence whether or not a contribution gets accepted or not. Comments are still just comments and everyone can comment on pull requests, even if it is not meant to be a review. A study[33] therefore concluded that positive comments affect the pull request acceptance likelihood positively, whereas the percentage of negative comments affects negatively. The study was conducted by scraping 24 million pull requests which had at least five comments and the comments were then classified as either being negative, neutral or positive. From that they could correlate the type of comment to an acceptance likelihood.

Another thing to factor in is the **difficulty of Git**. Git is used for version control and a study concluded that the inconsistency of Git commands and options and undo/redo Git commands are the influence factors of difficulties to developers[34].

Conformation to the projects coding standard is very important when it comes to developing software. A mix of different styles can easily lead to messed up and hard to maintain code. It is for this reason that integrators usually ask for a certain standard to which all the contributions must be held. This can range from the code style to the way the documentation must be kept up to date to following the process of development. It is therefore that a study concludes that a contribution is more likely to be accepted if it adheres the coding practices of the project itself[35].

Reopened pull requests are also a factor. Pull requests can be closed off for various reasons, like the contribution is not feasible for the project or the contribution is already covered in other pull requests. It is also possible for a pull request to be reopened to continue development on the contribution. However, it is concluded that pull requests that are reopened have a lower acceptance rates than non-reopened pull requests.[36]

In modern systems, the use of **Continuous Integration** (CI) is often found in the project. It also factors in with the acceptance rate. The availability of CI in a project hastens the process [37], increases the acceptance rate of pull requests [38] and the amount of processed pull requests increases as well [39]. In even more detail, there is a paper[40] states that PR opened with passed CI builds have 1.5 more chance of being merged than those failing. Such chances increase up to 1.72 if the build passes at the end of the PR discussion.

On the other hand, there is also the use of **automation** that you can consider as a factor that influences the acceptance rate. A study [41] has concluded after analyzing the usage of bots on OSS projects which had more that 2500 stars that automation did not show any consistent statistically significant results across the analyzed projects.

Age of the pull request plays a role in the process as well. Pull requests have various reasons to stay open. Either the project is abandoned or during the development the pull request lacks certain aspects that are desired by the integrators of the project. It is concluded in the literature that a pull request is less likely to be accepted the longer the age [42]

	Participants																			
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Experience	H	H	H	L	H	H	H	H	H	H	L	L	H	L	H	L	L	L	H	L
PR Reviewed	P●	P●	P⊖	A●	A⊖	A⊖	A⊖	T⊖	T⊖	T⊖	P●	P●	P⊖	A●	A●	A⊖	A⊖	T●	T●	T⊖
Decision Evaluation	T✓	–	F✓	–	T✗	F✓	T✗	T✗	–	T✗	T✓	T✓	F✓	F✗	T✓	F✓	T✗	T✓	T✓	T✗
Overview																				
Code Signals	67%	66%	66%	21%	59%	25%	70%	83%	27%	73%	60%	70%	25%	69%	52%	75%	86%	63%	58%	27%
Technical Signals	26%	30%	28%	48%	29%	49%	22%	11%	57%	17%	31%	24%	62%	25%	42%	18%	7%	28%	38%	56%
Social Signals	7%	4%	6%	31%	12%	26%	8%	6%	16%	10%	8%	5%	13%	7%	6%	7%	7%	9%	3%	17%
Code Signals																				
After Code Snippet (AC)	97%	90%	88%	80%	98%	80%	89%	96%	71%	74%	93%	100%	28%	94%	97%	86%	89%	82%	99%	54%
Before Code Snippet (BC)	3%	10%	12%	20%	2%	20%	11%	4%	29%	26%	7%	–	72%	6%	3%	14%	11%	18%	1%	46%
Technical Signals																				
Contribution Activity (CA)	47%	65%	48%	36%	–	18%	11%	–	35%	20%	19%	5%	18%	24%	43%	12%	11%	28%	5%	11%
Commit Details (CD)	7%	1%	–	2%	–	2%	19%	–	–	–	–	1%	9%	3%	–	3%	–	9%	3%	3%
Contribution Heat Map (HM)	14%	16%	17%	12%	–	8%	11%	25%	13%	8%	23%	28%	10%	14%	14%	13%	26%	19%	44%	18%
Pull Request Title (PT)	2%	3%	3%	9%	63%	23%	20%	18%	17%	28%	33%	24%	18%	7%	8%	6%	25%	15%	13%	5%
Popular Repositories (RE)	23%	15%	17%	28%	12%	26%	11%	46%	17%	13%	14%	11%	13%	23%	23%	32%	2%	16%	30%	45%
Submission Details (SD)	6%	–	15%	13%	25%	24%	28%	10%	18%	31%	12%	32%	32%	28%	13%	35%	36%	13%	6%	18%
Social Signals																				
Avatar Image (AI)	25%	20%	51%	28%	13%	16%	7%	64%	26%	52%	35%	33%	7%	24%	50%	21%	74%	42%	35%	46%
Display Name (DN)	16%	24%	4%	8%	–	3%	5%	12%	11%	–	–	–	5%	8%	14%	5%	14%	11%	–	11%
Followers/Following (FF)	6%	19%	19%	6%	–	11%	–	3%	–	–	–	–	–	2%	–	2%	–	–	–	–
Repository Popularity (RE)	–	–	–	–	–	–	–	5%	–	–	–	–	–	–	–	–	–	1%	–	–
Repository Stars (RS)	45%	21%	17%	–	12%	32%	–	3%	14%	–	3%	–	–	8%	–	22%	–	5%	–	13%
To Merge (TM)	6%	4%	–	42%	70%	29%	39%	–	28%	13%	58%	57%	63%	44%	20%	24%	3%	11%	65%	21%
User Details (UD)	2%	13%	9%	16%	5%	9%	49%	14%	20%	35%	5%	10%	25%	13%	16%	26%	10%	29%	–	9%

Figure 13: Table that shows social and technical skills measured with participation fixation using an eye tracker[43]

An interesting factor is **social and technical skills**. Among these social skills can be classified the display name of the contributor, the profile picture and popularity of their previous projects. On the other hand among the classification of

PDF-XChange Editor Click to BUY NOW! www.pdf-xchange.com

technical skills are commit details, contribution activity and submission details. The conclusion of this study is that not only the code but also other technical and social aspects are influencing the review process of the pull request.[43]

During the process of the project, a lot of different types hurdles can occur. It is therefore that the **prioritization of work** can influence the acceptance rate of a pull request. The main problems when working with pull requests is maintaining project quality and prioritization [44, 42]. This can mean that a pull request might not make it not because the quality of the pull request is not good enough, but mainly because the pull requests contains functionality that is not needed in the current stage of the project.

Observing the characteristics of the process of a pull request led us to conclude that most of the factors are influenced by the conformation to the standards the project adheres. Software projects naturally want contributions of suitable quality and that show unity with the work that is already there. More often than not a factor seems to be linked to either good quality of code or bad quality of code. The literature concludes that if you want your pull request accepted in the process of the pull request, it benefits if the pull requests is beneficial for the project and understands its philosophy and practices.

8 Code

In total, 23 papers were used to indicate the factors. In figure 14 we show the distribution of how the studies were conducted in these papers.

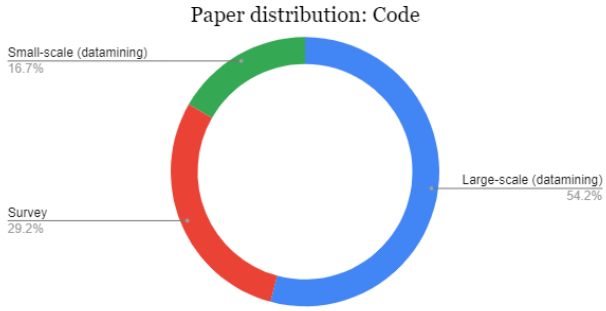


Figure 14: How the studies were conducted in the papers regarding the code.

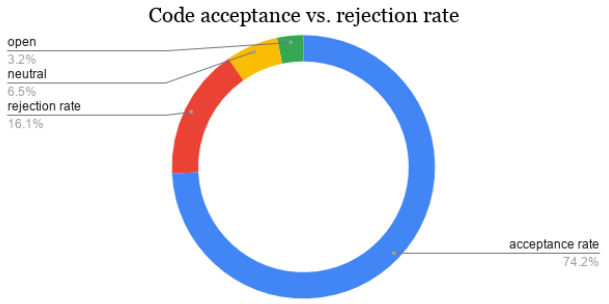


Figure 15: How the outcomes were reached in the studies of the papers regarding the code.

When looking at the outcomes we find that 48.4% of the papers show factors that lead to an increase of the acceptance rate or decrease of the rejection rate of pull requests. 41.9% of the papers show factors that lead to an increase of the rejection rate or decrease of the acceptance rate of pull requests. 6.5% of the papers show factors that do not show a clear difference in acceptance rate nor rejection rate. Only 3.2% of the papers show factors that lead to an open pull request. In figure 15 we show the distribution of papers talking about acceptance vs rejection, whether it is an increase or decrease.

In total, we found 12 different code related factors, these are as followed:

The **contribution quality** talks about whether the non-functional characteristics (e.g. style of code) of the code are understandable and elegant. There are studies that conclude that the pull request is more likely to be accepted if non-functional characteristics of the code are understandable and elegant.[8, 45, 46]. There is however a study that suggests that the quality of the code submitted in the pull request does not influence at all in terms of acceptance.[47] The interesting part is that all the papers that suggested that it matters have done surveys among the integrators of pull requests [8, 45, 46], while the study that suggests that the quality does not matter used machine learning techniques to classify and measure its accuracy on the data.[47]

Comments in source code looks at the amount of comments added in the source code. A statistical model analyzing the association of different pull request, submitter, and repository measures of contributions with the likelihood of the contribution being accepted was created. This showed that an increase in comments in the source code increases the pull request acceptance rate. [32]

Number of blank lines looks at the amount of blank lines in the source code of a pull requests in the project Ansible3. This open source project was selected because it is the fourth on the list of most reviewed GitHub projects in 2017 and it has a significant process. There are around 5.9K contributors in Ansible3, which is the tenth most-discussed repository. The outcome is evaluated by looking at code contributions of the developers using the pull request acceptance rate. This paper states that an increase in blank lines on the source decreases the pull-request acceptance rate. [48]

Number of changed lines looks at the number of changed lines included in the pull request. Multiple researches show that an increase in changed lines of code have a negative effect on the acceptance rate. Different sorts of research, like a survey and datamining, all had this outcome. [17, 48, 49, 50, 51]

A certain data analysis also showed the negative effect on the merge decision. [32] However, the developers they asked about the influence of the size of the pull requests (in terms of lines of code) had split opinions. Only 56% of the developers agreed that the number of changed lines affects the merge decision. This paper states, “A possible explanation for this is that a larger PR has a higher chance of containing more than one “atomic” change, which is against PR submission policies in many software projects. While developers may not mind accepting a large PR that is coherent and single-purpose, they may feel more negatively about a large PR that is a collection of loosely related changes.”

No	Repository	Reviews	Pull request #	Contributors #
1	DefinitelyTyped/ DefinitelyTyped	800	68	5,952
2	Kubernetes/ kubernetes	680	1,067	1,645
3	Homebrew/ homebrew-core	580	66	6,907
4	Ansible/ansible	550	1,431	3,391
5	Nodejs/node	480	128	1,964
6	NixOs/nixpkqs	480	832	1,597
7	Apache/spark	450	516	1,218
8	Rust-lang/rust	390	144	2,056
9	Symfony/ symfony	340	159	1,622
10	Tensorflow/ tensorflow	340	200	1,432

Figure 16: The details of the top ten most reviewed projects[48]

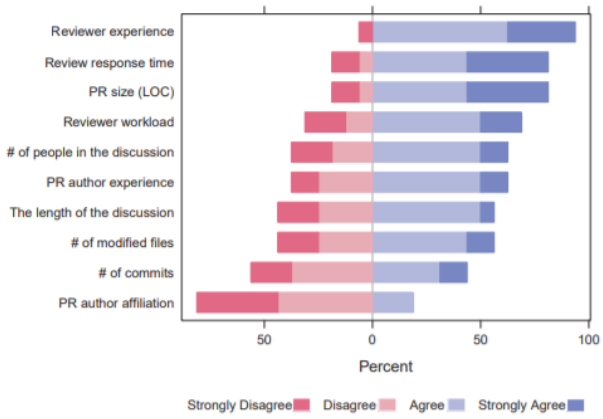


Figure 17: Factors influencing PR review outcome[32]

Code style looks at the inconsistency of the code style for each pull request. This paper found that the inconsistency of code style of pull requests decreases the pull request acceptance rate, though the effect of the inconsistency on the acceptance rate is small. [52] To see how the code inconsistency was measured, look at A.4.

Number of tests, this factor is measured by the existence of test code in the pull request. One paper stated "In our sample, 33% of the pull requests included modifications in test code, while 4% modified test code exclusively. Of the pull re-quests that included modifications to test code, 83% were merged, which is similar to the average" [1] These results were collected by looking at 291 projects from multiple programming languages. Another paper stated that the existence of a test suite increases the acceptance rate, this information was conducted through surveys with a diverse group of GitHub users.[53] Also, another paper stated that the existence of a test suite increases the acceptance rate. This information was conducted by looking at 12,482 projects from the GitHub API. [14]

Annotations, this factor is measured by looking at disciplined and undisciplined annotations in 110 open source projects. Undisciplined annotations are annotations of brackets that to not align with the code or individual tokens. In these projects undisciplined annotations were changed to disciplined annotations by manually refactoring one annotation per project. Submitting pull requests with these changed showed that the acceptance rate increases with disciplined annotation. [54]

The **functionality of the code** is also factor that determines whether or not the pull request gets accepted or denied, This must be reviewed and deemed suitable for the project. Multiple requests doing the same thing or a pull request that does not change anything that is needed are examples of this. In the literature it is found that if the functionality of the code is not needed and/or a duplicate to other pull requests, the rejection rate of the pull request increases [55]. This study however has the prerequisite that the contribution is made by a quasi-contributor, which is someone identified as a newcomer to the project who had no previous accepted contribution to the project.

PDF-XChange Editor Click to BUY NOW! www.pdf-xchange.com

Number of commits, is measured by observing the pull request and observing the amount of commits that have been made in the pull request. The number of commits can indicate that the pull request has a lot of changes, since every commit can be considered as one contribution. Different papers have found this as factor influencing the acceptance rate. However, the outcomes were divided. Some papers found that the acceptance rate increases if the number of commits increases increase [26, 27], while others found that the acceptance rate decreases. [49]

Complexity, the cyclomatic complexity is used to compute this factor. If the cyclomatic complexity of a piece of code increases, it is harder to understand for developers. This factor leads to a lower acceptance ratio because most of the project will have an open status. The review time of the pull request also increases. [56]

Purpose of the Pull-request, looks at what problem the pull request solves, it could be a bug fix, new feature, etc. Multiple papers found this factor and mentioned that bug fixes and documentation changes were accepted quicker and accepted more often than new features or big changes. So we could say that the purpose of the pull request does have an influence on the acceptance ratio. [50, 57]

Modified files, is measured by observing the pull request and observing the amount of files that have been modified. 4 papers found this factor and all 4 paper stated that if the amount of files that were modified increases the acceptance rate of the pull request decreases. For this unanimous result we can conclude that if the number of modified files increases the acceptance rate decreases. [48, 31, 58, 27, 17]

Expected is that the main code related factor would be the quality of the commits added in the pull-request. By looking at all the code related factors that influence the acceptance rate of pull-requests, our main observation is that the amount of code that is edited in a pull request has a large influence on the acceptance rate. We find that the pull requests are most likely to be accepted if the changes are small and clear. Small changes in the sense of literally not many changes in lines, but also in the coding style. Clear changes in the sense that there are no useless changes like blank lines, understandable commenting and disciplined annotation.

9 Discussion

In this section, we discuss the decisions made during the process of writing this literature survey, while also evaluating the process of researching the topic.

Firstly, almost all the literature was found by using Google Scholar. While other search engines are available, most of the literature has been retrieved with Google Scholar. It should be noted that we have collected all the papers in the relevant time period (2009-2019) related to our search terms available on Google Scholar and using snowballing we found more papers without the direct use of Google Scholar.

Secondly, the topics of the papers were sometimes not relevant to our main research question and researched something that was not of direct interest to our literature review. However, these papers were still used because they did contain useful information about a factor that was indeed used in pull-based development.

As stated before, it should also be noted that most of our papers got their data either from Github or from other papers, whom made use of data from Github. This, with the fact that the ratio between datamined data and survey is pretty low in regards to the developer, could suggest a bias within our results.

Moreover, at the start of the project there was a division in characteristics. It could be argued that these four characteristics (developer, project, process, code) do not cover the whole range of factors, but the subsequent collection did not search on characteristics, but rather single factors whom were subsequently categorised into these characteristics.

Also, it could be argued that in the fast evolving discipline of software development the time frame used to search these factors could lead to outdated factors, whom were once important but could now be irrelevant to the process. The time frame was however chosen because it could be argued that this would deliver the survey a good amount of relevant factors, while also having a good quantity of factors as well.

Furthermore, the categories process and code rely more on surveys compared to the other categories, which means that the factors within these categories might have a bias from the developers that were surveyed, because it is a subjective factor and it is not measured objectively.

Finally, the methodology states the search terms used in the literature survey. It could happen that these do not retrieve all the relevant papers. However, all the search terms used were the most relevant to the topic and to our understanding we have used all the relevant search terms.

When looking at the outcomes we find that more papers show factors that lead to an increase of the acceptance rate of pull requests in comparison to outcomes based on the rejection rate. In the future more papers should give insight on the factors influencing the rejection of pull requests.

10 Conclusion

The goal of this work was to obtain factors from papers about the pull-based development model and with the obtained factors look at the correlation of every factor with the acceptance rate. From the initial point of view the code quality is the only factor that has an influence on the decision making. However, when we analyzed around 80 papers, we found that this was not the case. Multiple varying factors that have an influence on the decision making were found. The distribution on how these factors were obtained can be seen in figure 18. The total rates in terms of acceptance (73.8%), rejection (20.2%), open (1.2%) and neutral (4.8%) can be seen in figure 19.

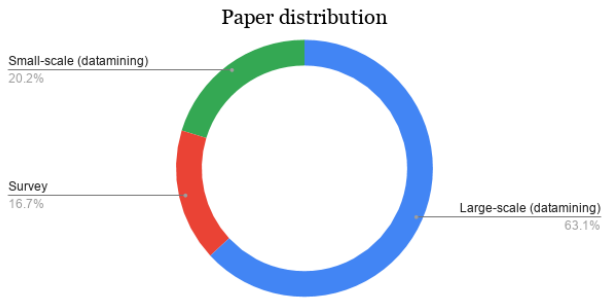


Figure 18: Total distribution of methods used of all papers.

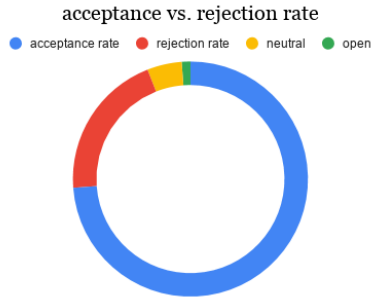


Figure 19: Total rates of acceptance vs. rejection.

After analyzing the type of factors that were found, we concluded that 4 categories of factors do influence the decision making. The categories are developer, project, process and code. For the developer we intrinsically look at factors related to the skill of the developer (i.e. track-record), the environment of the developer (i.e. geographical location) and the social factors (i.e. prior interaction). For the project we intrinsically look at factors related to the unchangeable project settings (i.e. the age of the project) and changeable project settings (i.e. programming language). For the process we view the pull based development model, as what happens before, during and after the pull request. With this, you can view the pull request process as a certain pipeline, with factors related before you go into the pipeline (i.e. difficulty of Git), during the process (i.e. continuous integration) and after the pipeline (i.e. contribution quality). For the code we did not found any intrinsic factors. All factors were related to either the amount of the code modified (i.e. number of changed lines) or the absence of some code (i.e. code style). To summarize, factors related to the contributors and integrators fall under developer, those related to the actual writing of code fall under code, those related to the life-cycle of the pull request fall under process and, finally, the factors related to specifications surrounding a project fall under project. Furthermore, we've found in total 42 different factors influencing the acceptance rate of pull-request. From these, 6 are code related, 12 are developer related, 10 are project related and 14 are related to the process. We noticed after comparing the distributions of the categories that most of the data was gathered through datamining, with the exception of process (and code) in which a sizeable portion of the factors come from surveys.

To answer what factors influence pull request decision making, we created a table with the outcomes, per factor, and how they were collected. With this, a positive effect in the decision making indicates a higher acceptance rate or lower rejection rate, while a negative effect in the decision making indicates a higher rejection rate or lower acceptance rate. Our main findings are as follows:

1. **Developer**. Multiple papers found that the experience, the track record of the developer, the affiliation of the developer with the project and the prior interaction between the developer and core contributor influences the decision making of the pull request. And all papers that found these factors individually stated that an increase in these factors leads to the a positive effect in the decision making. Also, it is indicative in multiple papers that casual developers or first-time contributors have a higher negative effect on the decision making.
2. **Project**. Multiple papers found that the number of developers and the age of the project influence the decision making of the pull request. And all papers that found these factors individually stated that an increase in these factors lead to a negative effect in the decision making. Furthermore, for the purpose of the pull request and the popularity of the project, multiple papers that found these factors stated varying conclusions related to the effect in the decision making. For example, in the case of the purpose of the pull request some papers found that a bug fix leads to a positive correlation on the acceptance, while others stated that a pull request that adds a new feature leads to a negative correlation on the acceptance. And in the case of popularity, some papers stated that when an increase in popularity occurs, it had a positive effect and some stated that it had a negative effect on the decision making.
3. **Process**. Multiple papers state that number of comments and the type of comments on a pull request has an influence on the decision making. For the papers that found the number of comments as an influencing factor, some found that an increase in comments has a positive effect while other papers found a negative effect. However, a majority found that it an increase has a negative effect on the decision making. As for the papers that found type of comments an influencing factor, all of them stated that negatively coloured comment has a negative effect on the decision making and positive comment has a positive effect on the decision making. It is generally accepted, within the papers that found the age of a pull request as an influencing factor on the decision making, that an increase in this factor leads to a negative effect on the decision making.
4. **Code**. As said before, code quality is initially expected to be the main influencing factor, but besides the factor code quality, multiple papers found that number of changed lines, modified files and number of tests are a big influence on the decision making. Small and simple changes have the most positive effect on the decision making.

References

[1] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM, 2014.

[2] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. Communication in open source software development mailing lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 277–286, Piscataway, NJ, USA, 2013. IEEE Press.

[3] Brian de Alwis and Jonathan 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*, pages 36–39. IEEE Computer Society, 2009.

[4] N.B. Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35:5–9, 2010.

[5] Georgios Gousios. The ghtorent dataset and tool suite. In *Proceedings of the 10th working conference on mining software repositories*, pages 233–236. IEEE Press, 2013.

[6] Linux Torvalds. git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git, January 2012.

[7] Mozilla. Firefox input, http://input.mozilla.org/, 2012.

[8] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 358–368. IEEE Press, 2015.

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

[10] Mohammad Masudur Rahman and Chanchal K Roy. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 364–367. ACM, 2014.

[11] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, 3:e111, 2017.

[12] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. The influence of non-technical factors on code review. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 122–131. IEEE, 2013.

[13] Yujuan Jiang, Bram Adams, and Daniel M German. Will my patch make it? and how fast?: Case study on the linux kernel. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 101–110. IEEE Press, 2013.

[14] Jason Tsay, Laura Dabbish, and James Herbsleb. 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*, pages 144–154. ACM, 2014.

[15] A. Ihara A. Jongyindee, M. Ohira and K. Matsumoto. Good or bad committers? a case study of committers cautiousness and the consequences on the bug fixing process in the eclipse project. In *The Joint Conference of the 21th International workhop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pages 116–125. IWSM/MENSURA2011, 2011.

[16] Manoel Limeira de Lima Júnior, Daricélio Moreira Soares, Alexandre Plastino, and Leonardo Murta. Developers assignment for analyzing pull requests. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1567–1572. ACM, 2015.

[17] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366. ACM, 2014.

[18] Damien Legay, Alexandre Decan, and Tom Mens. On the impact of pull request decisions on future contributions. *arXiv preprint arXiv:1812.06269*, 2018.

[19] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 495–504, New York, NY, USA, 2010. ACM.

[20] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM IEEE International Symposium on Empirical Software Engineering and Measurement*, page 22. ACM, 2018.

[21] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. The secret life of patches: A firefox case study. In *2012 19th Working Conference on Reverse Engineering*, pages 447–455. IEEE, 2012.

[22] Amiangshu Bosu and Jeffrey C Carver. Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*, page 33. ACM, 2014.

[23] P.C. Rigby and M.-A Storey. Understanding broadcast based peer review on open source software projects. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 541–550. ACM, 2011.

[24] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology*, 84:48–62, 2017.

[25] Jing Jiang, David Lo, Xinyu Ma, Fuli Feng, and Li Zhang. Understanding inactive yet available assignees in github. *Information and Software Technology*, 91:44–55, 2017.

[26] Daricélio Moreira Soares, Manoel L de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 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)*, pages 960–965. IEEE, 2015.

[27] Georgios Gousios and Andy Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 368–371. ACM, 2014.

[28] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. A preliminary analysis on the effects of propensity to trust in distributed software development. In *Proceedings of the 12th International Conference on Global Software Engineering*, ICGSE '17, pages 56–60, Piscataway, NJ, USA, 2017. IEEE Press.

[29] Gustavo Pinto, Luiz Felipe Dias, and Igor Steinmacher. Who gets a patch accepted first?: Comparing the contributions of employees and volunteers. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '18, pages 110–113, New York, NY, USA, 2018. ACM.

[30] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 871–882. ACM, 2016.

[31] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1541–1546. ACM, 2015.

[32] Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart de Water. Studying pull request merges: A case study of shopify's active merchant. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, pages 124–133, New York, NY, USA, 2018. ACM.

[33] Rahul Iyer. Effects of personality traits and emotional factors in pull request acceptance. Master's thesis, University of Waterloo, 2019.

[34] Yusuke Saito, Kenji Fujiwara, Hiroshi Igaki, Norihiro Yoshida, and Hajimu Iida. How do github users feel with pull-based development? In *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 7–11. IEEE, 2016.

[35] B.D. Sethanandha, Bart Massey, and William Jones. Managing open source contributions for software project sustainability. pages 1 – 9, 08 2010.

[36] Jing Jiang, Abdillah Mohamed, and Li Zhang. What are the characteristics of reopened pull requests? a case study on open source projects in github. *IEEE Access*, 7:102751–102761, 2019.

[37] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 59(8):080104, 2016.

[38] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 84–94, Piscataway, NJ, USA, 2017. IEEE Press.

[39] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 805–816. ACM, 2015.

[40] Fiorella Zampetti, Gabriele Bavota, Gerardo Canfora, and Massimiliano Di Penta. A study on the interplay between pull request review and continuous integration builds. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 38–48. IEEE, 2019.

[41] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW):182:1–182:19, November 2018.

[42] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: the contributor's perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 285–296. IEEE, 2016.

[43] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. Beyond the code itself: how programmers really look at pull requests. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*, pages 51–60. IEEE Press, 2019.

[44] Di Chen, Kathryn T Stolee, and Tim Menzies. Replication can improve prior results: a github study of pull request acceptance. In *Proceedings of the 27th International Conference on Program Comprehension*, pages 179–190. IEEE Press, 2019.

[45] Mehrdad Nurolahzade, Seyed Mehdi Nasehi, Shahedul Huq Khandkar, and Shreya Rawal. The role of patch review in software evolution: an analysis of the mozilla firefox. In *EVOL/IWPSE*, 2009.

[46] M. V. Mäntylä and C. Lassenius. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3):430–448, 2009.

[47] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, and Davide Taibi. Does code quality affect pull request acceptance? an empirical study. *arXiv preprint arXiv:1908.09321*, 2019.

[48] Panthip Pooput and Pornsiri Muenchaisri. Finding impact factors for rejection of pull requests on github. In *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, ICNCC 2018, pages 70–76, New York, NY, USA, 2018. ACM.

[49] Nikhil Khadke, Ming Han Teh, and Minghan Shen. Predicting acceptance of github pull requests. 2012.

[50] Jennifer Marlow, Laura Dabbish, and Jim 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*, pages 117–128. ACM, 2013.

[51] Passakorn Phannachitta, Pijak Jirapiwong, Akinori Ihara, Masao Ohira, and Ken-Ichi Matsumoto. Understanding oss openness through relationship between patch acceptance and evolution pattern. In *Proceedings of the International Workshop on Empirical Software Engineering in Practice 2011 (IWESEP 2011)*, 2011.

[52] Weiqin Zou, Jifeng Xuan, Xiaoyuan Xie, Zhenyu Chen, and Baowen Xu. How does code style inconsistency affect pull request integration? an exploratory study on 117 github projects. *Empirical Software Engineering*, pages 1–33, 2019.

[53] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 112–121. IEEE Press, 2013.

[54] Romero Malaquias, Márcio Ribeiro, Rodrigo Bonifácio, Eduardo Monteiro, Flávio Medeiros, Alessandro Garcia, and Rohit Gheyi. The discipline of preprocessor-based annotations does #ifdef tag n't #endif matter. In *Proceedings of the 25th International Conference on Program Comprehension*, ICPC '17, pages 297–307, Piscataway, NJ, USA, 2017. IEEE Press.

[55] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A. Gerosa. Almost there: A study on quasi-contributors in open source software projects. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 256–266, New York, NY, USA, 2018. ACM.

[56] Vishal Midha, Rahul Singh, Prashant Palvia, and Nir Kshetri. Improving open source software maintenance. *Journal of Computer Information Systems*, 50(3):81–90, 2010.

[57] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. A study of external community contribution to open-source projects on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 332–335. ACM, 2014.

[58] T. Zimmerman G. Jeong, S. Kim and K. Yi. Improving code review by predicting reviewers and acceptance of patches. *ROSAEC-2009-006*, 2009.

[59] Charles J Clopper and Egon S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[60] Corinne A Moss-Racusin, John F Dovidio, Victoria L Brescoll, Mark J Graham, and Jo Handelsman. Science faculty's subtle gender biases favor male students. *Proceedings of the National Academy of Sciences*, 109(41):16474–16479, 2012.

[61] Foyzur Rahman and Premkumar Devanbu. Ownership, experience and defects: A fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500, 2011.

[62] Murtuza Mukadam, Christian Bird, and Peter C Rigby. Gerrit software code review data from android. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 45–48. IEEE, 2013.

[63] Yguaratã Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, Tassio Ferreira Vale, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process*, 26(7):620–653, 2014.

[64] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.

[65] Jing Liu, Jiahao Li, and Lulu He. A comparative study of the effects of pull request on github projects. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 313–322. IEEE, 2016.

[66] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. Wait for it: determinants of pull request evaluation latency on github. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 367–371. IEEE, 2015.

[67] Yao Lu, Xinjun Mao, Gang Yin, Tao Wang, and Yu Bai. Using pull-based collaborative development model in software engineering courses: A case study. In *International Conference on Database Systems for Advanced Applications*, pages 399–410. Springer, 2017.

[68] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey. Investigating technical and non-technical factors influencing modern code review. *Empirical Softw. Engg.*, 21(3):932–959, June 2016.

[69] A. Ihara M. Ohira P. Phannachitta, P. Jirapiwong and K. Matsumoto. An analysis of gradual patch application: A better explanation of patch acceptance. In *The Joint Conference of the 21th International workhop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pages 106–115. IWSM/MENSURA2011, 2011.

[70] Alberto Bacchelli, Marco D'Ambros, and Michele Lanza. Extracting source code from e-mails. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, ICPC '10, pages 24–33, Washington, DC, USA, 2010. IEEE Computer Society.

[71] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. Extracting structural information from bug reports. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, pages 27–30, New York, NY, USA, 2008. ACM.

[72] Peter Weissgerber, Daniel Neu, and Stephan Diehl. Small patches get in! In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, pages 67–76, New York, NY, USA, 2008. ACM.

[73] Rajdeep Grewal, Gary L. Lilien, and Girish Mallapragada. Location, location, location: How network embeddedness affects project success in open source systems. *Manage. Sci.*, 52(7):1043–1056, July 2006.

[74] Jin Xu, Yongqin Gao, S. Christley, and G. Madey. A topological analysis of the open souce software development community. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 198a–198a, Jan 2005.

[75] Daricélio M Soares, Manoel L de Lima Júnior, Alexandre Plastino, and Leonardo Murta. What factors influence the reviewer assignment to pull requests? *Information and Software Technology*, 98:32–43, 2018.

[76] Yida Tao, Donggyun Han, and Sunghun Kim. Writing acceptable patches: An empirical study of open source project patches. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 271–280. IEEE, 2014.

A Table: factors influencing pull-request decision making

Developer				
Factor	**How the factor is measured**	**Outcomes**	**How the study is conducted**	**Explanation**
Gender of the developer	First, the author of the paper [11] extracts the email addresses of GitHub users in GHTorrent [5]. For each email address, the author uses the search engine in the Google+ social network to search for users with that email address and parse their about data and include only the genders 'Male' and 'Female'.	This paper [11] states that a female developer submitting a pull-request leads to a higher acceptance rate of the pull-request, if their gender is not identifiable. Pull requests made by women were less likely to be accepted than those made by men.	Analyzing GitHub data through GHTorrent [5] and evaluate whether pull requests from women are accepted less often. Out of 4,037,953 GitHub user profiles with email addresses, the authors were able to identify 1,426,127 (35.3%) of them as men or women through their public Google+ profiles. The merge rate was calculated, with the corresponding confidence interval. [59].	The study, based on [60], hypothesized that pull requests made by women were less likely to be accepted than those made by men. Prior work on gender bias in hiring suggests that this hypothesis may be true. The hypothesis is not only false, but it is in the opposite direction than the author expected. Women tend to have their pull requests accepted at a higher rate than men.
Experience of the developer	The authors of the paper [12] extracted, prepocessed, identified the factors affecting review delays and outcomes and performed data analysis on the WebKit code review data from Bugzilla[1]. In order to assess whether contributor experience influence review acceptance, the paper calculates the number of submitted changes for each contributor and then discretizes patch owners according to their contributions.	The paper [12] concludes that more experience leads to faster response time and greater acceptance rate. Developer experience plays a major role during code review.	The study was conducted through extracted this data by scraping Bugzilla for all patches submitted between April 12, 2011 and December 12, 2012. The data consists of 17,459 bugs, 34,749 patches, 763 email addresses, and 58,400 review-related flags.	The authors correlate experience of the contributor to activity of the developer. So the more active the developer is, the more experienced they deem him.
	The paper [13] measures data through extracting data from an OSS project and putting it through analysis. Developers who have had a commit being merged into Linus Torvalds' repository [6] before are taken into consideration as a positive outcome.	The result of the paper [13] indicates that the contribution is more likely to be accepted with an experienced contributor.	The authors datamined and analyzed the patched that make it to an official release within Linus Torvald's repository of the Linux Kernel [6], which contains all commit information of accepted patches from June 26, 2005 to December 31, 2012.	This study [13] mirrors the results by other studies conducted on this factor.

[1] https://bugs.WebKit.org/

	The paper [14] created and analyzed a dataset of both interview data and contribution discussions from the social open source software hosting site GitHub.	The contribution is more likely to be not be re-opened with an experienced developer [15, 14]. "The submitter's prior interaction on a project also had a positive association with acceptance." [14]	From a larger dataset of 659,501 pull requests across 12,482 GitHub projects, the authors created a sample of highly discussed pull requests. Highly discussed indicates that the number of commits is one standard deviation higher than the mean in the dataset.	This study mirrors the results by other studies conducted on this factor.
	The authors look at the pull requests in relation to developer experience in months. The factor is measured based on these two parts within the MSR challenge dataset [5].	Contributors with 20 months to 50 months of experience are found the most productive and have the highest acceptance rate. Contributors within that time-frame have greater merge successes as opposed to merge failures. This also holds within 60 to 70 months. Contributors within a span of 10 to 20 months have a greater failure rate and, surprisingly, this is also the case for contributors between 50 to 60 months. This indicates, as the paper [10] states, that more experience seems to indicate a greater acceptance rate.	Collection of the committers activities and rewriting it in the form of patterns that emerge. In this study, the authors conduct a comparative study between successful and unsuccessful pull requests made to 78 GitHub base projects by 20,142 developers from 103,192 forked projects. [10]	"Number of developers involved into a project along with their working experience with the project are also two contributing factors that can influence the success and failure rate of the pull requests." [10]
Affiliation of contributor	The authors of the paper [12] extracted, prepocessed, identified the factors affecting review delays and outcomes and performed data analysis on the WebKit code review data from.	According to the paper [12], an integrator is more likely to accept or be positive if the contributor is affiliated with the organization of the project itself.	The study was conducted through extracted this data by scraping Bugzilla for all patches submitted between April 12, 2011 and December 12, 2012. The data consists of 17,459 bugs, 34,749 patches, 763 email addresses, and 58,400 review-related flags.	From pair-wise comparison, the authors found that there is statistically significant difference between positivity of Apple reviewers towards their own patches to the patches of both Google and 'the rest'. The other pair that was statistically different is positivity of Google reviewers between their own patches and patches from 'the rest'. Possible explanations for this include that there is a clear bias among Apple reviewers, or that Apple patches are of extreme quality, or that Apple applies some form of internal code review process.

	The paper [16] predicted, through a statistical model, with a random forest algorithm, the acceptance rates.	This paper states that a developer that follows a core contributor has a higher increase in acceptance rate (187%) and a requester with many followers has also has a higher increase (18.1%)	Analyzing open source projects hosted by GitHub, extracted by the GHTorrent tool [5] and exported to the database management system MySQL to a database. This database contains 3,200,428 pull requests distributed in 8,510,504 projects, many of which use the methodology of contribution based on pull requests. Considering the projects that are not forks, 68 have more than 2,000 registered pull requests.	The followers of a developer and whether a developer follows a core contributor apparently improves the likelihood of a merge.
	This paper [17] states that Contributor status[2] is a dichotomous variable for the user's collaborator status within the project.	This paper [17] says "Perhaps unsurprisingly, when submitters with commit access choose to create pull requests instead of directly merging code, their pull requests are more likely to be accepted than non-collaborators. Being a collaborator on a project increases the likelihood of contributions being accepted by 63.6%."	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Track record of the developer	By asking 749 integrators through an anonymized survey [8] and datamining extracted PRs through GHTorrent [5]. Also used in this and many other papers is this dataset [27].	The pull-request is not of high priority, based on the survey. However, a known contributor has a higher chance of acceptance [8]. In the case of the papers, [18, 19], a contributor's PRs are more likely to be accepted when he has submitted more PRs previously. Furthermore, a contributor that has contributed faulty code has lower file level contribution than background code owners [61].	A large-scale two-round survey of 749 integrators, split into three logical sections: demographic information, multiple choice or Likert-scale questions, open-ended questions.	Based on the findings of the survey the developer's track record is not commonly used as a criterion to assess or prioritize contributions by. "The track record is mostly used as an auxiliary signal." [8] "In this paper, they provide preliminary quantitative results showing that a contributor's PRs are more likely to be accepted when he has submitted more PRs previously." [18]

[2] In GitHub, a collaborator for a project has direct commit access to the repository. Therefore, they do not need to perform the pull request process in order to merge code contributions into the project. However, interviews with GitHub users indicate that many collaborators opt to create pull requests for code contributions despite having commit status. Often, this is done to allow other users to review changes before accepting the code contribution.

	This paper [49] found the factor by extracted 3000 pull requests events that were made in the window from 04/01/2012 to 04/14/2012. After data sanitization and filtering, this resulted in 2734 usable data points. Then they ran queries on GitHub Archive using Google BigQuery to get information such as the number of successful pull requests and total number of pull requests made for each contributor and repository	This paper says "The track record of the contributor plays a role in pull request acceptance, since it is an indicator of the quality of their work."[49] This means that the bigger the contributions role of a developer the higher the acceptance rate of the pull request will be.	The papers states that the primary data source or this research was the GitHub Archive, data that is available for querying via Google's BigQuery. After that Google BigQuery was used to extract relevant data by filtering data on related event types. Once a resulting data has been generated using Google BigQuery, the data was exported over to Google Cloud Storage in Comma Separated Values (CSV) format, some preprocessing was done, and the date was imported into Sqlite3 tables for easy querying.	Because pull requests are made from external, unlisted contributors, there is an element of uncertainty in whether a pull request will be accepted. While there are certain guidelines and factors that may approximately indicate a successful pull request, the overall result is not clear and deterministic. The goal of this research is to use machine learning techniques to glean insights into what contributes to a successful pull request. If the pull request prediction accurately can be modelled, men has the e ability to concretely understand the mechanisms that motivate successful pull requests, and use this to improve the collaborative software engineering process
Geographical location	First, the authors select geographical locations, which represent at least 1% of the total pull requests on GitHub. This criterion ensures that they select diverse geographical locations with significant pull requests count for analysis. They selected United States, United Kingdom, Germany, France, Canada, Japan, Brazil, Australia, Russia, Netherlands, China, Spain, India, Switzerland, Sweden, Italy and Belgium representing approximately 83% of the contributor population for analysis. Secondly, they select 70,740 pull requests by contributors from the above-mentioned 17 locations. Lastly, they measure statistical significance at a p-value ≤ 0.05.	The paper [9] states that when contributors and integrators are from the same geographical location, there is a higher chance to get pull requests accepted compared to when the contributors and integrators are from different geographical locations.	A mixed-methods approach is used and present analyses of 70,000+ pull requests and 2,500+ survey responses.	The Analyses of 70,000+ pull requests from GitHub projects' data shows that geographical location has a significant influence on the pull request acceptance rate.

	The country which developers reside when they submit pull requests, as well as the country of the integrators. Using the combined data, the paper [20] prepares two statistical models: one focusing solely on the country of the contributor in relationship to pull request acceptance and one taking into account whether the contributor and integrator are in the same .	The outcome is evaluated by looking at the code contribution of the developer using the pull-request acceptance rate. This paper states that a developer submitting a pull-request leads to a higher acceptance rate if the evaluator of the pull-request origins from the same country.	An analysis of 70,000+ pull requests selected was presented from 17 most actively participating countries to model the relationship between the geographical location of developers and pull request acceptance decision	Countries with no apparent similarities such as Switzerland and Japan had one of the highest pull request acceptance rates, while countries like China and Germany had one of the lowest pull request acceptance rates. Notably, higher acceptance rates were observed when pull requests were evaluated by developers from the same country. The paper mirrors what was found by other papers for this factor.
Level of participation	The paper [21] looks at the Mozilla patch lifecycle to come to its conclusions.	The authors noticed, by comparing the lifecycles of core (> 100 patches) and casual (< 20 patches), that casual contributors have a lower acceptance rate and a greater rejection rate.	In the paper [21] the authors study the patch lifecycle of the Mozilla Firefox project [7]. The model of a patch lifecycle was extracted from both the qualitative evidence of the individual processes (interviews and discussions with developers), and the quantitative assessment of the Mozilla process and practice.	"Comparing the lifecycles for core vs. casual contributors, we noticed that, in general, casual contributors have 7% fewer patches that get accepted or checked into the codebase and have 6% more patches that get rejected." [21]
Expertise of the integrator	The paper [23] measures the factors by interviewing nine long-serving core developers in 5 big OSS projects. Furthermore they, manually examined hundreds of reviews across five high profile OSS projects.	The code is more likely accepted if the contribution is in the expertise of the integrator.	Using theoretical sampling to select 5 projects and then interviewing nine core developers from Apache, SVN, Linux Kernel, Free BSD and KDE, all of which were long time contributors to their respective project. [23]	"Interviewing core developers allowed us to understand why developers decide to review a particular patch. We asked, "How do you determine what to review?" All the interviewees said that they review patches that are within their area of interest and expertise that they see as important contributions." [23]
	The paper [21] looks at the Mozilla patch lifecycle to come to its conclusions.	Contributions from core developers are rejected faster. Contributions from casual developers are accepted faster.	In the paper [21] the authors study the patch lifecycle of the Mozilla Firefox project [7]. The model of a patch lifecycle was extracted from both the qualitative evidence of the individual processes (interviews and discussions with developers), and the quantitative assessment of the Mozilla process and practice.	The reason for this is most probably, as the authors state, that the contributions from casual developers tend to be smaller.
	The paper [22] performed a social network analysis of the code review data from eight popular OSS projects.	The results suggest that core developers indeed enjoy quicker first feed back intervals, shorter review intervals, and higher code acceptance rates than the peripheral developers.	The paper [22] describes that they developed the Gerrit-Miner tool, based on this paper [62], that can mine code review data in a Gerrit repository, which they used to mine the eight OSS projects.	A recommendation from the paper: "We recommend that projects allocate resources or create tool support to triage the code review requests."

Activeness of the contributors	The paper [24] gathered information from 19,543 pull requests, 206,664 comments and 4,817 commenters from 8 popular projects in GitHub.	Based on the papers [24, 17] pull requests with many associated comments are much less likely to be accepted. "Uncertain pull requests tend to require negotiation, and pull requests with lots of comments may signal controversy." [24] Activeness has a better result and precision than composite approaches. This outcome indicates that activeness is the most important of the four researched/looked at attributes, thus can be used to better incorporate the time spent per contributor .	They collected 19,543 pull requests, 206,664 comments and 4,817 commenters from 8 popular projects in GitHub and build approaches based on different attributes, including activeness, text similarity, file similarity and social relation. Also composite approaches were build, including time-based text similarity, time-based file similarity and time-based social relation.	Do note that this result is based on 8 projects and the author does not necessarily make a claim that it will hold for every other project. To the contrary.
	By sending questionnaires to understand impacts of inactive assignees.	The paper [25] states that the the pull-request has a higher change of staying open and not being accepted if the reviewer of the pull-request is a inactive developer.	The authors collect 2,374,474 records of activities in 37 popular projects, and 797,756 records of activities in 687 projects belonging to 8 organizations. They compute the percentage of inactive assignees in projects, and compare projects with and without inactive assignees. Then they analyze datasets to explore why some assignees are inactive.	"In GitHub, an issue or a pull request can be assigned to a specific assignee who is responsible for working on this issue or pull request. Due to the principle of voluntary participation, available assignees may remain inactive in projects. If assignees ever participate in projects, they are active assignees; otherwise, they are inactive yet available assignees (inactive assignees for short). Objective: The objective in this paper is to provide a comprehensive analysis of inactive yet available assignees in GitHub" [25].

Submitter's prior interaction	Lift of the rule: $first\_pull \rightarrow status\_pull = closed$. Through the analysis of the Lift, looking at the chances of rejection with the association rule $first\_pull = true$. The selection of each factor was governed by the ability to accurately calculate its values from the data (i.e., the authors [26] did not include a factor if they could not collect the corresponding data, or if a heuristic was required to compute it). The outcome is evaluated by looking at the code contribution of the developer using the pull-request acceptance rate.	The paper [26] states that a developer submitting a pull-request leads to a higher acceptance rate if the amount prior submitted pull-request of a developer increases. Submitting the first pull request is a factor that has the greatest influence in the rejection of the contribution. Also, the more prior interaction the developer has, the bigger the acceptance rate becomes. When the first contribution of a developer occurs via pull request, the chances of rejection are 3.38 times higher.	Adoption of a data mining technique, more specifically, the extraction of association rules, in order to identify new and useful patterns from pull requests data. Data mining techniques have been employed in the extraction of knowledge from software repositories [63, 64]. The exploratory analysis done through the ex-traction of association rules focuses on the discovery of intrinsic information from data set [26].	Generally, the more prior interaction the developer has, the bigger the acceptance rate becomes.
	This paper [27] states the factor selection was based on prior work in the areas of patch submission and acceptance, code reviewing, bug triaging and also on semi-structured interviews of Github developers. This factor was measure by "Number of pull requests submitted by a specific developer, prior to the examined one "	By looking at the results in paper [27] we can conclude that the previous pull requests and the acceptance rate do have a strong possitive correlation with the requester his pull request success rate, which means that if the previous pull requests on a project increases the acceptance of the pull request increases.	To understand what the underlying principles that guide pullbased development are, pullreqs was created pullreqs, a curated dataset of almost 900 projects and 350,000 pull requests, including some of the largest users of pull requests on Github.	A previous version of the dataset has been used to quantitatively study the pull request development process. The pullreqs dataset is based on previous work on GHTorrent, albeit only for its construction. While GHTorrent is a full mirror of all data offered by the Github API, the pullreqs dataset includes many features extracted by combining GHTorrent and the project's repository; the dataset is offered in a format ready to be processed by statistical software. In this paper, the construction process of the dataset and outline directions for further research with it is explained.

	This paper [17] measured prior interaction, by counting the number of events before a particular pull request that the user has participated in for this project. Events include participating in issues, pull requests, and commenting on various GitHub artifacts.	This paper [17] says "Prior interaction was also positively associated with acceptance, increasing acceptance likelihood by 35.6% per unit."	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Relation between contributors	Propensity to trust[3] In particular, the authors processed the content of the emails using Tone Analyzer and obtained an agreeableness score, which is defined within the interval [0, 1]. Values smaller than 0.5 are associated with low agreeableness and therefore, to the tendency to be less compassionate and cooperative towards others. Instead, values equal to or greater than 0.5 are in general associated with high agreeableness.	The paper [28] states that A high agreeableness/propensity to trust leads to a higher pull request acceptance rate.	The authors used the GHTorrent [5] database to retrieve the chronologically-ordered list (i.e., history) of pull requests opened on GitHub between March 2015 and December 2016. For each pull request, in particular, they stored the following information (i) the contributor, (ii) the date when it was opened, (iii) the status (i.e., merged or not), (iv) the integrator who merged it, if any, and (v) the date when it was it closed or merged. Once a mapping was obtained of the core-team members and their communication records, the propensity to trust scores from the content of the entire corpus of their emails were computed.	"The object of this study was to make a quantitative analysis on how the propensity to trust affects the success of collaborations in a distributed project, where the success is represented by pull requests whose code changes and contributions are successfully merged into the project's repository." [28]

[3]Refers to an individual's general tendency to perceive the other individuals as trustworthy, agreeableness is used as a proxy measure of the individual's propensity to trust.

	Social distance This paper [17] states that this factor was measured a dichotomous variable indicating whether or not the submitter follows the user that closes the pull request. This was used as a proxy of the social closeness between the submitter and the closer in a particular pull request.	This paper [17] says "The measure of social distance had the strongest influence on likelihood of acceptance as compared with other pull-request level factors, increasing acceptance by 187% when the submitter follows the project manager."	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Type of developer[4]	The chosen projects were initially developed by (and are maintained at) GitHub, therefore the authors could take advantage of GitHub features to understand whether a contributor is a employee or a volunteer. The paper [29] relied on GitHub $site_admin$ flag. This flag is set to GitHub users that are GitHub employees. Therefore, for any user that does not work for GitHub, this is set to false. Consequently, they used this flag to categorize employees and volunteers in the analyzed OSS projects, which are owned by GitHub.	The paper [29] states that in case of the developer being a volunteer the rejection rate of the pull request increases.	In this study, the authors characterize the sample as multiple cases of company-owned OSS. To define their sample, they searched for software companies that have made some of their software publicly available as OSS. They used the cloc4 utility to calculate the Lines of Code (LoC). It includes code from all the languages in which a project was developed, as well as blank lines and commented lines.	"In this early report, we studied three research questions: (i) Do volunteers have to try more than employees to have a patch accepted? (ii) Do volunteers have to wait much more than employees to have a patch processed? (iii) Do volunteers follow contributing best practices?" [29]

[4] Volunteer or employee

Project				
Factor	**How the factor is measured**	**Outcomes**	**How the study is conducted**	**Explanation**
Pull request system	The paper [30] compared a number of classic projects that use patch-based tools and pull-request-based tools respectively. In particular, they gather results of eight mailing-list-based projects from the published studies, and retrieve the contribution history of four GitHub projects that use pull request systems and measure their effectiveness using published metrics [30].	This paper [30] "Although there is a statistically significant improvement of accept rate in the pull request system, the practical importance (odds ratio) is modest in magnitude", this means that the acceptance rate of the pull request increases if for a certain project a pull request system is used. The acceptance rate increases of the pull request system as compared to the issue tracker (mailing-list-based). The contributions are processed faster and more frequently using pull request systems.	Eight projects using mailing lists are retrieved from previous two papers. Four of them using pull request systems from GitHub. They are Rails, jQuery, PPSSPP and Rust, representing different application domains, scale, and popularity. jQuery is a small JavaScript library mainly used for building dynamic web pages, PPSSPP is a virtual emulator of Sony Play Station Portable, and Rust is a programming language which targets at running fast, preventing crashes, and eliminating data races.	The results clarify the importance of understanding the role of tools in effective management of the broad network of potential contributors and may lead to strategies and practices making the code contribution more satisfying and efficient from both contributors' and maintainers' perspectives.
Project age and maturity	The paper [10] considered a timeline of five plus years from February, 2008 to October, 2013 with one year interval, and determine the average number of pull requests made to any single base project each month during each interval. Note that up to September, 2009, no pull request are made (i.e., not recorded in the challenge dataset), and from October, 2009 to onward, the pull requests (i.e., both successful and failed) increase almost exponentially.	The paper says that "Moreover, with the increase in forked projects, the failure rate of pull requests increases especially for the projects with more than 2,000 forks." This means that if the age and maturity of a project increases the acceptance ratio decreases.	This paper used the challenge dataset, and consider the whole sequence of conversations associated with a successful or a failed pull request as a document, and collect 9,601 conversation documents for the experiment. But they limited their study to the 78 base projects and their forked projects, and also use other 68,916 pull requests (i.e., not containing discussion) for the experiment. The details were collected of each project (e.g., domain, programming language, age and maturity) and the corresponding developers (e.g., number, experience) for the comparative analysis.	In this research, a comparative study is conducted between successful (i.e., merged with base repository) and unsuccessful (i.e., failed to merge with base repository) pull requests by analyzing different related artifacts such as the pull request discussion texts (i.e., code review comments), pull request history, and project and developer specific statistics. The study can provide important insights into the success and the failure of a pull request at GitHub repositories.

	This paper [17] measured a continuous variable representing the project's age how long a project has existed on GitHub since the time of data collection. This was used as an indicator of the repository's maturity	This paper [17] says "The older a project, used here as a proxy for maturity, the less likely it is to accept pull requests, with acceptance decreasing by 18.0% per unit of project age"	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Programming language	The paper [10] found 13 programming languages used in the 78 GitHub projects, and found nine of them having 8-10 projects each. However, they also select R language containing three projects, and discard CSS, Go and TypeScript from the experiment due to their insignificant number of projects. The average number of successful and failed pull requests for a project were considered from each of the programming languages.	The paper says that "We note that projects using Scala, C, C#, R and PHP programming languages made more successful pull requests on average than failed ones, whereas projects of JavaScript(JS), Java, Python, Ruby and C++ did the opposite" This means that languages like {Scala, C, C#, R and PHP} have an increase of the acceptance rate and languages like {JavaScript(JS), Java, Python, Ruby and C++} have a decrease of the acceptance rate	This paper used the challenge dataset, and consider the whole sequence of conversations associated with a successful or a failed pull request as a document, and collect 9,601 conversation documents for the experiment. But they limited their study to the 78 base projects and their forked projects, and also use other 68,916 pull requests (i.e., not containing discussion) for the experiment. The details were collected of each project (e.g., domain, programming language, age and maturity) and the corresponding developers (e.g., number, experience) for the comparative analysis.	In this research, a comparative study is conducted between successful (i.e., merged with base repository) and unsuccessful (i.e., failed to merge with base repository) pull requests by analyzing different related artifacts such as the pull request discussion texts (i.e., code review comments), pull request history, and project and developer specific statistics. The study can provide important insights into the success and the failure of a pull request at GitHub repositories.

	The data set used in the experiment comprises of 61,592 pull requests made on 72 different projects. The attributes considered in the analysis are: project id (project identifier); language (programming language); developer type (main team or external); first pull (informs if the pull request is the first made by the requester); commits pull (amount of commits per pull request); files added (amount of files added); file edited (amount of files edited); files removed (amount of files removed); files changed (amount of files added, moved, and edited); analysis time (time required to analyze the pull request); status pull (pull request final status). The value of the last attribute determines whether the pull request is accepted (merged) or rejected (closed).	This paper [31] says that "Pull requests written in programming languages like Java, CSS, JavaScript, and C++, have the merge chances reduced, whereas C#, C, TypeScript (few projects), Scala, and Go (few projects) increase the chances of occurrence of a merge."	This paper used an adoption of a data mining technique, more specifically, the extraction of association rules, in order to identify new and useful patterns from pull requests data. Data mining techniques have been employed in the extraction of knowledge from software repositories [63, 64]. The exploratory analysis done through the ex-traction of association rules focuses on the discovery of intrinsic information from data set.	The work of the researchers proposes an exploratory study on pull requests through data mining. More specifically, they extract association rules, aiming at discovering relationship among features of pull requests obtained from GitHub. In this section, they detail the materials and methods used in this experiment.
Number of developers	The paper [10] collected the information of 20,142 developers involved into 78 base projects, whose working experience varies from five months to 68 months.	The paper says that "however, the projects with a large developer crowd may make an excessive number of unsuccessful pull requests." This means that if the number of developers on a project increases the rejection rate probably increases. One can also derive this from the following quote: "the rate of unsuccessful pull requests increases exponentially (for one project) with more than 4000 developers involved."	This paper used the challenge dataset, and consider the whole sequence of conversations associated with a successful or a failed pull request as a document, and collect 9,601 conversation documents for the experiment. But they limited their study to the 78 base projects and their forked projects, and also use other 68,916 pull requests (i.e., not containing discussion) for the experiment. The details were collected of each project (e.g., domain, programming language, age and maturity) and the corresponding developers (e.g., number, experience) for the comparative analysis.	In this research, a comparative study is conducted between successful (i.e., merged with base repository) and unsuccessful (i.e., failed to merge with base repository) pull requests by analyzing different related artifacts such as the pull request discussion texts (i.e., code review comments), pull request history, and project and developer specific statistics. The study can provide important insights into the success and the failure of a pull request at GitHub repositories.

	First, the authors of the paper [65] extends three typical approaches used in bug triaging and code review for the new challenge of assigning reviewers to pull-requests. Second, they analyze social relations between contributors and reviewers, and propose a novel approach by mining each project's comment networks(CNs). Finally, they combine the CNs with traditional approaches, and evaluate the effectiveness of all these methods on 84 GitHub projects through both quantitative and qualitative analysis.	This paper states that the most prominent factor for the acceptance rate is the forks, stars and watches (i.e. the size of the contributors).	Evaluate the performances of the integrators on 84 projects of GitHub using precision, recall and F-Measure.	Social coding on Github, with the high volume of incoming pull-requests, poses a new challenge for integrators. With greater efficiency of the integrator's time, They can infer a higher merge rate could be achieved.
	This paper [27] states the factor selection was based on prior work in the areas of patch submission and acceptance, code reviewing, bug triaging and also on semi-structured interviews of Github developers. This factor was measure by "Number of active core team members during the last 3 months prior to creation (of the pull request)."	By looking at the results in paper [27] we can conclude that the size of the team and the acceptance rate do have a medium negative correlation with the requester his pull request success rate, which means that if the size of the team on a project increases the acceptance of the pull request decreases.	To understand what the underlying principles that guide pullbased development are, pullreqs was created pullreqs, a curated dataset of almost 900 projects and 350,000 pull requests, including some of the largest users of pull requests on Github.	A previous version of the dataset has been used to quantitatively study the pull request development process. The pullreqs dataset is based on the previous work on GHTorrent, albeit only for its construction. While GHTorrent is a full mirror of all data offered by the Github API, the pullreqs dataset includes many features extracted by combining GHTorrent and the project's repository; the dataset is offered in a format ready to be processed by statistical software. In this paper, the construction process of the data-set and outline directions for further research with it is explained.

	This paper [17] measured the number of collaborators on a project. The number of collaborators is used as a proxy for the relative size of the development team involved in a particular GitHub project.	This paper [17] says "Number of collaborators, used as a proxy for project team size, has the smallest influence on acceptance likelihood out of the three establishment measures, decreasing acceptance by 4.6% per unit of collaborator count. "	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Popularity of project	This paper [27] states the factor selection was based on prior work in the areas of patch submission and acceptance, code reviewing, bug triaging and also on semi-structured interviews of Github developers. This factor was measure by "Project watchers (stars) at creation"	By looking at the results in paper [27] we can conclude that the number of watchers and the acceptance rate do have a medium negative correlation with the requester his pull request success rate, which means that if the number of watchers on a project increases the acceptance of the pull request decreases.	To understand what the underlying principles that guide pullbased development are, pullreqs was created pullreqs, a curated dataset of almost 900 projects and 350,000 pull requests, including some of the largest users of pull requests on Github.	A previous version of the dataset has been used to quantitatively study the pull request development process. The pullreqs dataset is based on the previous work on GHTorrent, albeit only for its construction. While GHTorrent is a full mirror of all data offered by the Github API, the pullreqs dataset includes many features extracted by combining GHTorrent and the project's repository; the dataset is offered in a format ready to be processed by statistical software. In this paper, the construction process of the dataset and outline directions for further research with it is explained.
	This paper [17] measured by the number of followers a GitHub user has at time of data collection	This paper [17] says "Having followers increases the likelihood of acceptance by 18.1% per unit of followers. This suggests that submitters with higher community standing are more likely to have their pull requests accepted" This means that if the number of followers increases the acceptance rate increases.	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.

	This paper [17] measured a continuous variable for the number of stars on a project. When evaluating projects, GitHub users make use of the number of stars as a signal for community interest in the project.	This paper [17] says "Popularity had the strongest negative influence on acceptance, with projects 35.2% less likely to accept pull requests per unit of increase in stars"	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.

Process				
Factor	**How the factor is measured**	**Outcomes**	**How the study is conducted**	**Explanation**
Number of comments (on pull-request)	Outcome is evaluated by looking at code contributions of the developer using the pull-request acceptance rate. [17, 13] Calculated from a collected and preprocessed dataset of PR's [32] The paper [10] collected for each topic, the frequency of the pull requests was collected (i.e., discussion documents) and the information regarding their successes and failures. the paper[66] identifies candidate non-forked projects that received at least 1000 pull requests in total, select, from among these, the top ten projects (ranked by number of pull requests) for each programming language.	Contribution is more likely to be accepted if it has a more comments.[13, 32] Contributions with more comments are less likely to be accepted [17, 10, 66]	This paper[17] created a statistical model analyzing the association of different pull request, submitter, and repository measures of contributions with the likelihood of the contribution being accepted. This paper [32] created models that tried to capture the relationship between explanatory variables as well as conducted a survey This paper[13] cross links and analyzes eight years of patch reviews from the kernel mailing lists and committed patches from the Git repository to understand which patches are accepted and how long it takes those patches to get to the end user. This paper[10] used the challenge dataset, and consider the whole sequence of conversations associated with a successful or a failed pull request as a document, and collect 9,601 conversation documents for the experiment. But they limited their study to the 78 base projects and their forked projects, and also use other 68,916 pull requests (i.e., not containing discussion) for the experiment. The details were collected of each project (e.g., domain, programming language, age and maturity) and the corresponding developers (e.g., number, experience) for the comparative analysis. This paper[66] composes a sample of GitHub projects that make heavy use of pull requests and CI. They only consider projects using Travis-CI.	"The third and fourth most important metrics are related to patch maturity, i.e., the amount of reviewing discussion preceding a patch submission and the number of previous iterations of the patch in the same thread. Unsurprisingly, the more mature a patch, the higher the probability of acceptance.[13] In this research[10], a comparative study is conducted between successful (i.e., merged with base repository) and unsuccessful (i.e., failed to merge with base repository) pull requests by analyzing different related artifacts such as the pull request discussion texts (i.e., code review comments), pull request history, and project and developer specific statistics. The study can provide important insights into the success and the failure of a pull request at GitHub repositories.
Type of comments (on pull-request)	Comments were classified as being negative, neutral, or positive	This paper [33] states that of positive comments affects the pull request acceptance likelihood positively, whereas the percentage of negative comments affects negatively.	From a set of 24 million pull request collected, pull requests who had at least five comments were selected. They classified the comments retrieved from the Github API as having negative, neutral, or positive with their state of the art model.	-

Difficulty of git (i.e. the usage / commands)	The study[34] conducted a large-scale survey in order to investigate a large number of developers from a wide range of beginner to expert.	Developers feels GitHub and pull request more user-friendly than Git. From further examination, the inconsistency of Git commands and options and undo/redo Git commands are the influence factors of difficulties to developers. [34]	A large-scale survey in order to investigate a large number of developers from a wide range of beginner to expert. An overview of the 1,552 respondents (11.8% answer rate): Most of the respondents work for industries (74%) and have more than 6years of development experience (82%). For VCS experience,the percentage was mostly same among 1-5, 6-10 and more than 10 years. On the other hand, the majority of experience in Git and GitHub was 3-5 years (58% and 61%).	This study was only conducted using git and not other version control systems.
Conformation to the projects coding standard	This paper[35] analyzed and compared the documentation that describes their patch contribution processes to identify key process components.	A contribution is more likely to be accepted if it adheres the coding practices of the project itself [35]	The study is conducted by reviewing the processes of a wide range of OSS projects, which are among the most popular active projects from the ohloh.net website in 2010	Coding standards make it so the project does not become bloated with different coding standards, making it harder to maintain. The study focused on how to manage contribution to an open source project in order to make the project sustainable.
Reopened Pull request	This paper[36] used The Mann-Wilcoxon-Whitney Test to show differences in the evaluation time between non-reopened and reopened pull requests	This paper[36] states that reopened pull requests have lower acceptance rates, than non-reopened pull requests.[36]	Composed a dataset of pullrequest with the help of GHTorrent[5] and processed data out of that	-
Continuous Integration	This study [37] look based on the PR_merged and $CI\_Failure$ metric and looked at the acceptance rate. This study[38] ran a SQL query to find ratio of total number of pull requests, to the pull requests that were merged. They also ran SQL queries on the build history of pull request group's projects which they collected from Travis CI before. This study[39] looked at $Merged\_PR$ and $Rejected\_PR$ and see how $proj\_age$ correlates to those. This paper[40] looked at the acceptance rate of PRs	This paper[37] states that the availability of the CI pipeline may hasten the evaluation process. This paper[38] states that Pull request that had a Travis CI build had slightly higher merge rate. This paper[39] states: as observed earlier, the overall number of pull requests managed (both merged and rejected) increases after CI. This paper[40] states that PR opened with passed CI builds have 1.5 more chances of being merged than those failing. Such chances increase up to 1.72 if the build passes at the end of the PR discussion.	This study [37] composed a sample of GitHub projects that make heavy use of pull requests and CI. In this preliminary study they only consider projects using Travis-CI . The study[38] wanted to find how many of pull requests were accepted. They also want to evaluate whether other factors such as availability of automated builds influenced acceptance rates. Finally, they examined the targeted developer survey to understand how developers perceived the effectiveness of greenkeeper.io. From GHTorrent[5] dump dated 10-11-2014, they looked at projects exceeding 200 pull requests written in the most popular languages, which resulted in 1884 projects in total. [39] They analyze the process ofPR reviewing in 69 open source projects hosted on GitHub and using Travis-CI as CI infrastructure.[40]	"The general literature on CI suggests that the continuous application of quality control checks speeds up overall development, and ultimately improves software quality." This paper indicates that prediction to be valid. "To learn whether dependency management tools help developer—and if they do, to what extent—we extracted data from five sources: (i) GitHub's public dataset on Google BigQuery, which is the complete content of public repositories on GitHub; (ii) GitHub Archive dataset on Google BigQuery, which is a project that collects all the public events on GitHub; (iii) Travis-CI API; (iv) GitHub API; and (v) Git history of public projects on GitHub."

Automation	The study[41] measured by examining several metrics(including accepted and rejected PR's) across the 44 selected projects for six consecutive months before and after the first adoption of a bot	This paper [41] states they could not find any consistent statistically significant results across the analyzed projects	This study[41] selected OSS project on GitHub (excluding non-software projects, such as textbooks or bookmarks) that received at least 2.5k stars before August 2017,To identify the bot, a GitHub account was verified and its name and description were analyzed for bot references. To enable comparison, only those projects that had been active for at least six months before and six months after the bot adoption retained in the data set.	"We aimed to understand how commonly OSS projects use bots and what they use them for by manually analyzing a subset of the most starred projects from GitHub, Then we want to understand whether the acceptance rate, interaction, and decision-making time of a project change after the bot adoption. At last we are trying to understand: (1) whether stakeholders perceive the presence of bots on pull request that they submit/merge; (2) whether stakeholders agree about the relevance of bot support on software tasks; (3) problems/challenges of using bots; and (4) missing features."
Age of the pull request	This study[42] measures the factor by asking 749 integrators through an anonymized survey. This paper [67] sought to understand the students' work practices in the pull-based model and analyzed the PR data in TRUSTIE to learn the PRs' intentions and their handling time for merging. Besides, they collected the students' response in the questionnaire to understand their collaboration practices in the process.	The pull-request is less likely to be accepted the longer the age. [42] This paper [67] says "Besides, they tend to notify the group leaderafter submitting PRs, and the group leaders tend to accept PRs in ashort time span, with few code reviews." Which means that if the the age of the pull request increases the acceptance rate of the pull request decreases.	This study[42] used a large-scale two-round survey of 749 integrators, split into three logical sections: demographic information, multiple choice or Likert-scale questions, open-ended questions.	"By age of the pull-request, I mean the total minutes between open and the current time window start time."
Social and technical aspects	This study[43] measured the factor by fixation count with the use of an eye-tracker.	This study[43]. observed that both social and technical aspects are being taken into consideration when deciding upon pull request acceptance. Moreover, they observe that many more social aspects are being considered during the experiment than reported during the post-experiment survey.	This study[43] conducted an eye-tracking experiment with 42 participants to obtain a more granular understanding of which of pull request elements are considered.	The results are mainly shown through a table, thus check the differences between code signals, technical signals and social signals in it. Generally, it appears that more social aspects / signals are considered than let on in surveys. Do note that 42 participants are not that many, though.
Prioritization of work	By surveying 750 integrators through an anonymized survey. [42]	The main problems when working with pull requests is maintaining project quality and prioritization [44, 42].	A large-scale two-round survey of 750 integrators, split into three logical sections: demographic information, multiple choice or Likert-scale questions, open-ended questions.	The paper [44] mentions a potential implementation (PRioritizer) of the previous results of their paper that was conducted [42].

Code				
Factor	**How the factor is measured**	**Outcomes**	**How the study is conducted**	**Explanation**
Comments on source code	This paper [32] created models that tried to capture the relationship between explanatory variables as well as conducted a survey	This paper states that a an increase in comments in the source code increases the pull-request acceptance rate [32]	Created a statistical model analyzing the association of different pull request, submitter, and repository measures of contributions with the likelihood of the contribution being accepted	"The higher number of developers commenting on a PR leads to a lower chance being approved, while the number of developers leaving comments on the source code is likely to increase the chance of a PR being accepted and merged."[32]
Number of blank lines	Fifty factors from the related works used in this study include not only the pull request or code review on GitHub but also code quality metrics. The relationship of these factors are examined by using association rules. [48]	This paper[48] outcome is evaluated by looking at code contributions of the developer using the pull-request acceptance rate. This paper states that a an increase in blank lines on the source decreases the pull-request acceptance rate	In this study[48], open source projects on GitHub are selected since they are the most world's largest open source communities. Ansible3 is selected to be the open source project because it is the fourth on the list and it has a significant process. There are around 5.9K contributors in Ansible, which is the tenth most-discussed repository	More than 90% of all pull requests are rejected when requesting for a permission at the first time due to many reasons such as the pull requests require further information, the pull requests have an incomplete description, etc. For these reasons, the rejected pull requests are selected to analyze which aim at finding impact factors for rejection of pull requests.
Number of changed lines	This paper [49] found the factor by extracted 3000 pull requests events that were made in the window from 04/01/2012 to 04/14/2012. After data sanitization and filtering, this resulted in 2734 usable data points. Then they ran queries on GitHub Archive using Google BigQuery to get information such as the number of successful pull requests and total number of pull requests made for each contributor and repository	This paper says "Changes to code should be succinct; having too many lines changed reduces the chance of acceptance" [49], which means that an increase in changes lines of code, leads to a decrease in acceptance rate.	The papers states that the primary data source or this research was the GitHub Archive, data that is available for querying via Google's BigQuery. After that Google BigQuery was used to extract relevant data by filtering data on related event types. Once a resulting data has been generated using Google BigQuery, the data was exported over to Google Cloud Storage in Comma Separated Values (CSV) format, some preprocessing was done, and the date was imported into Sqlite3 tables for easy querying.	Because pull requests are made from external, unlisted contributors, there is an element of uncertainty in whether a pull request will be accepted. While there are certain guidelines and factors that may approximately indicate a successful pull request, the overall result is not clear and deterministic. The goal of this research is to use machine learning techniques to glean insights into what contributes to a successful pull request. If the pull request prediction accurately can be modelled, men has the e ability to concretely understand the mechanisms that motivate successful pull requests, and use this to improve the collaborative software engineering process

	This paper [50] identified ten recent pull requests interviewees had received from contributors they had not directly interacted with before. These pull request interactions varied in terms of whether or not the project owner looked at the profile of the contributor, the nature of the code being submitted, the perceived expertise of the contributor, the amount of discussion surrounding the pull request Larger changes are introducing a new feature, or conflicting with other existing functionality.	The papers says that "Owners were more certain about the value of simple changes that addressed features the owner had wanted to add, were small in scope, or fixed a known bug. Owners were less uncertain about the value of code that was suggesting a larger change, introducing a new feature, or conflicting with other existing functionality.", This means that changes that are small in scope(small number of lines) do increase the acceptance rate of a pull request and changes that are big in scope(big number of lines) do decrease the acceptance rate of a pull request.	This paper conducted interviews with 18 GitHub users focusing in detail on how they formed impressions of new people they encountered on the site. Using information obtained through the GitHub API, GitHub users who owned at least one open source project were identified. Potential interviewees who had publicly-displayed e-mail addresses available on their personal profiles were contacted to see if they would like to participate in the study.	The goal in this work was to develop a more detailed understanding of impression formation in online peer production. The researchers build on the distributed social cognition model, which posits that impression formation is an active process influenced by behavior in a network or group. This model shows that when forming impressions of others, individuals engage in an active process that involves the following steps: 1) choosing whether to obtain information about the target; 2) choosing what information is elicited, and 3) interpreting the elicited information to form a person model (an integrated interpretation of what a person is like.) The distributed social cognition model provides general guidelines about how impression formation occurs but doesn't describe what the process looks like in a specific setting.
	This paper [32] states that previous research suggests a set of different metrics that can affect code review time and outcome [68, 1]. The selection of each factor was governed by the ability to accurately calculate its values from the data (i.e., we did not include a factor if we could not collect the corresponding data, or if a heuristic was required to compute it).	This paper [32] "the PR size metric is included in the final model for review outcome (i.e., merge decision). Its negative regression coefficient indicates that larger PRs are more likely to receive a negative merge decision (i.e., a PR is not merged) than smaller ones.", which means that the number of lines of code does decrease the acceptance rate of a pull request. [32]	To understand developers' work practices and their vision of the PR review process used in the project, the researches conducted a survey. The survey contained three groups of questions: nine questions related to the demographic information about participants and their work practices, three Likert-scale questions focused on PR review, and four open-ended questions asked participants to provide more information concerning their responses to the Likert-scale questions. Participants were informed that the survey would take 10–15 minutes to complete.	The goal of the study was built around a quantitative analysis of the Active Merchant project repository, as well as an exploratory survey that we conducted with Shopify developers

	This paper [48] found fifty factors from the related works used in this study include not only the pull request or code review on GitHub but also code quality metrics. The relationship of these factors are examined by using association rules.[48]	This paper state that from the rules that they created, which present the transactions probability that having this pattern, C present the confidence value of each rule, L present the lift vale of each rule, that Rule1 - Rule7 have a confidence value equal to one, which means that if the patterns from Rule1 - Rule7 occurred in pull requests, the pull requests are definitely rejected (100%). Rule 8 - Rule10 have a confidence value equal to 0.99, which means that if the patterns from Rule8 - Rule10 occurred in pull requests, the pull requests are rejected 99[48]	The first dataset gives all labels values in "Yes" and "No" flags to identify the labels used in the pull request. If the flag is "Yes", the label is used in the pull requests. If not, it is not used in the pull request. Due to the huge amounts of "No" flag in the first dataset, some related labels are merged in the second dataset. The factors meanings are the related labels merging keys. After merging the related labels, there are only five label groups in the representative labels: community, specific needs, new submitting transaction, review type, and pull request type. r.1) code chunk -> file changed r.2) code chunk and blank line count -> file changed r.3) changes count and multi-line count -> code chunk r.4) file changes count, muti-line count and no community label -> blank line count r.5) code chunk -> blank line count r.6) file changes count, code chunk -> blank line count r.7) code chunk -> file changes count r.8) file changes count, lines of code changed count -> blank line count r.9) file changes count -> blank line count r.10) file changes count, no community -> blank line count	The main contribution of this work is to provide a method to find the impact factors on the pull requests and also the relationships among impact factors.
	The researchers of this paper devised an algorithm to analyze the portion patch acceptance concerning only source-code patches. Evaluating this proposed algorithm, they develop a method to extract patches from mailing list and Bugzilla. Mailing list is usually used for verifying patches, and Bugzilla is for tracking bugs. They are the common channels that developers discuss about patching. After they extracted all patches, the percentage of the acceptance in both fully accepted and partial accepted cases.	This paper [69] says that "Our results concluded the contradictory that the lager patches have more acceptance rate." This concludes that lager pull request are do decrease the acceptance rate.	This paper [69] decided to study on two systems, Mailing list and Bugzilla, which are totally different. There are several recently proposed methodologies for extracting patches from mailing list [70], [71]. They choose to improvise and extend the Weissgerber et al.'s proposed [72] . Which proposed method is straightforward that is the most suitable for us since improving a method to extract patches from an email is out the scope.	

	This paper [17] states that commit size was measured by the number of lines changed in the pull request.	This paper [17] says "Lines changed had a stronger effect but negative, with each unit of lines changed decreasing the chance of acceptance by 26.2%"	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Code style	This paper [52] measured the factor by a two-step approach to calculate the code style inconsistency for each PR. First, the inconsistency of code styles between each pair of the original version and the modified version of one file. The inconsistency for each pair is defined as follows: $each_{inconsistency}(s_{original}, s_{modified}) = s_{original}\ XOR\ s_{modified}$. Code style inconsistency for a PR, was only calculated by looking by considering the files that were modified by the PR. Newly added files that did not have corresponding original files were ignored during calculation.	This paper [52] found that the code style inconsistency of pull requests had a small negative effect on the decision of merging pull-requests (Acceptance ratio). This meant that a pull request with larger code style inconsistency was more likely to be rejected.	This paper [52] experimental process included four parts. First, the selected relevant projects from GitHub as the experimental subjects. Secondly, 37 code style criteria based on literature to characterize the code style of pull request were defined. Thirdly, the code style inconsistency was calculated based on the above criteria. Last, the effect of code style on integrating pull requests was explored via answering four Research Questions	In RQ1(Is there any difference between the code style of submitted PRs and that of the existing source code?), A general statistics is given on the code style inconsistency between submitted PRs and existing code. In RQ2(Which code style criteria tend to show much inconsistency among PRs?), the code style was further analyzed and answered which criteria contributed most to the above inconsistency. In the follow-up RQ3(How does code style affect the merging of PRs?) and RQ4(How does code style affect the closing time of PRs?), the regression models were leveled to measure the effect of code style inconsistency on PR integration, including the decision of merging PRs and the time cost of closing PRs

Number of tests	The feature selection was based on prior work in the areas of patch submission and acceptance, code reviewing, bug triaging and also on semistructured interviews of Github developers.	This paper [1] says "In our sample, 33% of the pull requests included modifications in test code, while 4% modified test code exclusively. Of the pull requests that included modifications to test code, 83% were merged, which is similar to the average. This seems to go against the findings by Pham et al [53]"	291 carefully selected Ruby, Python, Java and Scala projects (in total, 166,884 pull requests), and identify we studied, using qualitative and quantitative analysis, the factors that affect pull request lifetime, merging and rejection.	The study is based on data from the Github collaborative development forge, as made available through the GHTorrent project. Using it, we first explore the use of almost 2 million pull requests across all projects in Github.
	This paper [53] that in the course of the interviews, several steps of the contribution process on GitHub emerged. After receiving a pull request, the first step that project owners conducted was to manually review the contribution and assess it by different aspects. After this review, they merged the pull request into a testing branch and resolved conflicts manually. Superficial adjustments like code style corrections or comments were added based on preference	This paper [53] states that if a test suite existed, project owners ran it to check whether or not this contribution passed tests. This increased confidence in the contribution. Finally, the contribution was merged into the main branch of the project. This means that if the number of test increases the pull request acceptance rate increases.	The researched focused on understanding which testing-related norms and conventions exist on GitHub. For their investigation, they obtained 16,000 email addresses of recently active users by querying the GitHub Archive3. From this pool, 50 users were invited to semi-structured interviews and another 50 users by randomly choosing a member from each of the 50 most successful GitHub teams as listed on the Coderwall4 leaderboard. This sampling strategy resulted in a diverse population: highly experienced as well as regular users of GitHub.	This paper investigates how testing behavior is influenced by the peculiarities of social coding sites like GitHub. Different testing practices may influence software quality, maintainability, and development times. For example, in some cases testdriven development (TDD) has been shown to significantly lower the defect rate of software products. If social coding sites can influence the testing behavior of developers, they might also have an impact on the progress of software development projects and their resulting products. Understanding these influences better might enable individual developers and software development organizations to positively influence their own teams' testing practices.
	This paper [17] states that test inclusion was measured by a dichotomous variable indicating whether or not the pull request included test cases.	This paper [17] says "The inclusion of test cases was positively associated with pull request acceptance, with acceptance likelihood increased by 17.1% when tests are included"	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.

Annotations	This paper [54] found several undisciplined annotations in 110 open-source C/C++ systems of different domains, sizes, and popularity GitHub metrics. Then, manually refactored the code to make the annotations disciplined. We refactor only one annotation per project. Right away, we submit a pull request with the code changes. Also comments and questions were submitted using GitHub to understand the developers thoughts about the pull requests	This paper [54] says "Refactoring 2 was applied just in a few pull requests. When considering the "Might accept" cases as accepts, Refactoring 4 and Refactoring 6 reach 74% and 64% of acceptance rate, respectively", which means that the acceptance rate increases for disciplined code annotations.		
Contribution quality	By asking 749 integrators through an anonymized survey.[8] This paper[45] uses results from a survey to measure the contribution quality. This paper[46] used peer reviews to measure the quality. This paper[47] used machine learning techniques to classify the factor and then they calculated the accuracy measures.	The pull-request is more likely to be accepted if non-functional characteristics of the code are understandable and elegant.[8, 45, 46] However, there is also a study[47] that concludes that the quality of the code submitted in a pull request does not influence at all its acceptance.	This paper[8] used a large-scale two-round survey of 749 integrators, split into three logical sections: demographic information, multiple choice or Likert-scale questions, open-ended questions. This paper[45] used a survey among developers to determine the quality. This paper[46] used a survey among students to qualitatively determine the quality of code and used this data to label the results. This paper[47] used several machine learning techniques and then measured its accuracy among the data.	"*Integrators often relate contribution quality to the quality of the source code it contains. To evaluate source code quality, they mostly examine non-functional characteristics of the changes. Source code that is understandable and elegant, has good documentation and provides clear added value to the project with minimal impact is preferred. Apart from source code, the integrators use characteristics of the pull request as proxies to evaluate the quality of the submission. The quality (or even the existence) of the pull request documentation signifies an increased attention to detail by the submitter.*". As such it holds that: "*Top priorities for integrators when evaluating con-tribution quality include conformance to project style and architecture, source code quality and test coverage. Integrators use few quality evaluation tools other than continuous integration.*"

Functionality of the code	This study[55] measured the factor with a survey where the answers were quantitatively and qualitatively analyzed. a survey.	The paper[55] states that given the contributor is a quasi-contributor, and the functionality of the code is not needed/duplicate to other PR's, that the rejection rate of the pull request increases.	After curating the selection of OSS projects, they aimed to identify the quasi-contributors. They consider quasi-contributors those newcomers to a project who submitted pull-request(s), but had no "accepted contribution" to that specific project. They consider an accepted contribution any changes that passed the pull-request cycle and, therefore, were merged to the project code base. First, they examined the distributions of the quasi-contributions and quasi-contributors. In the second analysis, they followed open coding and axial coding procedures to qualitatively analyze open-ended questions from their surveys and pull-requests discussions. In addition, they quantitatively analyzed closed-ended questions to understand developers'perceptions about nonacceptance	"*Recent studies suggest that well-known OSS projects struggle to find the needed workforce to continue evolving—in part because external developers fail to overcome their first contribution barriers. In this paper, we investigate how and why quasi-contributors (external developers who did not succeed in getting their contributions accepted to an OSS project) fail. To achieve the goal, we collected data from 21 popular, non-trivial GitHub projects, identified quasicontributors, and analyzed their pull-requests. In addition, we conducted surveys with quasi-contributors, and projects' integrators, to understand their perceptions about nonacceptance. We found 10,099 quasi-contributors — about 70% of the total actual contributors — that submitted 12,367 nonaccepted pull-requests.*"
Complexity of the project	This paper [56] computed cyclomatic complexity by, subjecting each source code file to a commercial software code analysis tool. To account for the effects of size, the complexity metric was normalized by dividing it by the number of lines of code for each software project. This procedure also reduces collinearity problems when size is included in the regression models [20]. The Change in Cognitive Complexity (ChgCC) was calculated by subtracting cyclomatic complexity measure of the first version from the cyclomatic complexity measure of the second version, i.e., $CC_{2nd} \breve{} CC_{1st}$	This papers says "In a complex piece of code, it takes longer for a developer to determine the flow of logic resulting in slower progress of the project." This means that the if the complexity of the code rises the pull request are more likely to be open.	This paper examined OSS projects hosted at SourceForge, which was the primary hosting place for OSS projects at the time,which houses about 90% of all OSS projects. It has been argued SourceForge is the most representative of the OSS movement, in part because of its popularity and the large number of developers and projects registered [73, 74].	In this article, software complexity and its impacts are examined in the context of open source software (OSS). Past efforts have been piecemeal or based on limited information. For example, comprehension of the source code has been linked with source code complexity

PDF-XChange Editor Click to BUY NOW! www.pdf-xchange.com

Purpose of pull request	This paper [50] identified ten recent pull requests interviewees had received from contributors they had not directly interacted with before. These pull request interactions varied in terms of whether or not the project owner looked at the profile of the contributor, the nature of the code being submitted, the perceived expertise of the contributor, the amount of discussion surrounding the pull request Larger changes are introducing a new feature, or conflicting with other existing functionality.	The papers says that "Owners were more certain about the value of simple changes that addressed features the owner had wanted to add, were small in scope, or fixed a known bug. Owners were less uncertain about the value of code that was suggesting a larger change, introducing a new feature, or conflicting with other existing functionality.", This means that purposes like fixing a bug and add known features do increase the acceptance of a pull request.	This paper conducted interviews with 18 GitHub users focusing in detail on how they formed impressions of new people they encountered on the site. Using information obtained through the GitHub API, GitHub users who owned at least one open source project were identified. Potential interviewees who had publicly-displayed e-mail addresses available on their personal profiles were contacted to see if they would like to participate in the study.	The goal in this work was to develop a more detailed understanding of impression formation in online peer production. The researchers build on the distributed social cognition model, which posits that impression formation is an active process influenced by behavior in a network or group. This model shows that when forming impressions of others, individuals engage in an active process that involves the following steps: 1) choosing whether to obtain information about the target; 2) choosing what information is elicited, and 3) interpreting the elicited information to form a person model (an integrated interpretation of what a person is like.) The distributed social cognition model provides general guidelines about how impression formation occurs but doesn't describe what the process looks like in a specific setting.

	This paper [57] found that 7,529 closed pull requests in the data-set (about 10 percent) had labels associated with them. We analyzed these labels and classified the pull requests as a bug fix, feature enhancement or documentation contribution and observed the merge ratio for each of these categories.	The papers says that "Core developers seem relatively open to accept bug fixes and documentation changes from the external community, but may be apprehensive about proposed feature enhancements.", This means that purposes like fixing a bug and adding documentation do increase the acceptance of a pull request and new features do decrease the acceptance rate of a pull request.	This paper used the data set which contains, among other things, commit history and pull requests of 89 top-starred GitHub projects written in different languages and their forks. Of the 108,629 forks in this data-set, only 18,343 had at least one commit, indicating that a majority of the forks are just stubs. The root projects and forks combined account for a total of 548,299 commits by 23,237 distinct users, which are analyzed and classified, with respect to each root project, as CORE, EXTERNAL or MUTANT	In this paper, a data set of 89 popular projects hosted on GitHub and their 108,000+ forks is analyzed in order to study the levels of participation from different communities of the root projects, which are labeled as CORE, EXTERNAL or MUTANT. Note that each of these categories are defined on a per-project basis, so a CORE-committer in one project may be an EXTERNAL committer in another project. The study is formulated with the folowing 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? RQ2: Is the distribution of relative sizes of communities impacted by the main programming or scripting language used by the root project? RQ3: Are the communities that are contributing to these open-source projects geographically diverse or concentrated? 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?

Number of modified files (many files)	This paper [48] found fifty factors from the related works used in this study include not only the pull request or code review on GitHub but also code quality metrics. The relationship of these factors are examined by using association rules.[48]	This paper state that from the rules that they created, which present the transactions probability that having this pattern, C present the confidence value of each rule, L present the lift vale of each rule, that Rule1 - Rule7 have a confidence value equal to one, which means that if the patterns from Rule1 - Rule7 occurred in pull requests, the pull requests are definitely rejected (100%). Rule 8 - Rule10 have a confidence value equal to 0.99, which means that if the patterns from Rule8 - Rule10 occurred in pull requests, the pull requests are rejected 99[48]	The first dataset gives all labels values in "Yes" and "No" flags to identify the labels used in the pull request. If the flag is "Yes", the label is used in the pull requests. If not, it is not used in the pull request. Due to the huge amounts of "No" flag in the first dataset, some related labels are merged in the second dataset. The factors meanings are the related labels merging keys. After merging the related labels, there are only five label groups in the representative labels: community, specific needs, new submitting transaction, review type, and pull request type. r.1) code chunk -> file changed r.2) code chunk and blank line count -> file changed r.3) changes count and multi-line count -> code chunk r.4) file changes count, muti-line count and no community label -> blank line count r.5) code chunk -> blank line count r.6) file changes count, code chunk -> blank line count r.7) code chunk -> file changes count r.8) file changes count, lines of code changed count -> blank line count r.9) file changes count -> blank line count r.10) file changes count, no community -> blank line count	The main contribution of this work is to provide a method to find the impact factors on the pull requests and also the relationships among impact factors.

	The data set used in the experiment comprises of 61,592 pull requests made on 72 different projects. The attributes considered in the analysis are: project id (project identifier); language (programming language); developer type (main team or external); first pull (informs if the pull request is the first made by the requester); commits pull (amount of commits per pull request); files added (amount of files added); file edited (amount of files edited); files removed (amount of files removed); files changed (amount of files added, moved, and edited); analysis time (time required to analyze the pull request); status pull (pull request final status). The value of the last attribute determines whether the pull request is accepted (merged) or rejected (closed).	This paper [31] says that "Rule 3 (commits pull = 1 commit $\wedge$ files added = no file $\wedge$ developer type = main team $\wedge$first pull = false) displays a conjunction of good factors with positive influence over the acceptance of a pull request. It is noteworthy that a request with a sole commit, that does not add new files, made by a member of the main team which had already made a pull request before, has the chance of merge increased in 40%."	This paper used an adoption of a data mining technique, more specifically, the extraction of association rules, in order to identify new and useful patterns from pull requests data. Data mining techniques have been employed in the extraction of knowledge from software repositories [63, 64]. The exploratory analysis done through the ex-traction of association rules focuses on the discovery of intrinsic information from data set.	The work of the researchers proposes an exploratory study on pull requests through data mining. More specifically, they extract association rules, aiming at discovering relationship among features of pull requests obtained from GitHub. In this section, they detail the materials and methods used in the experiment.

	To characterize patches and build prediction models, the researchers did extract features from patches and considered three possible feature sources: patch meta-data, patch content, and bug report information. Patch meta-data: Patch meta-data includes patch information such as patch writer, patch file names, and the number of patch files. Patch content: The quality of patch content plays a major role in deciding the review outcome. Correct, good, and simple patches will be accepted. However, extracting patch quality as features is a challenging task. Bug report information: The bug reports and patches included in the bug reports are closely related. For example, if a bug in a report is not a real bug or a trivial feature enhancement request, then patches of the bug may not be interested either.	This paper [58] states if the that the number of files changed in creases the rejectance rate of a single path decreases.	For this paper, the review processes of two open-source projects were observed, Firefox and Mozilla Core. It is noticed that code reviews are mostly organized manually. In particular, finding appropriate reviewers is a complex and time-consuming task and, surprisingly, impacts the review outcome: review requests without an initial reviewer assignment have lower chances to be accepted (and take longer).	
	This paper [27] states the factor selection was based on prior work in the areas of patch submission and acceptance, code reviewing, bug triaging and also on semi-structured interviews of Github developers.	By looking at the results in paper [27] we can conclude that the number of modified files and the acceptance rate do have a weak positive correlation with the requester his pull request success rate, which means that if the number of modified files of the pull project increases the acceptance of the pull request increases.	To understand what the underlying principles that guide pullbased development are, pullreqs was created pullreqs, a curated dataset of almost 900 projects and 350,000 pull requests, including some of the largest users of pull requests on Github.	A previous version of the dataset has been used to quantitatively study the pull request development process. The pullreqs dataset is based on previous work on GHTorrent, albeit only for its construction. While GHTorrent is a full mirror of all data offered by the Github API, the pullreqs dataset includes many features extracted by combining GHTorrent and the project's repository; the dataset is offered in a format ready to be processed by statistical software. In this paper, the construction process of the dataset and outline directions for further research with it is explained.

	This paper [17] states that this factor was measured by looking at the number of files changed in the pull request	This paper [17] says "Lines changed had a stronger effect but negative, with each unit of lines changed decreasing the chance of acceptance by 26.2% compared to 7.3% with each unit of files changed." This means that is the number of changed files increases the acceptance rate decreases.	A dataset was created of pull requests and the users and repositories associated with each pull request through sampling for active, collaborative projects on GitHub. The dataset comprises information gathered from the GitHub Application Programmer Interface (API). In total, this includes 659,501 pull requests across the 12,482 projects. For this dataset, information was gathered about each unique GitHub user associated with the set of pull requests. This set of user information includes 95,270 unique GitHub user accounts. The API was also used to gather information on all issues and comments for each repository	This paper present a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests, which are one of the primary methods for contributing code in GitHub. The association of various technical and social measures is analyzed with the likelihood of contribution acceptance.
Number of commits	This paper [49] found the factor by extracted 3000 pull requests events that were made in the window from 04/01/2012 to 04/14/2012. After data sanitization and filtering, this resulted in 2734 usable data points. Then they ran queries on GitHub Archive using Google BigQuery to get information such as the number of successful pull requests and total number of pull requests made for each contributor and repository	This paper says "Multiple commits imply a complex change and reduces the chance of acceptance" [49], which means that an increase in commits, leads to a decrease in acceptance rate.	The papers states that the primary data source or this research was the GitHub Archive, data that is available for querying via Google's BigQuery. After that Google BigQuery was used to extract relevant data by filtering data on related event types. Once a resulting data has been generated using Google BigQuery, the data was exported over to Google Cloud Storage in Comma Separated Values (CSV) format, some preprocessing was done, and the date was imported into Sqlite3 tables for easy querying.	Because pull requests are made from external, unlisted contributors, there is an element of uncertainty in whether a pull request will be accepted. While there are certain guidelines and factors that may approximately indicate a successful pull request, the overall result is not clear and deterministic. The goal of this research is to use machine learning techniques to glean insights into what contributes to a successful pull request. If the pull request prediction accurately can be modelled, men has the e ability to concretely understand the mechanisms that motivate successful pull requests, and use this to improve the collaborative software engineering process
	This paper [27] states the factor selection was based on prior work in the areas of patch submission and acceptance, code reviewing, bug triaging and also on semi-structured interviews of Github developers.	By looking at the results in paper [27] we can conclude that the number of commits and the acceptance rate do have a weak positive correlation with the requester his pull request success rate, which means that if the number of commits of the pull project increases the acceptance of the pull request increases.	To understand what the underlying principles that guide pullbased development are, pullreqs was created pullreqs, a curated dataset of almost 900 projects and 350,000 pull requests, including some of the largest users of pull requests on Github.	A previous version of the dataset has been used to quantitatively study the pull request development process. The pullreqs dataset is based on the previous work on GHTorrent, albeit only for its construction. While GHTorrent is a full mirror of all data offered by the Github API, the pullreqs dataset includes many features extracted by combining GHTorrent and the project's repository; the dataset is offered in a format ready to be processed by statistical software. In this paper, the construction process of the dataset and outline directions for further research with it is explained.

	The factor is measured by looking at code contributions of the developer using the pull-request acceptance rate. [26] This paper [75] uses the GHTorrent[5] and datamining to look at the factor. This paper used manual inspection and measured the amount of commits versus the acceptance rate [76]	The paper states that an increase in commits in the pull request reduces the pull-request acceptance rate [26] The paper[75] states that the acceptance rate went down within previous studies and papers with the size of the pull request (number of commits/files etc.). With this paper, the addition gets added that those factors also may influence assignment of the reviewers. This paper [76] says that if the size of the request is big, it has less chance to be accepted.	Adoption of a data mining technique, more specifically, the extraction of association rules, in order to identify new and useful patterns from pull requests data. Data mining techniques have been employed in the extraction of knowledge from software repositories [63, 64]. The exploratory analysis done through the ex-traction of association rules focuses on the discovery of intrinsic information from data set. derive a comprehensive list of patch rejection reasons from a manual inspection of 300 rejected Eclipse and Mozilla patches, a large-scale online survey of Eclipse and Mozilla developers.[76] In this work, we adopted a data mining technique called association rules extraction. Specifically, the study employed the Knowledge Discovery in Databases (KDD) process, as follows: (1) data selection; (2) pre-processing; (3) transformation and data enrichment; (4) association rules extraction; and (5) results interpretation and evaluation. GHTorrent was used for data selection. [26]	"*In pull requests with several commits* $(commits\_pull = manycommits)$*, the chances of rejection increase in* 51%*. Figure 2 shows the Lift values for association rules of type* $commits\_pull^3 \rightarrow status\_pulls = closed$*.Thus, we can conclude that the greater the number of commits performed in a pull request, the higher is the chance of rejection.*" "Factors such as the number of commits and files in the pull request may influence the reviewer assignment. For example, some reviewers have preference for smaller pull requests, with few commits and files, with confidence ranging from 82% to 100%." reviewers reject a large patch not solely because of its size, but mainly because of the underlying reasons that induce its large size, such as the involvement of unnecessary changes