FSE 2022 Preprint

Code, Quality, and Process Metrics in Graduated and Retired ASFI Projects

Ștefan Stănciulescu
sstanciulescu@ucdavis.edu
University of California, Davis
USA

Likang Yin
lkyin@ucdavis.edu
University of California, Davis
USA

Vladimir Filkov
vfilkov@ucdavis.edu
University of California, Davis
USA

ABSTRACT

Recent work on open source sustainability shows that successful trajectories of projects in the Apache Software Foundation Incubator (ASFI) can be predicted early on, using a set of socio-technical measures. Because OSS projects are socio-technical systems centered around code artifacts, we hypothesize that sustainable projects may exhibit different code and process patterns than unsustainable ones, and that those patterns can grow more apparent as projects evolve over time. Here we studied the code and coding processes of over 200 ASFI projects, and found that ASFI graduated projects have different patterns of code quality and complexity than retired ones. Likewise for the coding processes – e.g., feature commits or bug-fixing commits are correlated with project graduation success. We find that minor contributors and major contributors (who contribute <5%, respectively >=95% commits) associate with graduation outcomes, implying that having also developers who contribute fewer commits are important for a project's success.

This study provides evidence that OSS projects, especially nascent ones, can benefit from introspection and instrumentation using multidimensional modeling of the whole system, including code, processes, and code quality measures, and how they are interconnected over time.

CCS CONCEPTS

• **Software and its engineering → Open source model**.

KEYWORDS

Open Source Sustainability; Code Quality

ACM Reference Format:
Ștefan Stănciulescu, Likang Yin, and Vladimir Filkov. 2022. Code, Quality, and Process Metrics in Graduated and Retired ASFI Projects. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549132

1 INTRODUCTION

Open Source Software (OSS) project sustainability is of key importance to our digital society infrastructure. Successful and sustainable OSS dominates the Internet (e.g., Apache, Linux, Android, Mozilla, Chromium, and LibreOffice), and is reportedly used in 98% of enterprises[1]. The blockbuster projects notwithstanding, for many other OSS projects, especially nascent ones, becoming sustainable and later remaining on a trajectory to long-term sustainability is an ongoing issue.

The consequences of a popular OSS project getting off that trajectory can be disastrous, as recent worldwide cybersecurity incidents like log4j have demonstrated [26, 44]. In response to such incidents in software security, a meeting in January 2022 at the US White House,[2] between government and private sector stakeholders, resulted in a statement recognizing OSS maintainability and sustainability as issues of national importance, and calling to action to identify "sustainable mechanisms" to maintain the most important OSS projects. But what are the characteristics of sustainable OSS projects, and how can they be leveraged into sustainable mechanisms for project maintenance?

We know that in spite of concentrated programmer labor, many open source projects fail and are abandoned. Coelho and Valente have studied over 100 GitHub projects and identified different reasons for OSS failure, including low number of developers involved, low engagement, and other reasons. [8]. To increase their visibility and chances of success many projects join foundations, like the Apache Software Foundation (ASF), which provide common standards and guidance in exchange for higher community uniformity. The popularity and high standards of foundation supported projects can contribute to their continued sustainability, by attracting a steady supply of programmer effort. ASF in particular serves as an example that has been successfully churning out popular and high quality projects for years. Its incubator, and the Apache Way,[3] has served many nascent projects, graduating those that demonstrated a track record toward sustainability, and retiring others.

Moreover, OSS projects come in different shapes and sizes. This diversity is there from the beginning, when nascent projects are initiated, and continues throughout their life trajectory, and for some, into their eventual sustainable regime. That every project may have a different story and trajectory is in line with contingency theory and organizational management studies [14]. Recognizing this, modern software engineering research is doing away with the "rules of thumb", and the one-size-fits-all solutions and replacing them with deeper and more meaningful bespoke solution, where the context and the ecosystem environment are significant determinants of project sustainability [21, 52]. These research efforts

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3549132

[1] https://www.pcmag.com/archive/survey-98-percent-of-companies-use-open-source-29-percent-contribute-back-253661
[2] https://www.whitehouse.gov/briefing-room/statements-releases/2022/01/13/readout-of-white-house-meeting-on-software-security/
[3] https://www.apache.org/theapacheway/

have benefited from the availability of large-scale repositories of project traces and developer behaviors. The most promising of them have combined multiple different types of data, including traces and communication activities, surveys and developer interviews, and used modeling at both software engineering and social and governance levels [51, 52]. Recent work on open source sustainability focused both on technical factors [36, 37, 41] and social factors [17, 47, 49, 52], and has demonstrated the benefits of using socio-technical networks to forecast project graduation success in the Apache Software Foundation Incubator (ASFI) [21, 52]. Those studies found that different patterns of social interactions and different socio-technical behavior mediate differential success outcomes, in particular with respect to sustainability metrics.

Where the evidence is scant, however, is where we find the next natural research direction: to connect sustainability to the way the software artifacts are created, especially the code. It stands to reason that because code is the product of coding, a social process, and the process of coding is interleaved with activities of the socio-technical system representing the project, a connection, even if indirect, exists among them. In fact, since the initial postulation of Conway's law [9], it has been amply established that the software project socio-technical structure in commercial projects is associated with the organization [33] and quality [6] of the software artifacts created.

A Motivating Example. We take a look at two different yet similar projects from ASF. *Apache Celix* is a framework to develop modular software applications. It graduated from the incubator after almost four years, while having only four code contributors and a bit over 300 commits. Its approach was to have a consistent long-term activity, without high intensity development periods. On the other spectrum is *SpamAssassin*, a well-known anti-spam platform developed for more than 20 years. SpamAssassin was highly active during its incubation period, had many contributors, and averaged over 150 commits per month. It graduated much faster from the incubator, likely also due to its previous well established codebase and community. Both projects were and still are successful, both graduated from the incubator, but had very different trajectories both at code level and at process level.

1.1 Our contributions

Guided by the above, and the prior work showing that the notion of OSS sustainability, in the ASF Incubator sense of a trajectory leading to self-reliance has strong connections to a project's socio-technical structure, we hypothesize that:

> Code, the coding process, and project quality (e.g., graduation in ASFI) are associated with project sustainability in a way that can be quantitatively measured.

The following questions, then, arise naturally: is the code and the development process in sustainable projects perceptibly different than in those that are not sustainable? Is the code quality in the former any different than in the latter? To effectively answer these questions one would need guiding theories connecting organizational behavior to outcomes, and data of code, process, and quality, together with data of project sustainability outcomes.

Here, we analyze a dataset of source code and digital traces from the repositories of 200+ ASFI projects, that have been judged to be on the path to sustainability, i.e., retired or not. We extracted software metrics for code, process, and quality, and studied the metric-space differences between graduated and retired ASFI projects at individual metric level, metrics over time, and in models of sustainability. We found the following:

- Graduated and retired projects are different in their code (graduated have less code per author, and less directories per author), processes (graduated commit more and delete more), and quality (graduated have more complex code and more test code);
- Graduated and retired projects follow different trajectories once they enter the incubator. Some projects are somewhat better equipped to graduate fast, while others strive for a more constant but less commit-heavy activity. Finally, retired projects are more likely to have a higher burden per contributor due to having fewer contributors, an increasing codebase size, and being less likely to attract new contributors;
- An increase in the following metrics increases the odds of graduation: lines of code, major and minor contributors, features commits, corrective commits, medium complexity (11-25 McCabe) functions, and very large functions. On the flip side, the increase in the following metrics decreases the odds: top level directories, avg. files modified per commit, very large file sizes, and code duplication percentage.

This paper is a first step showing that it is possible to associate sustainability with code, quality and process metrics. Our motivation goes beyond ASFI as many more projects fail outside of ASFI, therefore, maintaining projects with large code bases may be more pertinent to them. Our replication package is available on Zenodo.[4]

2 BACKGROUND AND THEORY

In this section, we summarize the underlying mechanism of the Apache Software Foundation Incubator, the related work on OSS sustainability, and the concepts on code complexity and quality measures. We also introduce the relevant contingency theory.

2.1 Apache Software Foundation Incubator

Apache Software Foundation Incubator (ASFI),[5] is a project incubator that provides a deterministic path for in-the-wild OSS projects to attract highly skilled developers, regulate working behaviors, and further joining Apache community. During the incubation, the projects' long-term goal is to become self-sustainable, i.e., the project's community can self-govern itself to sustain its activity and productivity over time. To help incubating projects achieve such level of sustainability, ASF, different from most OSS foundations, provides each incubating project with in-depth mentorship from senior ASF committers. The incubation process often takes several months, and when the project's community is ready for exiting the incubator, all contributors have the obligations to vote for either graduation and retirement; if the decision from the community is positive, then the ASF committee will additionally evaluate the projects' progress on contributor diversity, attractiveness to new contributors, and community building. If all goes well, the project will be graduated, otherwise retired. There is a tenet in Apache

[4] https://zenodo.org/record/6374071
[5] https://incubator.apache.org/

Community, and that is 'Community Over Code'. The belief is that if good care is taken of the community, good code will emerge from it. Based on such belief, projects in ASFI are encouraged to build a diverse and meritocratic-based community, though there is little explicit emphasis on code quality or new features to be developed. Previous work has found that there are strong relationships between code quality and project's progress [1, 48].

2.2 OSS Success & Sustainability vis-a-vis Code

OSS success has been studied from various angles, though there is still no universal agreement on the definition of success in OSS projects [10, 23, 40]. Some researchers use metrics from the user's perspective to measure OSS success, e.g., project downloads, usage, popularity, licenses [7, 31, 43]. Others use project-centered metrics, e.g., the number of issues, commit frequency, active developers [23].

Coelho et al. surveyed 118 developers and found that projects fail for several reasons [8]: low interest from developers, low code maintainability, and newer projects overtaking the original projects. Furthermore, they discover that failed projects are less likely to follow best practices (e.g., provide contributing guidelines), and failed projects are more similar to less popular projects. A codebase with high complexity and many code dependencies requires more efforts to maintain or to develop new features. Code complexity can also increase due to low code ownership [6], developer turnover or accepting external contributors' code [16]. Sachs used developers' comments to predict a project's success [39] by analyzing the interactions between individuals as a group, based on SYMLOG, a body of social psychology theories [2]. They find that developers' personas do not increase the accuracy of traditional metrics-based methods for predicting a project's success.

Code quality is an important non-functional requirement in a project's lifecycle. Nagappan et al. show that organizational metrics fare better than traditional metrics in predicting faults in OSS projects [33]. Other research has shown that code and process metrics can be useful for detecting code smells [29], design issues [20, 32], predicting defects [36] or finding vulnerabilities [42], which can be useful in understanding a project's evolution and potential issues.

Although related, OSS sustainability and OSS success are two different concepts. One project can be successful and not sustainable and the most recent *log4j* vulnerability [26] is a convincing evidence: a well-known and widely used OSS project, but barely maintained by a group of only four non-paid developers. Xia et al. used GitHub related metrics (pull requests, issues, etc) to predict the health of a project [50]. Compared to previous work, they achieve high accuracy (<10% error rate) by tuning their models and parameters, and show that they can accurately predict many health indicators for the next 1, 3, 6, and 12 months. More recently, Yin et al. showed that socio-technical factors (mostly social and technical networks) can be used to forecast the incubation sustainability outcome for ASF incubator projects [52]. They find that a higher number of nodes in the social networks is positively correlated with graduation, whereas the number of nodes in the technical network (code and developers' coding behavior) is negatively associated with graduation for minor projects. Ghapanchi studied project sustainability over time based on two metrics: defect rate velocity and feature enhancement rate [18]. They found that a higher bug-fixing and feature enhancement rate, together with an increase in the frequency of releases helps with long term sustainability. In comparison, we use comprehensive source code and coding process analysis, allowing us to be broader in our analysis by including code quality metrics. Izquierdo et al. studied how OSS projects evolve in the Eclipse incubator [19]. They use product and code related metrics to gauge their maturity. They find that modeling projects that went through the incubator remain stable throughout their lifecycle, indicating that they do not grow and are not able to attract new contributors. As the end result of OSS software is producing useful software, we expect that project sustainability is associated with its code characteristics. However, to the best of our knowledge, there is no study articulating the association between project sustainability and code, process, and quality metrics.

2.3 Contingency Theory

ASFI projects, regardless of size, naturally form an organization with internal rules, special regulations, and exclusive person power. Structural contingency theory argues that organizations cannot be fully effective without the necessary structural support [14]. Moreover, there is no single organizational structure that is best fitted for every organization [46]. Thus, to an organization, the most effective structure is contingent upon its internal and external context. According to Ruekert et. al. [38], organizational performance is a mix of many factors, including: structure, people, technology, strategy and culture. Organizations with better fit will achieve higher levels of performance. In a nutshell, the structural contingency theory claims that 'one size does not fit all', and that effective performance depends on many factors [53]. In the specific case of ASFI projects, the context of a project is multi-dimensional, including communications among contributors, seniority of the contributors, and programming languages being used to build the project. Therefore, it is likely that the association between code quality and sustainability is contingent upon those context variables.

3 RESEARCH QUESTIONS

In this paper we build on the previous work of Yin et al. [52] and study metrics complementary to their socio-technical ones. Our focus is on code, process, and quality metrics to explore how writing code differs across ASFI projects. We hypothesize that retired projects will exhibit different code complexity, process patterns, and quality measures than graduated projects.

To have actionable insights into projects evolution and sustainability, we consider them in a multi-dimensional space of our metrics. Understanding projects in such a space can give us a glimpse into projects' similarities and differences. Even if projects can be similar in terms of codebase size (measured by the number of files) and project size (measured by the number of active developers), we expect that projects can differ in many ways: they may differ in some process related aspects such as commit frequency, change size, commit types, or the ability of attracting new contributors. As predicted by the contingency theory, projects that successfully progressed to be self-sustainable can go through different trajectory under different context, and the same holds for retired projects that they fail to maintain sustainability due to various context. Therefore,

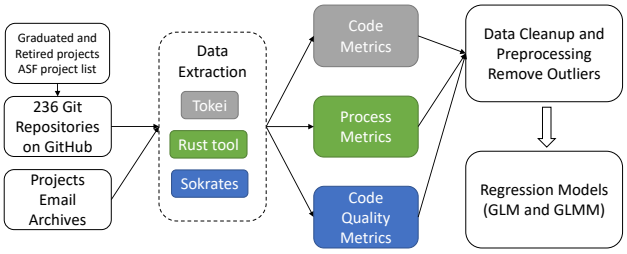

Figure 1: Diagram showing the overall process for collecting and analyzing the data. GLM: generalized linear regression model, GLMM: generalized linear mixed-effects regression model.

in such a complex multi-dimensional space, it is of importance to demonstrate that there is sufficient variance to capture differences and similarities between graduated projects and retired projects. Thus, we ask:

RQ$_1$: How do graduated and retired projects differ across our code, process, and quality metrics? How are the metrics related pairwise?

Next we turn to the temporal evolution of code, process, and quality metrics. In a socio-technical system, feedback from one side to another side takes time, especially in the circumstance of the OSS project where people can work from different time zones. As predicted by the contingency theory, the context of projects is of importance to the organization, and the context is changing over time. Therefore, code, process, and quality metrics' evolution are ideally to be studied temporally, which has been shown to perform better than static metrics for predicting buggy code [36]. We ask:

RQ$_2$: Are project trajectories of graduated and retired ASFI projects different, along our code, process, and quality metrics?

Finally, we put the metrics together in a model of project graduation. This question aims at exploring how well can our code metrics associate with graduation from the incubator. We expect the association between quality and sustainability can be identified while controlling the context, thus we ask:

RQ$_3$: What are the code, process, and quality metrics determinants of whether a project is graduated versus retired? Are they different for different programming languages?

4 DATA AND METHODS

In this section, we present the data collection process and the methods used in the study. Figure 1 presents the overall process.

4.1 Data Collection

We use the Apache Software Foundation *podlings.xml* file,[6] for a complete project list. It contains the project meta-data: project name, incubation start and end date, and current status: graduated, retired, or incubating. In total, the list contained 328 projects, out of which, 236 have a Git repository on GitHub. The rest of the projects are either SVN based or we were not able to locate their repository, thus we exclude them. Finally, we clone all 236 Git repositories from Apache's GitHub organization.[7] To collect the metrics from the Git repositories and the mailing archives, we built a tool in Rust that extracts data from repositories and downloads the mailing archives. It then launches two other popular tools: *tokei* for extracting code size metrics (files, blanks, lines, code, comments),[8] and *Sokrates* for calculating code quality metrics,[9] and collects their output. For each Git repository, we extract all commits and organize them into monthly batches, called *incubation months*, based on their date. A project's incubation months are computed for each month from their entry into the incubator (start date) and until they exit the incubator (end date). If a project enters the incubator on 2006-10-15 and leaves the incubator on 2007-01-26, then the first incubation month runs from 2006-10-15 until 2006-10-31, the second incubation month runs from 2006-11-01 until 2006-11-30, and so on, until the incubation month #4 which runs from 2007-01-01 until 2007-01-26. Note that the first and last month may have less data. This allows us to more easily align with the mailing archives which are stored per calendar month.

We analyze the commits and extract process metrics for each incubation month. To collect code metrics and code quality metrics, we check out the repository at the last commit in that respective incubation month and run the tools on the source code. Our Rust tool computes the number of directories and root level directories and extracts process metrics. We use *tokei* to extract code size metrics. *tokei*'s output provides these metrics for each programming language that it finds in the project, which allows us to also identify and record the dominant programming language, i.e. the one that has the most lines of code for each incubation month. To smooth the data for smaller projects, if in a particular incubation month there have been no commits, we reuse the code metrics values from the previous month. If there has been no activity in the first incubation month, we initialize all code and process metrics to 0.

For the classification of commits into change types, we use the classifier from [15], and use the available generated model into which we feed all the commits' messages.[10] The output is a list with all commits and their classification label: feature, corrective (fixing a bug), perfective (improving the code), non-functional (not related to source code, e.g., documentation), or unknown (unsure how to classify). We aggregate this as a sum of commits that have the same label, for each incubation month and project.

To measure differences in their development process, we compute for each incubation month a selection of several popular metrics [6, 19, 20, 36] from the project's Git repository: number of commits (excluding merge commits), average commits per developer, number of commit active days, files added, files deleted, files modified, average number of files modified per commit, churn (added + deleted lines), minor contributors (those that provide less than 5% of commits), major contributors (those that provide 95% or more of commits), new contributors (those that are first time commit contributors), number of emails. For code metrics, we record the number of files, lines, comments, blanks, and code (sloc), project root level directories, and total number of directories.

[6] URL to the podlings.xml file, downloaded June 2021

[7] https://github.com/apache/abdera/

[8] https://github.com/XAMPPRocky/tokei

[9] https://www.sokrates.dev/

[10] https://github.com/gesteves91/fasttext-commit-classification

For the emails, we download the *dev* mailing list archives for each project and each incubation month.[11] We focus on direct and explicit communication between developers, therefore we count the number of emails that are not JIRA-related emails, but we keep those JIRA emails containing "Commented" in the subject. We ignore other kind of JIRA emails (e.g., new issue, issue resolved, issue assigned, etc.). We then extract a set of the developers' emails that sent or replied to an email (emails_dev), and the number of emails. In a month with no emails, these metrics have a value of 0.

We use Sokrates to extract a number of metrics related to functions, file sizes, McCabe complexity metrics at function level, code duplication, and tests for each incubation month. While Sokrates is not the most obvious choice, due to the fact that our dataset exhibits a large number of programming languages, Sokrates is, to the best of our knowledge, one of the few tools that can perform such widespread source code analyses.

Our datasets, R scripts, and tools are available on Zenodo.[12]

4.2 Models

To analyze how graduation is associated with code, processes, and code quality, we use Generalized Linear Regression (glm), and Generalized Mixed Effect Regression (glmer) models [4]. We chose the latter to account for potential random effects introduced by different programming languages. We use R and the *glmer* function from the *lme4* package [5] to build the models. We first clean up the data by removing a number of projects that have either no activity or have incomplete data. From the total 236 Git repositories, we eliminate 18 projects, resulting in 5365 observations across 184 graduated and 34 retired projects. We further remove the top 3% of the outliers, resulting in 5245 observations.

For each incubation month we record the most prevalent programming language in terms of code, and we use that as a random effect. We build three regression models: a) the base model, b) the base plus the process metrics, c) and the base plus process metrics plus code quality metrics. The goal is to observe how the addition of process and code quality metrics associate with predicting the outcome - graduation or retirement. We build these three models for two datasets: a **glm** model using only the Java-based projects (which are the majority of the projects in the dataset), and a **glmer** model that includes all the projects and uses the programming language as a random effect. Recall that we are also interested in understanding if programming languages play a role in predicting the outcome. Next, we check for multicolinearity using the *Variance Influence Factor* (VIF) using the *check_multicolinearity* function from the *performance* R package [28]. Values below 5 indicate that multicolinearity is not significant. We use two pseudo R^2 values to report the model's goodness of fit, using the *r2_nakagawa* function from the same *performance* R package. The marginal R^2 represents the variance solely described by the fixed effects, and the conditional R^2 represents the variance introduced by both fixed and random effects in the model [34]. For creating the results table, we use the SjPlot package [27] to create an HTML table, which we then convert to Latex. The table shows the dependent variables (Predictors), the Odds-Ratio (a value >1 has a positive effect, and <1 has a negative effect on graduation) and P values. The probability definition of odds-ration is: $prob = \frac{odds\_ratio}{1 + odds\_ratio}$. For example, an odds-ratio of 1.8 means the probability of success is 64% higher with a unit increase in the selected variable.

4.3 Variables of Interest

For modeling, we use 23 independent variables representative of software metrics split into three categories:

§1 code: *lines of source code without comments (SLOC), number of directories, number of top level directories.*

§2 process: *major contributors*, *minor contributors*, *new contributors* - the number of authors that have not contributed before the current incubation month, *files added* - the number of files added in this incubation month, *files deleted* - the number of files deleted in this incubation month, *avg. files modified per commit* - the average number of files that were modified in a commit - it excludes files that were added or deleted, only those that existed and were modified, *active days* - the number of days in an incubation month that had at least one commit, *number of emails*, *corrective* - the number of commits that fixed a bug or an issue, *features* - the number of commits that added some feature, *perfective* - the number of commits that enhanced (e.g., performance) the code, *non functional* - the number of commits that were not code related, for example adding documentation, and *incubation month.*

§3 quality: *test code SLOC* - this provides a proxy for how mature a project is and its quality assurance, *the ratio between test SLOC and SLOC as a percentage* - a small value represents a project with few tests and weak code coverage, *number of functions that have a medium risk complexity (McCabe index between 11-25)* - functions should have a small cyclomatic complexity in order to be more easily maintained and tested, *number of functions that have a very high risk complexity (McCabe index >50)* - very complex functions are indicators of bugs hotspots due to being very difficult to test, *number of very large files (>1000 SLOC)* - large files have been shown to be more likely to contain faults [35], require more efforts from developers to understand and manage, and make the code less readable, *number of very large functions (>100 SLOC)* - large functions require extra effort to maintain and make code less readable [30], *number of lines of code for the most complex function.*

5 RESULTS

In this section, we present the analysis results and summarise our findings for each research question.

5.1 RQ_1: How do graduated and retired projects differ across our code, process, and quality metrics? How are the metrics related pairwise?

Our dataset consists of 5365 monthly snapshots from 218 projects (across all their incubation months). Among the 218 projects, 184 are graduated and 34 are retired projects. Table 1 shows the descriptive statistics of our data across all 23 metrics.

In contrast to Yin et al. [52], here we focus on the properties of the code, process, and code quality metrics. Graduated and retired projects distinguish themselves in several ways. First, one

[11] http://mail-archives.apache.org/mod_mbox/
[12] https://zenodo.org/record/6374071

Table 1: Descriptive statistics of graduated projects and retired projects. The mean, median and st. deviation values are computed over all incubation months. Authors are developers who made changes, though they did not necessarily commit those changes.

	184 graduated projects			34 retired projects		
	mean	median	st.dev	mean	median	st.dev
SLOC	123431.66	66219.00	200938.97	89866.66	37513.00	145736.58
Commits	50.36	22.00	90.96	11.39	2.00	23.70
Directories	435.19	222.00	583.98	311.33	176.00	316.17
Top Level Directories	10.31	8.00	10.26	9.53	8.00	7.22
Incubation Months	15.98	12.00	13.42	24.71	21.00	17.82
Authors	5.69	3.00	7.88	1.52	1.00	1.96
Major Contributors	2.92	3.00	2.18	1.29	1.00	1.36
Minor Contributors	2.73	0.00	6.73	0.21	0.00	0.85
New Contributors	1.65	0.00	3.72	0.37	0.00	1.03
Files Added	172.45	18.00	807.46	56.27	0.00	432.19
Files Deleted	131.30	3.00	735.82	37.94	0.00	379.24
Files	957.01	609.00	1077.36	807.78	371.00	1227.41
Avg Files Modified per Commit	5.72	3.33	11.78	5.26	1.49	23.14
Active Days	10.64	9.00	8.55	3.83	2.00	5.38
Emails	131.88	63.00	217.79	42.48	16.00	79.04
Corrective	11.60	5.00	18.56	2.47	0.00	5.65
Features	14.07	6.00	25.52	3.27	0.00	7.04
Perfective	23.34	9.00	53.25	5.41	1.00	11.98
Non Functional	0.86	0.00	1.83	0.17	0.00	0.59
Number of functions	3570.97	2350.00	3934.19	2768.23	1384.50	3888.07
Test/Main Lines of Code Percentage	47.43	27.56	126.30	32.87	23.41	36.01
#Functions /w McCabe Index 11-25	81.81	47.00	107.22	52.74	24.00	63.94
#Functions /w McCabe Index >51	2.47	0.00	5.41	0.82	0.00	1.69
Very Large File Size Count	6.89	1.00	21.97	3.40	1.00	4.69
Very Large Function Size Count	17.17	5.00	32.10	9.74	3.00	15.41
Code Duplication Percentage	16.38	12.69	14.92	16.93	13.63	15.69
Most Complex Function LOC	305.67	134.00	1193.30	178.81	120.00	156.75

requirement for graduating from the incubator is to attract new contributors to the projects. Our data indicates that retired projects are on average much less likely to attract new contributors during their incubation period, and have fewer main contributors. Second, projects have different code and process characteristics. Graduated projects have more SLOC, but retired have more SLOC per contributor. Similarly for directories. And while retired projects are smaller in size, they modify on average 5.2 files per commit. Graduated projects modify on average 5.7 files per commit, but they are much larger and have more files. Thus, we can argue that retired projects have a different committing style, a consequence, possibly, of less stringent rules on how to make changes. Furthermore, retired projects seem to delete significantly fewer files. Third, there are differences from a code quality perspective. Graduated projects seem to be more complex, having more large files (>1000 LOC), and almost twice the average of the number of very large functions (>100 LOC) when compared to retired projects. Perhaps in line with that, graduated projects exhibit more test code than retired ones.

> **RQ_1 Summary:** Graduated and retired projects differentiate themselves through their code (graduated have less code per author, and less directories per author), processes (graduated commit more and delete more), and quality (graduated have more complex code and more test code).

5.2 RQ_2: Are project trajectories of graduated and retired ASFI projects different, along our code, process, and quality metrics?

We find that the average incubation time over all projects is 18 months, with the average for graduated projects being 16 months, and for retired projects 24 months. Retired projects stayed on average 8 months longer in the incubator than graduated projects. We observe that a number of projects that stay long in the incubator, still manage to graduate — 28 graduated projects stayed 36 or more months in the incubator and still graduated, whereas 16 of the 34 retired projects stayed at least 36 months in the incubator. A feasible reason for this difference is that on average, retired projects may need longer time to develop, and get the appropriate accommodation for that. Yin et al. [52] found that while the shorter time spent in incubation is a characteristic of many graduated projects, that by itself is not a very good predictor of graduation.

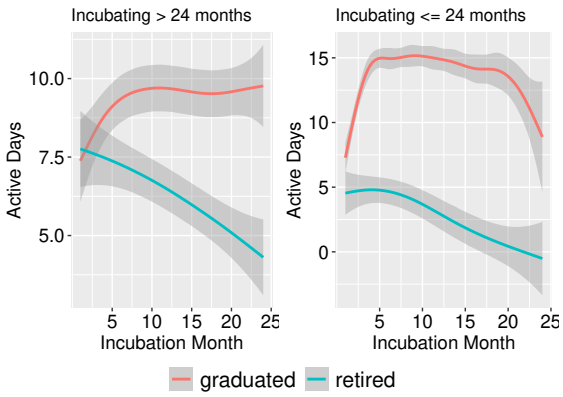

Figure 2: The number of commit-wise active days per incubation month. An active day is a day that had at least one commit. On the left are projects incubating for 24 months or more, on the right projects that spent 24 months or less in the incubator. For readability reasons, we show only the first 24 months of data.

On the other hand, projects that stay shorter in the incubator seem to do so because they are better equipped to graduate from the start (see Fig. 3, right plot), and even increase their average

Figure 3: Average number of commits per developer metric. Left plot shows projects that were in the incubator for more than 24 months and right plot for projects that spent less 24 months or less in the incubator. For readability reasons, we show only the first 24 months of data.

number of commits per developer in the first few incubation months. Figure 2 shows the number of active days in each incubation month. Retired projects start with only five days of commit activity a month (for projects incubating <= 24 months), and that decreases towards no activity after 20 months of incubation. Similarly, retired projects in projects that are in the incubator for at least 24 months, have a linear decrease of activity, reaching less than four active days around month 22 in the incubator, whereas graduated projects tend to average 8-10 active days per incubation month for longer period of times. It is clear that graduated projects have a more steady commit-activity throughout the incubation month, which might be one of the main factors for a more successful long-term project development.

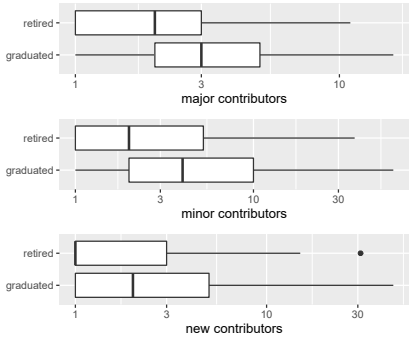

Figure 4: Major, minor, and new contributors (log-scale) in graduated and retired projects.

Along commit activity, we want to understand what kind of contributors contribute to a project. We define three types of contributors: 1) major contributors, those that commit 95% or more of the total commits; 2) minor contributors, those that contribute less than 5% of the total commits, and 3) new contributors, those that are first time contributors (have not contributed before). Figure 4 shows that graduated projects are capable of attracting a few new contributors, whereas retired projects are less likely to do so. Analyzing the minor/major contributors throughout the incubation period, we observe a linear increase for both minor and major contributors in graduated projects, and the opposite in retired projects. Graduated projects are likely more capable to recruit new developers and retain existing ones (indicated also by the new contributors metric). Interestingly, minor contributors seem to be more prevalent in graduated projects than in retired projects. From a sustainability perspective, minor contributors could become committers and maintainers of the project. Thus, projects and maintainers should consider attracting and providing guidance to minor contributors as a way of expanding the project's community.

Next, we focus on understanding the relationship between developers and files. We compute the ratio between the number of files and the number of contributors, which provides us with a measure of development effort. Figure 5 shows two plots: on the left, we see the ratio between the number of files and number of contributors, whereas on the right we see evolution of the codebase size in terms of SLOC. We observe that retired projects' size increases in the first 18 months of the incubation, and the ratio of files/contributors

Figure 5: Plots showing the ratio between files and number of developers (left) and the evolution of the code size (right) for all 218 projects.

increases linearly as well. That means that while retired projects' codebase is increasing, the number of contributors decreases or stays the same. In other words, retired projects do not seem to be able to sustain their development efforts due to fewer contributors. In the case of graduated projects, we see the ratio of files/developer also increasing, but being quite lower than those of retired projects. This indicates that graduated projects, while bigger in size, are more capable of attracting or retaining developers to cope with the increase of the codebase's size.

Projects evolve differently from a process perspective also, leading to different outcomes. Graduated projects exhibit a higher number of commits per developer, have more commit active days throughout the incubation, recruit more new contributors, and are able to increase and sustain their activity in the first few months in the incubator. They also communicate more via emails. These projects have a different approach to working once they enter the incubator. In projects that incubate for a shorter period of time, retired projects are less active from the start, and become inactive (commit-wise) towards month 22 of the incubation period. Projects that are able to have more commit activity throughout each month, are more able to keep ongoing the development. Previous work by Yin et al. [52] showed that the number of commits and the developer network are correlated with graduation. We further find that not only the number of commits is relevant, but also how often and what kind of changes happen. Projects should also keep an eye on their code quality and complexity, in addition to grooming long-term and short-term contributors.

Case Study. From the data we collected, we posit that there are different incubation strategies. Some projects have a a short but high intensity development period, and therefore they graduate fast. For example, SpamAssassin spent 7 months in the incubator and had an average of 173 commits per month from 5.8 contributors.[13] Another example is Spark which spent 9 months in the incubator averaging 272 commits per month from 35 contributors.[14] These were clearly under intense and sustained development.

Other projects strive to keep an ongoing activity despite the fewer commits and email exchanges. Apache Celix is a framework

[13] https://github.com/apache/SpamAssassin
[14] https://github.com/apache/spark

to develop modular software applications.[15] The project incubated for 45 months, had four contributors and 308 commits. However, for 39 months there was some activity in the project, either via commits or via emails. The project is still active and under development after its graduation. An opposite example is Weex,[16] a framework for building Mobile cross-platform high performance UIs. Weex spent 55 months in the incubator and had a very high commit activity for the first 12 months (>3500 commits). From month 39 to month 55, there was almost no activity (only 35 commits in total) and developers faced issues in graduating, which ultimately meant that the project had to be retired. After its retirement, it moved under *alibaba*'s open source organization, and continued to be developed. These projects forged their own path and evolved in different ways. As such, there is no exact one way a project may go; instead they need to constantly adapt and adjust based on their current processes, code, and other socio-technical factors.

> **RQ_2 Summary:** Graduated and retired projects follow different trajectories once they enter the incubator. Some projects are better equipped to graduate fast, while others strive for a more constant but less commit-heavy activity. Finally, retired projects are more likely to have a higher burden per contributor due to having fewer contributors, an increasing codebase size, and being less likely to attract new contributors.

5.3 RQ_3: What are the code, process, and quality metrics determinants of whether a project is graduated versus retired? Are they different for different programming languages?

In the previous RQs we examined metrics pairwise and in relationship to ASF sustainability. To study the relationship between sustainability and code quality, in the presence of multiple other metrics, we use generalized linear and mixed effects logistic models. We have 25 different programming languages in our data set, with a large spread of the number of projects using a specific language, from 160 projects in Java to 1 project using CVX. As Java is the predominant programming language used by ASF incubator projects, we first provide insights from projects that use Java as the primary programming language, and then we compare the models built on only Java projects to models on all projects programming languages in the ASF incubator. In the following we present those two studies.

Java Projects. The results of the first study, on all Java projects are shown in Table 2. The three regression models are predicting the binary outcome of graduation or retirement, the first using base (code) metrics, the second adds process metrics to the base, and the third is the full model, with all code, process, and quality metrics. This allows us to understand the contribution of the different metrics groups to the efficacy of the models. We will use the full model for explaining the details.

In code metrics, as expected, more SLOC increases the odds of graduation by 15%. The top level directory number is interesting as it is significant and negative, implying 21% lower odds. This is consistent with the notion that less complex code is easier to maintain and thus needs fewer people. As a control, having a higher value for the incubation month decreases graduation substantially (41% lower odds), as discussed earlier in RQ2.

In process metrics, major contributors is positively and significantly associated with graduation, yielding 84% increased odds; that is, having more major contributors that contribute 95% or more of the total commits, the more likely it is to graduate (i.e., to become ASF sustainable). The effect of minor contributors is small (5% increase in odds) but significant. Prior work has shown they are valuable. They often pick low hanging fruit issues to fix, or work on the code for their own needs. In the long term, some of those minor contributors are given commit access and the possibility to join the project, enabling a project's long-term sustainability.

Code usually evolves through different kind of changes, including adding files, removing files, fixing bugs, adding new features, creating documentation or performing re-factorings. We use the number of added files and the number of deleted files to simulate the dynamics of the code evolution from a file organization perspective. The model shows that there is a small negative effect (3% decrease in odds) and borderline significant for projects that delete more files during their incubation period. Moreover, we find that the average number of files modified per commit has a significant negative effect on graduation, lowering the odds by 6%. Ideal changes should of course be small and well documented in a commit. When the code is complex with many dependencies, as is the case with graduated projects on average, it is likely that more files would have to be modified per commit. This could be attributed to code dependencies, less scrutiny over how to commit changes, or developers experimenting more with the code.

We also find that two types of commit changes are associated with graduation: corrective commits, which fix bugs and issues, and feature commits, which that add new features, have a significant positive effect on graduation, each adding 7-8% to the odds.

In code quality metrics, first, having a higher number of files with more than 1KLOC has a sizeable negative effect (19% drop in the odds) associated with graduation. One possible reason for this is the fact that large files are more difficult to maintain and test, and to collaborate on. On the other hand, projects that have a higher number of functions of very large size (>100LOC) are more likely to be self-sustainable, by almost the same odds difference. This is somewhat surprising and merits further study. Given the verbosity of the Java language, we would expect that simply having more functions (modularity) of reasonable sizes is important, from the maintenance, program comprehension (readability), and testing perspective. While sometimes code clones are not considered harmful [22], many other times cloned code leads to more maintenance effort due to the need of fixing the code in multiple places. The model shows that the more duplicated code a project has, the less likely it is to graduate. Finally, and surprisingly test LOC/code LOC does not appear to be significant for sustainability.

All Projects. We performed a second study on all projects after excluding nine projects that solely use a certain language (i.e., we eliminate projects whose dominant coding language is only used by them). There we used a generalized linear mixed effect model, with *programming language* as the random effect. The results are

[15] https://github.com/apache/celix
[16] https://github.com/apache/incubator-weex

Table 2: Summary of three GLM models for Java-based projects. Due to *code* and *directories* being highly correlated, we removed the *directories* variable. Similarly, we removed the *most complex function LOC* variable.

	Base		Base, Processes		Base, Processes, Quality	
Predictors	Odds Ratios	p	Odds Ratios	p	Odds Ratios	p
Intercept	0.83	0.378	3.49	**<0.001**	1.85	0.120
SLOC	1.22	**<0.001**	1.11	**0.001**	1.15	**0.004**
#Top Level Directories	0.79	**<0.001**	0.80	**<0.001**	0.79	**0.001**
Incubation Month	0.48	**<0.001**	0.63	**<0.001**	0.59	**<0.001**
#Major Contributors			1.82	**<0.001**	1.84	**<0.001**
#Minor Contributors			1.06	**0.012**	1.05	**0.022**
#New Contributors			0.97	0.151	0.98	0.227
#Files Added			1.02	0.372	1.02	0.245
#Files Deleted			0.96	**0.010**	0.97	**0.047**
Avg. Files Modified per Commit			0.95	**0.016**	0.94	**0.004**
#Emails			0.96	**0.011**	0.96	0.052
#Corrective			1.08	**<0.001**	1.07	**0.001**
#Features			1.08	**<0.001**	1.08	**<0.001**
#Perfective			1.00	0.963	0.98	0.453
#Non Functional			0.99	0.557	0.99	0.757
Test/Main Lines of Code Percentage					1.01	0.713
#Functions /w McCabe Index 11-25					1.10	**0.002**
#Functions /w McCabe Index >51					1.01	0.408
Very Large File Size Count					0.81	**<0.001**
Very Large Function Size Count					1.18	**<0.001**
Code Duplication Percentage					0.85	**<0.001**
Observations	3795		3795		3795	
R2 Tjur	0.116		0.185		0.239	

shown in Table 3. The random effect assumes a large portion of the variance in the full model (84.4% conditional vs. 15.4% marginal) as expected due to the oversized importance of programming language choice on all aspect of code, process, and quality.

We note that the results are qualitatively very similar as the results on only Java projects. This is not unexpected since 76% of the projects are Java projects, and thus the properties of the Java projects carry over. Still, the remaining projects do show significant congruence with Table 2. It did occur to us to contrast our metrics between functional and imperative languages. Unfortunately we did not have enough examples of projects using the former in order to fit a model successfully.

RQ_3 Summary: We find that an increase in the following increases the odds of graduation: lines of code, major and minor contributors, features commits, corrective commits, medium complexity (11-25 McCabe) functions, and very large functions. On the flip side, the increase in the following decreases the odds: top level directories, avg. files modified per commit, very large file sizes, and code duplication percentage.

6 DISCUSSION

We start with a caveat about regression and causality. When models are well fitted and confounds are accounted for, multiple regression can quantify directional effects (e.g., variable x on variable y) and thus, methodologically go beyond mere symmetric correlation. Hence the standard language used: "the effect of changing variable x is that variable y will change", etc. However, this is not to imply any strict causality relationship, temporal or otherwise. Moreover, quasi-experimental studies like ours that do not rely on randomized group assignment have in general lower discovery power than well designed randomized trials [25].

We found that the coefficients of some code quality and process measures are consistent across different projects, while some others are not. E.g., the number of major contributors are positively associated with project sustainability. It suggests that increased number of champions in technical contributions is a signal for project sustainability. As for the design implication, we suggest that OSS maintainers should help create a positive feedback loop for all contributors, giving more ownership and responsibility, while at the same time offering the flexibility to choose what features to work on, and contribute to the project's long-term vision [12]. We also find that the number of corrective commits are positive across projects, suggesting that the presence of explicit corrective message contained in commits may aid project sustainability. Such

Table 3: Summary of three GLMM models for all projects. The marginal R^2 is much smaller than for the Java-based models.

	Base		Base, Processes		Base, Processes, Quality	
Predictors	Odds Ratios	p	Odds Ratios	p	Odds Ratios	p
Intercept	0.69	0.491	21.31	**<0.001**	60.25	**<0.001**
SLOC	1.21	**<0.001**	0.93	0.129	0.98	0.759
#Directories	1.27	**<0.001**	1.22	**0.002**	1.15	**0.045**
#Top Level Directories	0.67	**<0.001**	0.63	**<0.001**	0.65	**<0.001**
Incubation Month	0.53	**<0.001**	0.73	**<0.001**	0.70	**<0.001**
#Major Contributors			2.15	**<0.001**	2.21	**<0.001**
#Minor Contributors			1.09	**<0.001**	1.09	**<0.001**
#New Contributors			0.98	0.269	0.98	0.264
#Files Added			1.00	0.764	1.01	0.675
#Files Deleted			0.96	**0.001**	0.96	**0.003**
Avg. Files Modified per Commit			0.93	**0.001**	0.92	**<0.001**
Active Days			1.35	**0.015**	1.30	**0.036**
#Emails			0.97	0.078	0.98	0.179
#Corrective			1.06	**0.001**	1.05	**0.005**
#Features			1.06	**<0.001**	1.06	**<0.001**
#Perfective			1.00	0.969	1.00	0.903
#Non Functional			1.03	0.138	1.03	0.159
Test/Main Lines of Code Percentage					1.04	0.079
#Functions /w McCabe Index 11-25					1.01	0.772
#Functions /w McCabe Index >51					1.01	0.446
Very Large File Size Count					0.90	**<0.001**
Very Large Function Size Count					1.19	**<0.001**
Code Duplication Percentage					0.87	**0.004**
Most Complex Function LOC					0.73	**<0.001**
Observations	4970		4970		4970	
Marginal R2 / Conditional R2	0.097 / 0.539		0.213 / 0.689		0.154 / 0.844	

result suggests corrective messages do not make the project look bad, they may instead signal a green light on the sustainability of the project: corrections are encouraged. Moreover, we found the average file modified per commit, among all process measures, has the most negative effect on project's sustainability. Along with the evidence that the number of files that contain more than 1000 LOC, and the function complexity being negative, we suggest that a commit should only contain the files that are required for that specific change, and to consider modularizing and refactoring large files and complex functions to reduce maintenance and development costs [3, 11, 35].

The fact that code duplication is negatively associated with sustainability makes sense: duplicated code is a pain to maintain and gives rise to technical debt [13, 24, 45]. Checking for duplicated code and specifically assigning resources to refactor such code clones should be prioritized at different stages throughout the project's evolution. If possible at all, avoiding the practice is good advice that will lower the maintenance efforts later on.

We were surprised to find that the number of very large functions increases the odds of graduation as very large functions are difficult to maintain. This can benefit from further study.

7 THREATS TO VALIDITY

Internal Validity. Collecting code quality data on projects involving a large number of different programming languages is difficult. Although we found no evidence of this in spite of spot checking, one internal threat to our study remains: that the Sokrates tool may not correctly collect and compute different metrics (e.g., complexity metrics are not computed uniformly across programming languages, due to the intricate differences between them). Likewise, the commit classifier might not be as accurate on our commit dataset as in the original study [15], even though it was trained on more than 5,000 commits yielding good accuracy. This is acceptable for our scope, as finding the best commit classifier is out of scope for this work. In our modeling we use the programming language as a random effect to control for any effects introduced by the programming language. While most of the projects have sufficient recorded data (e.g., commits), some repositories had no history (e.g., Guacamole), were an umbrella project for several projects (MyFaces), or had too few observations to make any meaningful conclusions. Thus, we excluded such projects and also eliminated those projects that have the majority of code written in Lua, Go, Erlang, CSS, C#, Kotlin, Autoconf due to being too few projects that use these languages, and thus having very few observations.

External validity. We carefully validated our hypotheses and answered our research questions based on the ASF Incubator data. Generalization beyond ASF projects might be difficult, due to the specific frame in which ASF incubating projects evolve. Any further generalization to projects beyond ASF must be made with care. Our aim was to provide evidence that these systems require complex analysis involving multiple perspectives, including the source code and the processes.

8 CONCLUSION

Motivated by prior work on OSS success, health, and sustainability, and driven by contingency theory, we hypothesized that there is a link between project graduation and code, process, and quality metrics of software in ASF incubator projects. In this paper, we presented the first study known to us associating code, process and quality with OSS sustainability. We find that while retired projects have a slightly higher cyclomatic complexity when adjusted for project size, their retirement does not seem to be associated with bugs, complexity or technical debt. We find that both major contributors and minor contributors (though less significant) play a positive role in increasing the sustainability of OSS projects. Among the process and quality factors, the file size, function size, function complexity seem to be most negative, suggesting keeping workflow simple and concise is of importance to sustainability. We consolidate those findings into takeaways for practitioners. Thinking ahead, we hope to generalize our model to projects outside Apache Software Foundation and develop tools for instrumenting general GitHub projects to help them on the trajectories to sustainability.

ACKNOWLEDGEMENT

We are grateful to the National Science Foundation for funding this project, under GCR Grant #2020751. We thank the FSE 2022 reviewers for their constructive comments.

REFERENCES

[1] Dimitrios Athanasiou, Ariadi Nugroho, Joost Visser, and Andy Zaidman. 2014. Test Code Quality and Its Relation to Issue Handling Performance. *IEEE Transactions on Software Engineering* 40, 11 (2014), 1100–1125. https://doi.org/10.1109/TSE.2014.2342227

[2] Robert Freed Bales. 2017. *Social Interaction Systems: Theory and Measurement*. Routledge. https://doi.org/10.4324/9781315129563

[3] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. 1993. Software Complexity and Maintenance Costs. *Commun. ACM* 36, 11 (nov 1993), 81–94. https://doi.org/10.1145/163359.163375

[4] Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2015. Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software* 67, 1 (2015), 1–48. https://doi.org/10.18637/jss.v067.i01

[5] Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2015. Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software* 67, 1 (2015), 1–48. https://doi.org/10.18637/jss.v067.i01

[6] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't Touch My Code! Examining the Effects of Ownership on Software Quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. 4–14. https://doi.org/10.1145/2025113.2025119

[7] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 334–344. https://doi.org/10.1109/ICSME.2016.31

[8] Jailton Coelho and Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *Proceedings of the 2017 11th Joint meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. 186–196. https://doi.org/10.1145/3106237.3106246

[9] Melvin E Conway. 1968. How do committees invent. *Datamation* 14, 4 (1968), 28–31.

[10] Kevin Crowston, Hala Annabi, and James Howison. 2003. Defining Open Source Software Project Success. (2003).

[11] Frank DeRemer and Hans H Kron. 1976. Programming-in-the-large versus programming-in-the-small. *IEEE Transactions on Software Engineering* 2 (1976), 80–86.

[12] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. 2021. What Makes a Great Maintainer of Open Source Projects?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 982–994. https://doi.org/10.1109/ICSE43902.2021.00093

[13] Georgios Digkas, Mircea Lungu, Alexander Chatzigeorgiou, and Paris Avgeriou. 2017. The Evolution of Technical Debt in the Apache Ecosystem. In *European Conference on Software Architecture (ECSA)*. Springer, 51–66. https://doi.org/10.1007/978-3-319-65831-5_4

[14] Lex Donaldson. 2001. *The Contingency Theory of Organizations*. Sage. https://doi.org/10.4135/9781452229249

[15] Geanderson E dos Santos and Eduardo Figueiredo. 2020. Commit Classification using Natural Language Processing: Experiments over Labeled Datasets.. In *CIbSE*. 110–123.

[16] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. 2015. Impact of Developer Turnover on Quality in Open-Source Software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 829–841. https://doi.org/10.1145/2786805.2786870

[17] Santiago Gala-Pérez, Gregorio Robles, Jesús M González-Barahona, and Israel Herraiz. 2013. Intensive Metrics for the Study of the Evolution of Open Source Projects: Case studies from Apache Software Foundation projects. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 159–168. https://doi.org/10.1109/MSR.2013.6624023

[18] Amir Hossein Ghapanchi. 2015. Predicting software future sustainability: A longitudinal perspective. *Information Systems* 49 (2015), 40–51. https://doi.org/10.1016/j.is.2014.10.005

[19] Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Jordi Cabot. 2017. An Empirical Study on the Maturity of the Eclipse Modeling Ecosystem. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 292–302. https://doi.org/10.1109/MODELS.2017.19

[20] Yue Jiang, Bojan Cukic, Tim Menzies, and Nick Bartlow. 2008. Comparing Design and Code Metrics for Software Quality Prediction. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE)*. 11–18. https://doi.org/10.1145/1370788.1370793

[21] Mitchell Joblin and Sven Apel. 2021. How Do Successful and Failed Projects Differ? A Socio-Technical Analysis. *ACM Trans. Softw. Eng. Methodol.* 31, 4, Article 67 (2021), 24 pages. https://doi.org/10.1145/3504003

[22] Cory J Kapser and Michael W Godfrey. 2008. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering* 13, 6 (2008), 645–692. https://doi.org/10.1007/s10664-008-9076-6

[23] Sang-Yong Tom Lee, Hee-Woong Kim, and Sumeet Gupta. 2009. Measuring open source software success. *Omega* 37, 2 (2009), 426–438. https://doi.org/10.1016/j.omega.2007.05.005

[24] Aversano Lerina and Laura Nardi. 2019. Investigating on the Impact of Software Clones on Technical Debt. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 108–112. https://doi.org/10.1109/TechDebt.2019.00029

[25] Tony Liu, Lyle Ungar, and Konrad Kording. 2021. Quantifying causality in data science with quasi-experiments. *Nature Computational Science* 1, 1 (2021), 24–32.

[26] Log4j Project. 2022. https://logging.apache.org/log4j/2.x/. Accessed: 2022-03-10.

[27] Daniel Lüdecke. 2021. *sjPlot: Data Visualization for Statistics in Social Science*. https://CRAN.R-project.org/package=sjPlot R package version 2.8.10.

[28] Daniel Lüdecke, Mattan S. Ben-Shachar, Indrajeet Patil, Philip Waggoner, and Dominique Makowski. 2021. performance: An R Package for Assessment, Comparison and Testing of Statistical Models. *Journal of Open Source Software* 6, 60 (2021), 3139. https://doi.org/10.21105/joss.03139

[29] Radu Marinescu. 2004. Detection Strategies: Metrics-Based Rules for Detecting Design Flaws. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 350–359. https://doi.org/10.1109/ICSM.2004.1357820

[30] Robert C Martin. 2009. *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

[31] Vishal Midha and Prashant Palvia. 2012. Factors affecting the success of Open Source Software. *Journal of Systems and Software* 85, 4 (2012), 895–905. https://doi.org/10.1016/j.jss.2011.11.010

[32] Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, and Anne-Francoise Le Meur. 2009. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering* 36, 1 (2009), 20–36. https://doi.org/10.1109/TSE.2009.50

[33] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *ACM/IEEE 30th International Conference on Software Engineering (ICSE)*. IEEE, 521–530. https://doi.org/10.1145/1368088.1368160

[34] Shinichi Nakagawa and Holger Schielzeth. 2013. A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution* 4, 2 (2013), 133–142.

[35] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. 2004. Where the Bugs Are. *ACM SIGSOFT Software Engineering Notes* 29, 4 (2004), 86–96. https://doi.org/10.1145/1013886.1007524

[36] Foyzur Rahman and Premkumar Devanbu. 2013. How, and Why, Process Metrics Are Better. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 432–441.

[37] Cobra Rahmani and Deepak Khazanchi. 2010. A Study on Defect Density of Open Source Software. In *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*. IEEE, 679–683. https://doi.org/10.1109/ICIS.2010.11

[38] Robert W Ruekert, Orville C Walker Jr, and Kenneth J Roering. 1985. The organization of marketing activities: a contingency theory of structure and performance. *Journal of marketing* 49, 1 (1985), 13–25.

[39] Alexander Sachs. 2019. *Predicting Repository Upkeep with Textual Personality Analysis*. Master's thesis. University of Waterloo.

[40] Charles M Schweik. 2013. Sustainability in open source software commons: Lessons learned from an empirical study of sourceforge projects. *Technology Innovation Management Review* 3, 1 (2013).

[41] Raed Shatnawi and Wei Li. 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *Journal of systems and software* 81, 11 (2008), 1868–1882. https://doi.org/10.1016/j.jss.2007.12.794

[42] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A Osborne. 2010. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Transactions on Software Engineering* 37, 6 (2010), 772–787. https://doi.org/10.1109/TSE.2010.81

[43] Chandrasekar Subramaniam, Ravi Sen, and Matthew L Nelson. 2009. Determinants of open source software project success: A longitudinal study. *Decision Support Systems* 46, 2 (2009), 576–585. https://doi.org/10.1016/j.dss.2008.10.005

[44] The MITRE Corporation. 2022. CVE-2021-44228. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228. Accessed: 2022-03-10.

[45] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An exploration of technical debt. *Journal of Systems and Software* 86, 6 (2013), 1498–1516. https://doi.org/10.1016/j.jss.2012.12.052

[46] Andrew H Van de Ven and Robert Drazin. 1984. *The concept of fit in contingency theory*. Technical Report. Minnesota Univ Minneapolis Strategic Management Research Center.

[47] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. 2016. The Sky Is Not the Limit: Multitasking Across GitHub Projects. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. 994–1005. https://doi.org/10.1145/2884781.2884875

[48] Carmine Vassallo, Fabio Palomba, Alberto Bacchelli, and Harald C. Gall. 2018. Continuous code quality: are we (really) doing that?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. 790–795. https://doi.org/10.1145/3238147.3240729

[49] Jing Wu, Khim-Yong Goh, and Qian Tang. 2007. Investigating Success of Open Source Software Projects: A Social Network Perspective. *ICIS 2007 Proceedings* (2007), 105.

[50] Tianpei Xia, Wei Fu, Rui Shu, and Tim Menzies. 2020. Predicting project health for open source projects (using the DECART hyperparameter optimizer). *arXiv preprint arXiv:2006.07240* (2020).

[51] Likang Yin, Mahasweta Chakraborty, Charles Schweik, Seth Frey, and Vladimir Filkov. 2022. Open Source Software Sustainability: Combining Institutional Analysis and Socio-Technical Networks. *Accepted at CSCW 2022, arXiv preprint arXiv:2203.03144* (2022). https://arxiv.org/pdf/2203.03144.pdf

[52] Likang Yin, Zhuangzhi Chen, Qi Xuan, and Vladimir Filkov. 2021. Sustainability Forecasting for Apache Incubator Projects. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 1056–1067. https://doi.org/10.1145/3468264.3468563

[53] Yang Zhang, Bogdan Vasilescu, Huaimin Wang, and Vladimir Filkov. 2018. One Size Does Not Fit All: An Empirical Study of Containerized Continuous Deployment Workflows. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 295–306. https://doi.org/10.1145/3236024.3236033