Discovering the metrics for assessing a project's maturity: An analysis of key indicators of maturity

Kendra Sartori

Supervisor(s): Sebastian Proksch, Shujun Huang

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Kendra Sartori
Final project course: CSE3000 Research Project
Thesis committee: Sebastian Proksch, Shujun Huang, Fenia Aivaloglou

Abstract

Continuous integration (CI) is a software engineering practice that promotes frequent code integration into a shared repository, improving the productivity within development teams as well as the quality of the software being developed. While CI adoption has gained traction, studies have examined its effective implementation and associated challenges. The idea that multiple contextual factors influence the adoption of CI prompts an exploration of suitable descriptive metrics for describing the CI practices employed. This paper aims to explore the metrics that best depict the level of maturity of a project, addressing the question: "What metrics can be used to describe the maturity level of a project?". With a lack of a comprehensive maturity framework, we leverage GitHub's API in an attempt to analyze various metrics to be used to create a framework for filtering projects.

Our findings indicate that project maturity cannot be captured by a single metric, but rather a combination of metrics reflecting different aspects throughout the project's lifecycle. Activity levels, including commits and pull requests, popularity indicators like stargazers, forks, and contributors, as well as repository size and age, emerge as primary indicators of maturity. By combining these metrics, a unified framework for categorizing mature projects can be established and further developed.

1 Introduction

Continuous Integration (CI) is a software engineering practice that enables developers to integrate their work into a shared repository while consistently reviewing each other's code, leading to high qualitative results [1]. CI has gained significant popularity among software development teams and has attracted the interest of researchers investigating its advantages. Previous studies indicate that CI can, for example, enhance software quality by detecting errors at an earlier stage in the software development process [2], as well as the level of productivity within development teams [3].

In the past, studies have explored methods to improve the adoption and implementation methods of CI from multiple points of view, for example researching methods to reduce the associated costs [4]. On the other hand, through research, different anti-patterns that can determine the benefits associated with CI have been described [5], with studies highlighting the existence of uncertainties and drawbacks surrounding the adoption and implementation of CI within development teams [2]. In this sense, Elzhary et al. [2] suggest that the adoption and implementation of CI practices are influenced by the specific context in which they are employed. To that extent, the idea that there may be multiple factors influencing CI implementation is presented.

The primary objective of this study is to examine which descriptive metrics within a project have an impact on the adoption of CI and how these metrics can be utilized to establish a framework for enhancing the CI implementation process. This gives rise to a complicated problem that requires a multi-faceted approach and a clear delineation of the different components that it is composed of. Thus, through this study, we will investigate the different metrics that describe a project's activity, maturity and context, as well as the way that project is performed in the build life cycle and the way that the CI pipeline is implemented.

Narrowing down the research to one of these dimensions, this paper focuses on the study of project maturity and aims to investigate which project metrics can be used to describe and classify projects as mature. The goal is to develop a guideline or framework that assists in determining the maturity level of projects with the help of the identified metrics.

Project maturity is an important aspect in project management as it indicates the level of development of a project as well as the quality of the work being carried out. In fact, project maturity is considered to be closely correlated to the notion of release-readiness, constituting an indicator of whether a project has reached a stage where it can be delivered to a stakeholder [6].

Realising the importance of this notion in the realm of software development, the main research question that will be answered through this research is: **"What metrics can be used to describe the maturity level of a project?"**. In order to answer the main question, the different aspects to analyse have been divided into several sub-questions:

RQ1: To what degree does a project's activity reflect its level of maturity?

RQ2: To what extent is the popularity and community involvement of a project descriptive of its level of maturity?

RQ3: What additional metrics can be used to categorize a project as mature?

Following a concise overview of related work in this domain, the remainder of this paper focuses on the methodology employed for extracting projects and relevant metrics, as well as clustering projects into two groups: mature and immature projects. Subsequently, we outline the meaningful observations we were able to make by comparatively analysing relevant metrics. Final sections outline practices of responsible research regarded in our work as well as further work that can be carried out and conclusions that have been reached in this paper.

2 Related work

To establish a strong groundwork for investigating metrics that can effectively describe the maturity level of a project, it is important to conduct a comprehensive analysis of prior research in this field. This analysis serves two purposes. Firstly, it allows for a deeper comprehension of the various perspectives on project maturity. Secondly, the literature review aids the consolidation of a list of relevant metrics that we can further investigate.

2.1 Maturity models

A considerable body of research has investigated the concept of project maturity and its description in the context of project management. In this sense, a large body of maturity models have been developed which describe frameworks for assessing the level of maturity of the process employed by a company while developing a new product. One of the most widely recognized such maturity models is the Capability Maturity Model (CMM). The CMM is a framework that provides a structured approach for assessing and improving the maturity of an organization's software development processes. It offers a set of recommended practices and process areas that have been proven to enhance software development capability and quality [7]. There are also other maturity models such as the Capability Maturity Model Integration (CMMI) which extends upon the CMM, or other models that also relate to process maturity such as the Open Source Usability Maturity Model (OS-UMM).

However, it is important to mention that while these maturity models have proven to be very useful to assess the level of maturity of the processes employed within companies, they offer a more generalised framework for assessing maturity. These models offer an overview of the maturity from multiple points of view, with lesser focus on the implementation aspect of a project and more focus on the processes followed within teams.

2.2 Descriptive metrics in literature

Although the concept of maturity at project level can oftentimes be considered common knowledge among developers, the literature review has revealed that there is no general common perception about what makes a project mature. Thus, it comes as no surprise that a unified framework for classifying projects as being mature has yet to be developed.

The filtering method developed by Gallaba and McIntosh [8] emerged as the closest approximation to a framework for classifying mature projects. To ensure the integrity of their findings, the researchers focused their analysis solely on projects that had achieved a specific level of maturity, thereby mitigating the potential impact of immature projects on their conclusions. To refine their project selection, the researchers have developed a filtering method consisting of four criteria, applied sequentially on a large set of repositories, in order to filter out only the projects of interest. Their filtering process involves identifying active projects based on commit activity and size of a repository, selecting projects that use Travis CI, excluding forked projects, and filtering out duplicates to minimize bias in the analysis. In Section 3, we will describe how this filtering method will be adopted and customized in this study in an attempt to cluster our dataset into mature and immature projects.

While no research with the same goal as ours has been found through literature review, in addition to Gallaba and McIntosh's [8] filtering method, numerous research articles mention project maturity as a criterion for filtering projects, highlighting metrics that they consider indicative of maturity. By scrutinizing these previous studies, a set of appropriate metrics to analyze can be compiled, which combined and verified could help set the foundations of a framework for classifying projects.

Previous studies have outlined different aspects of the commits within a repository as being descriptive of a projects level of maturity. Brindescu et al. [9], have indicated that commits tend to decrease in size over time, with a shift towards small fixes rather than the implementation of new features. In line with this finding, our research considers commit churn, which quantifies the ratio of lines added to lines deleted within a commit, as a metric to assess project maturity [10]. Additionally, the number of non-merged commits is surfaces as an indicator of maturity, as it is considered to have a tendency to decrease as projects age [10].

Another important category to consider for study is the metrics associated with pull requests. These metrics have been observed to hold significance in analyzing project maturity. Merges are observed to occur more rapidly [11], suggesting that these tend to involve smaller and easier-to-review changes. Furthermore, as mentioned earlier, the observation that commit size tends to decrease in more mature projects [9] can also impact the size of pull requests, in the number of additions, deletions and files changed.

Several studies employ popularity and community engagement when talking about a project's maturity [12], [10]. While quantifying popularity and community engagement may pose challenges, as they are not the primary focus of our paper, we will consider the metrics discussed in the literature as reliable indicators of these factors. Previous studies have relied on the number of stars [13] and the number of forks [14] to assess a project's reputation and popularity. Others, consider the growing number of pull requests received by a repository as indicative of the growing interest shown in a project [10]. Furthermore, the number of contributors associated with a repository is considered an indicator of its maturity as more contributors typically suggest a higher level of community involvement and stability [10].

Nery et al. [15] have employed the age of a repository as a criterion for categorizing projects as mature. Similarly, Ghaleb et al. [14] have identified several metrics to differentiate between mature and immature projects, mentioning age, source lines of code, and test density as particularly indicative of maturity.

Jin and Servant [16] consider that pojects with larger test cases and more source code lines or projects with longer usage of CI tend to have a higher rate of first build failures. They attribute this finding to the idea that as projects age and become more mature, they either demonstrate an improvement in bug-detection capabilities or more bugs affect their builds. Continuing the discussion on builds, Durieux et al. [13] suggest that the number of restarted builds is directly proportional to the complexity and maturity level of a project.

Table 1 contains a condensed list of metrics that have been identified as relevant in the existing literature. These metrics will form the theoretical basis for the research conducted in our study in an attempt to gain a deeper understanding between these and project maturity, providing insights for the development of a comprehensive framework for assessing and categorizing projects based on their maturity level.

3 Methodology

In this section we analyze the methodology employed for conducting our research, which was structured into two important parts. Firstly, we conducted data extraction to ob-

Table 1: Metrics used for analysis of project maturity

Category	Metric	Description
Activity level metrics	Commit size	Number of changes in a commit calculated as the commit churn (# additions + # deletions)
	# non-merge commits	Number of commits that were not merged into the default branch
	Merge speed	Time interval between the time a pull request was opened and the time a pull request was closed
	Pull request size	Number of changes introduced by a pull request in terms of code churn and # files changed
Popularity & community engagement metrics	# stargazers	Number of stars a GitHub repository has received
	# forks	Number of repository forks for a project
	# contributors	Number of unique GitHub users that have modified code in a repository
	Pull request frequency	The frequency at which new pull requests are opened
Other metrics	Project age	The difference in time between the present moment and the time the repository was created
	Project size (SLOC)	Number of source lines of code of a project
	Team size	The size of the team working on a project
	Test density	The number of test cases per 1000 source lines of code
	# first build failures	Number of builds that fail on their first run
	Frequency of restarted builds	Number of builds that are restarted after a failure

tain a comprehensive dataset. Secondly, the filtering method ensured the creation of two clusters: mature and immature projects. This allowed us to conduct a comparative analysis and identify the metrics that are indicative of project maturity.

3.1 Data extraction

The rise of GitHub[1], renowned for its ”social coding” features that support collaboration and sharing, has propelled collaborative software development to unprecedented levels [17]. As the largest code host in the open source community, GitHub has become a central hub for developers worldwide [17]. This remarkable growth and influence serve as the primary motivation for selecting GitHub as the platform for extracting repositories to be analyzed in this study. With that in mind, we leveraged the functionalities offered by PyGitHub[2], a Python library which facilitates the access to the GitHub API[3] [18].

3.1.1 Repository extraction

In order to collect a dataset for analysis, projects were extracted from GitHub using the search API provided in PyGithub. This API allows programmatic access to the GitHub search functionality, enabling us to retrieve repositories based on specific criteria.

To ensure the relevance and quality of the extracted projects, the following filtering criteria were applied:

- **Public Repositories**: Only repositories that were publicly available on GitHub were included in the dataset. This criterion was implemented to ensure that the extracted projects were accessible and transparent.
- **Non-Templates**: GitHub provides a feature to create template repositories, which serve as a starting point for similar projects. To focus on analyzing original projects, template repositories were excluded from the dataset.
- **Non-Forks**: A fork represents a repository that enherits its code and a number of settings from another, ”upstream” repository [19]. To ensure that the analyzed projects were not derivatives of other projects, we excluded forked repositories from our dataset.
- **Older than 1 year**: To ensure data adequacy for analysis, only repositories in existence for over a year were included. The purpose of this condition was to increase the likelihood of obtaining meaningful results by excluding repositories that were in their early stages of development.

After a first iteration of the repository retrieval with the aforementioned query parameters, it was observed that the search API's default retrieval order was based on the number of stars. This limitation seemed to impact certain metrics that are studied in our research and thus compromise the quality of the data. In order to avoid extracting only the repositories with highest numbers of stars while still satisfying the other query criteria, the project extraction process was divided into batches. A query parameter indicating the number of stars being in a certain interval was added for each batch. After retrieving 50 batches of 20 repositories each, 10 repositories were randomly selected from each batch, leaving us with 500 repositories to further analyse. The choice of narrowing down the list to 500 repositories was made in the interest of time allocated for conducting our study.

By utilizing the search API and applying the aforementioned criteria, a diverse dataset of 500 non-forked, non-template, and public repositories with different levels of star count was obtained for further analysis.

[1]https://github.com/

[2]https://pygithub.readthedocs.io/en/latest/introduction.html

[3]https://docs.github.com/en/rest?apiVersion=2022-11-28

3.1.2 Metrics extraction

The next step in our research process, following the extraction of the repositories we want to study, is to extract relevant metrics that describe these repositories. Following the literature review we were able to compile a list of metrics that we would like to study, as described in Section 2.2, and split them into three categories, each of which aids to support answering one of the research questions.

Some of these metrics are available to us through the use of GitHub API, while for some of the metrics we need to compute additional information. For instance, for the weekly aggregates of additions and deletions, we include an array that incorporates the churn metric, calculated as the sum of additions and deletions [10]. Listing 1 outlines the extracted metrics.

Listing 1: Repository metrics extracted

```
repository_name: {
    "created_at": Timestamp,
    "forks_count": Integer,
    "stargazers_count": Integer,
    "watchers_count": Integer,
    "open_issues_count": Integer,
    "contributors_count": Integer,
    "commits_count": Integer,
    "open_pull_requests_count": Integer,
    "all_pull_requests_count": Integer,
    "weekly_code_frequency": Integer Array,
    "weekly_code_additions": Integer Array,
    "weekly_code_deletions": Integer Array,
    "weekly_commit_count_last_year": Integer Array
}
```

In addition to the metrics that describe a repository, we also extract the last six months of pull requests for each repository. This data allows us to formulate part of the answers for the first two research questions, testing multiple assumptions such as the idea that the speed at which merges occur increases as projects mature [10] or the fact that a growing number of pull requests might be an indicator of growing popularity in mature projects [11]. Listing 2 outlines the metrics extracted for a pull request.

Listing 2: Pull request metrics extracted

```
{   "title": String,
    "additions": Integer,
    "deletions": Integer,
    "churn": Integer,
    "changed_files_count": Integer,
    "commit_count": Integer,
    "created_at": Timestamp,
    "merged": Boolean,
    "closed_at": Timestamp,
    "merged_at": Timestamp
}
```

By extending our analysis using these metrics, we seek to gain insights and validate the findings documented in previous research, in order to uncover the metrics that are descriptive of maturity.

3.2 Project filtering

After the initial extraction of repositories and metrics, the next step involved filtering the projects and categorizing them into two groups: mature and immature projects. Given that Gallaba and McIntosh's filtering process [8] represents the closest approximation to a viable framework for identifying mature projects, as described in Section 2.2, we adopted their work as a foundational reference. Their method serves as a basis for the methodology employed to filter mature repositories from GitHub, with the necessary adjustments to fit this research and the time frame that it takes place in. The following three are the filters that we applied in order to divide our dataset into two clusters:

CI implementation filter

The first filter focused on identifying projects that implemented CI practices aligning with the research's primary objective of studying the broader implementation and adoption of CI practices. Unlike studies that focus solely on a specific CI tool, such as the work conducted by Gallaba and McIntosh [8], our research considers a comprehensive range of widely used CI tools, including but not limited to GitHub Actions, Travis CI, and Jenkins. In order to filter the projects, we searched the repositories for directories of files that had names containing common CI tool file names such as ".travis.yml" or ".circleci". This approach allowed for the detection of potential indications of CI implementation based on the file naming conventions. If such files or directories were found, the repository was considered to have CI implemented.

Repository size filter

The second filter aimed to identify projects with a substantial number of files. According to Gallaba and McIntosh's analysis [8], a number of 500 files or higher is indicative of mature projects. Thus a threshold of 500 files was used for this filter. In order to accomplish this, the total number of files in a repository was recursively counted and tested against the threshold of 500 files.

Development stage filter

Gallaba and McIntosh [8] created a commit-based filtering approach for the activity of projects. However, we found this filtering method to be ineffective when we applied it to categorize our own set of projects and thoroughly examined the resulting clusters. We attributed the limitations of the initial activity filter to the fact that in our study, we extract data through GitHub API, whereas their research utilized the public GitHub dataset on Google BigQuery.

With that in mind, we decided to look at the activity of a project from the point of view of the weekly code churn in an attempt to identify patterns in the activity levels of projects, and find a delineation between implementation stages and maintenance stages. This idea was based on Zhao et al.'s [10] observation that as projects mature they tend to reach a more stationary phase, with code changes becoming smaller as no new features are being implemented.

To shape our final filter, we have leveraged the code churns extracted and computed as described in Section 3.1.2 and performed the following steps:

1. The code frequencies have been divided into intervals of 26 weeks, which corresponds approximately to a 6-

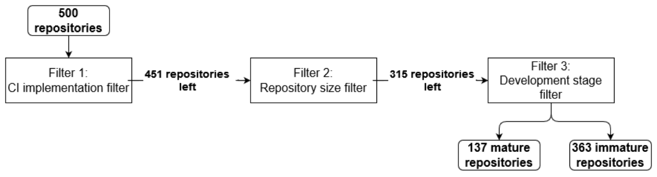
500 repositories
Filter 1: CI implementation filter
451 repositories left
Filter 2: Repository size filter
315 repositories left
Filter 3: Development stage filter
137 mature repositories
363 immature repositories

Figure 1: Results of sequentially applying filters to the repositories

month period, since the code frequencies are recorded on a weekly basis.

2. Subsequently, a dynamic threshold for each repository was calculated by multiplying the average code frequency with a ratio of 0.8, determined through multiple experiments.
3. Within each interval, we examined whether the code frequencies exceeded the dynamic threshold in at least 80% of the weeks, and ensured that there were no weeks without any code changes.
4. If two intervals exhibited signs of constant implementation and were not the final interval, it was considered that the code frequency indicated a mature project. The exclusion of the last interval aimed to classify only those projects in a maintenance stage as mature.
5. The final check involved observing the previous 52 weeks, equivalent to a year, and verifying that the repository had updates in at least one quarter of that period. This check aimed to identify if the projects were still being actively maintained.

By applying these filters, our dataset was divided into two distinct clusters, enabling us to conduct a focused analysis of relevant metrics and draw insightful conclusions regarding the indicators of project maturity. These filters facilitated the identification of repositories exhibiting signs of maturity. After the filtering process, 137 projects were considered mature, while the remaining 363 were labelled as being immature.

4 Experimental Setup and Results

Once all the necessary data was collected for our investigation, we proceeded with an analysis phase to examine various repositories and their associated metrics. The primary objective of our research is to investigate these metrics and their effectiveness in evaluating project maturity. Moreover, our dataset is divided into two distinct categories: mature projects and immature projects. This division allows us to perform a comparative analysis between these categories, enabling us to determine which metrics are more descriptive of either category. By conducting this comparative analysis, we aim to gain insights into the unique characteristics and factors that contribute to the maturity levels of different projects.

4.1 RQ1: To what degree does a project's activity reflect its level of maturity?

Commit size Our first research question aims to investigate whether the activity level in a project can serve as an indicator of its maturity level. To do this, we will first analyze the

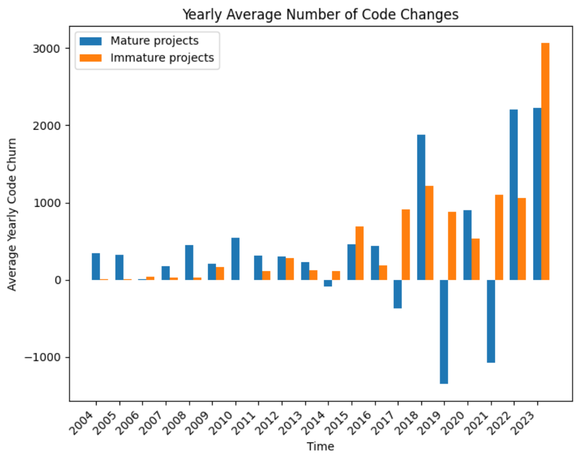

Figure 2: Yearly code churn average

trends in commit size. As a first step, we calculated the code churn by summing the weekly additions and deletions in a repository. Subsequently, we determined the yearly average of code churn and conducted a comparative analysis of the average code churns between mature and immature projects.

In Figure 2, we present the trend of commit sizes over the years, represented by the average code churn, calculated as described in Section 3.1.2. Notably, we observe that as time progresses, both mature and immature projects experience an increase in the number of code modifications. Interestingly, mature projects tend to exhibit negative churns more frequently compared to immature projects. This observation reinforces the idea that as projects mature, the focus shifts towards code fixes rather than the implementation of new features [10].

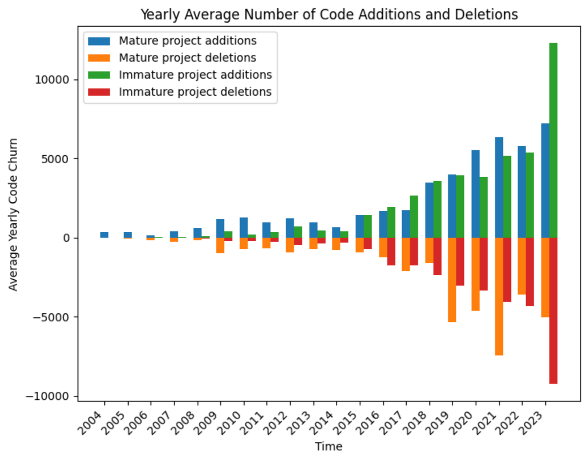

Figure 3: Yearly average additions and deletions

Additionally, by examining the data presented in Figure 3, we can gain a clearer understanding of the average number of deletions and additions. It is evident that both mature and immature projects show an increasing trend in the num-

Table 2: Pull Request characteristics for Mature and Immature Projects

Maturity	Avg. churn	PRs with code churn = 0	PRs with churn > avg.	PRs with < 1 commit	PRs with < 1 file changed	PRs merged	PRs closed w/o merging	PRs open	# PRs
Mature	628.72	0.25	0.04	0.4	0.54	0.72	0.16	0.09	42714
Immature	442.55	0.23	0.06	0.38	0.54	0.74	0.22	0.12	61742

ber of changes. However, the rate of growth in changes is more pronounced for immature projects, slowly surpassing the number of changes in mature projects as time progresses. This observation might imply that the immature projects we have selected are currently in the implementation stage, while the projects classified as mature have reached a certain level of maturity during the more recent years. This trend is also evident in both figures, where the number of code changes begins to stabilize and remains relatively constant for mature projects.

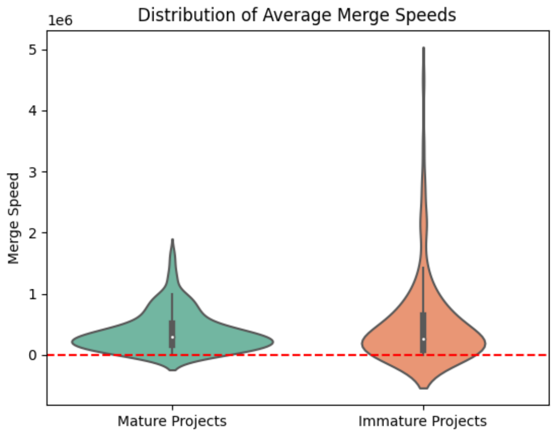

Figure 4: Average Pull request merge speed comparison

Pull request merge speed In our study, we have found that previous research conducted, highlighted a correlation between pull request (PR) trends and project maturity. The hypothesis that merges happen more quickly in projects that are more mature [11] was tested by examining the time it took for all PRs opened in the last six months to be merged after they were opened for our selected repositories. We then calculated the average speed of merging for each repository and categorized this data based on two predefined clusters. The results of the comparative analysis can be seen in Figure 4. Overall, it is evident that merges tend to occur more rapidly in mature projects, with an average time of 6286.80 seconds, compared to immature projects, where the average time before merging is 8985.75 seconds.

Pull request size On the same consideration that commit sizes decrease as projects age [9], we tested the hypothesis that PRs might also shrink in size as projects grow more mature. Figure 5 illustrates the average sizes of PRs, expressed in terms of code churn, in both mature and immature projects over the past six months. It is straightforward to observe that the PR sizes in terms of code churn tend to be slightly larger for immature projects, with a few minor exceptions. Additional information about PR sizes is highlighted in Table 2, for a more comprehensive comparative analysis.

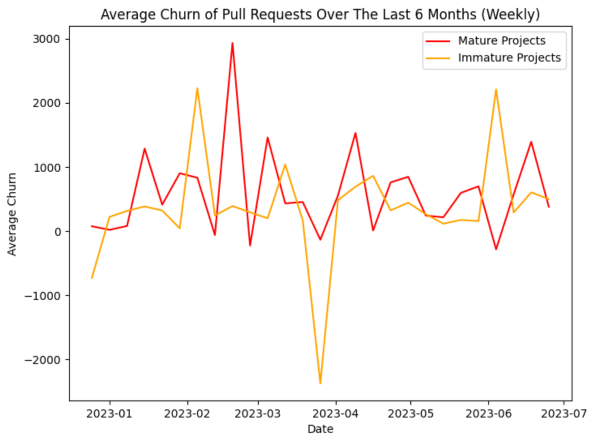

Figure 5: Pull Request churn over the past 6 months

4.2 RQ2: To what extent is the popularity and community involvement of a project descriptive of its level of maturity?

Number of stars, forks and contributors We begin our study of the impact of the popularity and community engagement of a repository on its level of maturity by studying the number of stars, forks and contributors for the 500 repositories we selected. In Figure 6 we can observe the disparity in the average counts of these metrics between mature and immature projects. Specifically, mature projects exhibit an average of approximately 16,950 stars, 4,500 forks, and 300 contributors, whereas immature projects show an average of 15,100 stars, 2,600 forks, and 160 contributors. The fact that immature projects have a significantly lower number of contributors and forks compared to mature projects enforces the idea that these metrics might be descriptive of a project's maturity. While the number of stars can also be viewed as an indicator of a project's maturity based on our results, it should be approached cautiously. It tends to be higher in mature projects, but the default retrieval ordering of GitHub's API could introduce a potential bias in the count of stars, regardless of the strategy we used to try to mitigate this limitation.

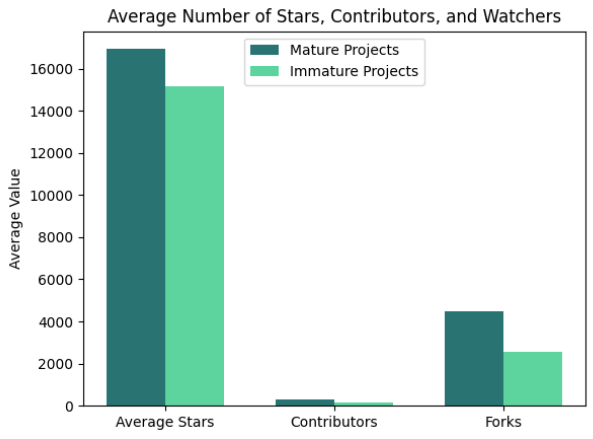

Figure 6: Average numbers of stars, watchers and contributors for mature and immature projects

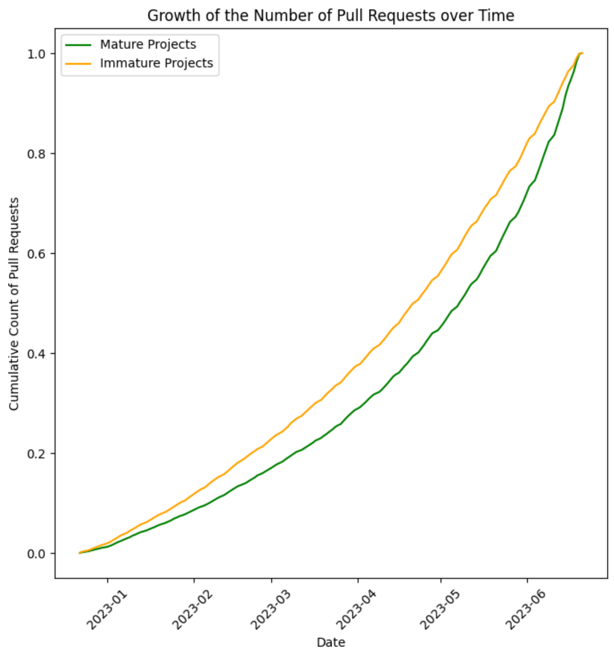

Figure 7: Growth of the number of Pull Requests opened over time

Growth rate in the number of PRs To further investigate the second research question, we analyze the growth rate of the number of PRs opened in the past six months. In Figure 7, both mature and immature projects demonstrate an upward trend in the number of PRs over time. However, mature projects exhibit a steeper growth compared to immature projects. This observation leads us to consider that the accelerated growth in the number of PRs for mature projects signifies a sudden increase in interest in the project once it has reached a certain level of maturity. In contrast, we could interpret the relatively consistent growth in PR numbers for immature projects as an ongoing implementation phase.

4.3 RQ3: What other metrics can be employed when studying the maturity of a project?

Age One of the concepts that is generally most associated with maturity is age. By studying this metric, we would like to investigate whether projects that exhibit signs of maturity are generally older than those that don't. Figure 8 highlights the age distribution in our dataset in the two clusters of projects. The plot on the left, which represents mature projects, highlights that the age distribution centers around 8 years and tends to be predominantly above this value. Conversely, the plot on the right, representing immature projects, displays an inverse pattern compared to the mature projects. Here, the majority of projects are clustered around the 4-5 year mark, with very few projects extending beyond 10 years of age.

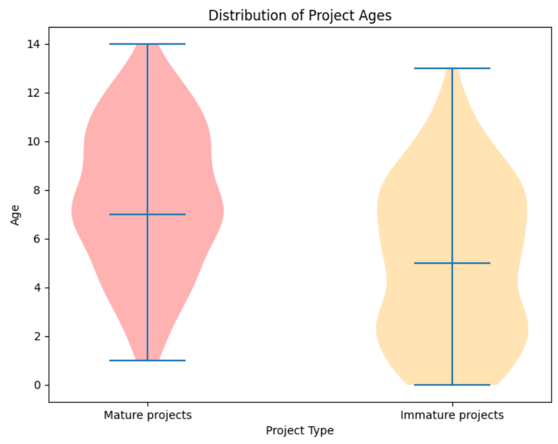

Figure 8: Age distribution of mature and immature projects

Size Due to time limitations, it was not feasible to study the size of the repositories in terms of source lines of code. However, we have analysed other metrics that can be related to size such as number of commits, pull requests, issues and watchers, considering how easily accessible these metrics were using GitHub's API. Our analysis revealed that mature projects generally exhibit higher values for these metrics compared to immature projects, as indicated in Table 3. However, it's important to note that our data may be influenced by the initial filtering approach used to cluster the projects, which involved evaluating project size based on file count.

5 Responsible Research

This section aims to address various aspects of the study to ensure the validity and reliability of our findings. Honesty, scrupulousness, transparency, independence and respon-

Table 3: Comparison of Size Related Metrics for Mature and Immature Projects

	Avg # Watchers	Avg # Commits	Avg # PRs	Avg # Open PRs	Avg # Open Issues
Mature	16933.80	50571.47	11596.48	155.17	1257.96
Immature	15156.69	6937.87	2344.48	48.79	333.96

sibility are the five most highly regarded principles by the ”Netherlands Code of Conduct for Research Integrity” [20]. In this context, our objective is to examine potential threats to the validity of our research and propose strategies to mitigate them, while also emphasizing the reproducibility of our study.

5.1 Construct validity

Construct validity refers to discrepancies between ”theory and observations” [12]. We recognize the following aspects as threats to the construct validity of our research:

- The filtering method is based on the closest approximation we could find through literature review for a framework to filter projects but this is not a generally adopted method for performing such task. Considering that we only filter the projects based on 3 criteria, it might be the case that some project falsely exhibit signs of maturity. Furthermore, the first filter, for CI implementation, might not contain an exhaustive list of all the naming conventions used for all CI tools, which might lead to falsely labelling a project as immature.
- We acknowledge that the use of median values for analysis across clusters might negatively impact the comparative study conducted as some key information might be obscured.
- GitHub’s default sorting based on stars may affect the quality and diversity of studied repositories, potentially introducing bias in the selection of projects. Although, we have tried mitigating this through batch retrieval of projects, there might still be bias in the data selected.

5.2 External validity

External validity is concerned with the extent to which our results can be generalised. With this, we acknowledge the following threats to external validity:

- Our study is based on a small dataset of 500 projects, split into mature and immature projects. Furthermore, the study of pull requests only considers a maximum of 500 pull requests and for a time frame of maximum 6 months. Thus, the generalizability of our finding is limited as the representativeness of the dataset under study is limited.
- When studying commits, we only take into consideration weekly churn for the lifetime of the project and the commit count for the past year. This approach may overlook important information and not be representative or comprehensive enough. This could impact the generalizability of the findings to a broader range of projects.
- We justify the limited amount of data analyzed by considering the API limit of 5000 requests per hour and the time frame in which the research was conducted.

5.3 Internal validity

Internal validity refers to the reliability of the results reached based on the methodology employed when conducting the study. Related to internal validity we make the following observations:

- The analysis and our results are limited to a rather small number of metrics. The knowledge gap regarding the general conception about project maturity contributed to the compilation of a very concise list of metrics which are generally perceived to be in relation with maturity.
- Our study fails to analyse some of the relevant metrics found through literature such as the relation between build level metrics and maturity, due to the limited time available for conducting the research.
- We base our observations on the comparative analysis of the two clusters of repositories, which were created using a filtering method of which reliability is only confirmed by one study and is not a generally recognized framework. Thus, the filtering process may have some faults, given that it is based solely on three aspects of projects, which we have not yet found in our research.

5.4 Reproducibility

To ensure the reproducibility of our research, we have made the code base publicly available on GitHub. The code can be accessed through a public repository named ”Descriptive-CI-Metrics”[4]. This repository serves as the foundation for our research and provides a comprehensive resource for replicating and verifying our results. By sharing the code base, we enable other researchers to examine and reproduce our methods, promoting transparency and facilitating further exploration in this domain.

By acknowledging and addressing the various limitations and threats to the validity of our research, as well as ensuring the reproducibility of our study, we have conducted the research responsibly. These considerations demonstrate our commitment to responsibility and transparency in our methodology.

6 Conclusions and Future Work

The primary goal of our study is to determine which metrics are most indicative of a project’s maturity level. To address this, we conducted an extensive literature review to identify

[4]https://github.com/raduConstantinescu/Descriptive-CI-Metrics

relevant metrics. Subsequently, we divided our dataset into two clusters, classifying projects as either mature or immature based on the only approximation of a framework to do so derived from the literature. By conducting a comparative analysis of these clusters we tried to measure the relevance that the previously identified metrics could hold in classifying mature projects.

To comprehensively investigate the impact of metrics on project maturity, we adopted a threefold approach, by dividing our main research question into three research subquestions. We examined the level of activity in projects, considering factors such as commit size and pull request merge speed, the influence of a project's popularity and community involvement, analyzing metrics like the number of stars, forks, and contributors, as well as the growth rate of pull requests and lastly, we investigated the significance of other project related metrics, such as a project's age and size.

In terms of activity levels, our findings suggest that commit size and pull request merge speed can serve as indicators of maturity. Mature projects tend to have smaller commit sizes, reflecting a focus on code fixes rather than the implementation of new features. Additionally, pull requests are merged more rapidly in mature projects compared to immature ones, indicating a higher level of project management and efficiency.

The main metrics related to popularity and community involvement, which show correlations with project maturity are the number of stars, forks, and contributors. Mature projects generally exhibit higher counts in these metrics, suggesting a greater level of recognition and engagement from the developer community. Although the growth rate of pull requests seems to be more accelerated in mature projects, this might be an area of further improvements in our study.

While age alone is not a definitive indicator of maturity, it can be used in conjunction with other metrics to provide additional insights into the maturity level. Size metrics, such as the number of commits, pull requests, and issues, also tend to be higher in mature projects. This could be either due to a more extensive development history and a larger user base, which correlates back to other metrics studied, or it could be a small imbalance in our dataset due to how we approached the clustering of projects.

Overall, our findings emphasize the importance of considering multiple metrics when evaluating project maturity. A combination of activity levels, popularity and community involvement, age, and size metrics provides a more comprehensive perspective on a project's maturity level compared to relying on individual metrics alone.

It is worth noting that our research is based on a specific and rather limited dataset, as described in Section 3.1.1, which was chosen while bearing in mind the time limitations for conducting the research and focused on a particular set of metrics. Future studies could expand upon our research to consolidate our findings by extending the dataset and study a higher number of repositories, while other studies could extend the list of metrics, such as metrics related to build performance, to further the exploration into the field of descriptive metrics that could offer information about a project's level of maturity.

References

[1] Moritz Beller, Georgios Gousios, and Andy Zaidman. Oops, my tests broke the build: An analysis of travis ci builds with github. 04 2016.

[2] Omar Elazhary, Colin Werner, Ze Li, Derek Lowlind, Neil Ernst, and Margaret-Anne Storey. Uncovering the benefits and challenges of continuous integration practices. 03 2021.

[3] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. pages 805–816, 08 2015.

[4] Xianhao Jin and Francisco Servant. A cost-efficient approach to building in continuous integration. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 13–25, 2020.

[5] Carmine Vassallo, Sebastian Proksch, Harald C. Gall, and Massimiliano Di Penta. Automated reporting of anti-patterns and decay in continuous integration. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 105–115, 2019.

[6] Miroslaw Staron, Wilhelm Meding, and Klas Palm. Release readiness indicator for mature agile and lean software development projects. In Claes Wohlin, editor, *Agile Processes in Software Engineering and Extreme Programming*, pages 93–107, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[7] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. Capability maturity model, version 1.1. *IEEE Software*, 10(4):18–27, 1993.

[8] Keheliya Gallaba and Shane McIntosh. Use and misuse of continuous integration features: An empirical study of projects that (mis)use travis ci. *IEEE Transactions on Software Engineering*, PP:1–1, 05 2018.

[9] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. How do centralized and distributed version control systems impact software changes? In *Proceedings of the 36th international conference on Software Engineering*, pages 322–333, 2014.

[10] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 60–71. IEEE, 2017.

[11] Sebastian Baltes, Jascha Knack, Daniel Anastasiou, Ralf Tymann, and Stephan Diehl. (no) influence of continuous integration on the commit activity in github projects. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics*, pages 1–7, 2018.

[12] Carmine Vassallo, Sebastian Proksch, Harald Gall, and Massimiliano Di Penta. Automated reporting of anti-

patterns and decay in continuous integration. pages 105–115, 05 2019.

[13] Thomas Durieux, Claire Goues, Michael Hilton, and Rui Abreu. Empirical study of restarted and flaky builds on travis ci, 03 2020.

[14] Taher A Ghaleb, Safwat Hassan, and Ying Zou. Studying the interplay between the durations and breakages of continuous integration builds. *IEEE Transactions on Software Engineering*, 2022.

[15] Gustavo Sizilio Nery, Daniel Alencar da Costa, and Uirá Kulesza. An empirical study of the relationship between continuous integration and test code evolution. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 426–436. IEEE, 2019.

[16] Xianhao Jin and Francisco Servant. A cost-efficient approach to building in continuous integration. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 13–25, 2020.

[17] Bogdan Vasilescu, Stef van Schuylenburg, Jules Wulms, Alexander Serebrenik, and Mark Brand. Continuous integration in a social-coding world: Empirical evidence from github. 12 2014.

[18] PyGitHub README. https://github.com/PyGithub/PyGithub. Accessed June 21, 2023.

[19] GitHub. About forks. GitHub Documentation.

[20] K Algra, L Bouter, A Hol, J van Kreveld, D Andriessen, C Bijleveld, R D'Alessandro, J Dankelman, and P Werkhoven. Netherlands code of conduct for research integrity 2018, 2018.