DOI: 10.15439/2023F5221
ISSN 2300-5963 ACSIS, Vol. 35

# On the Applicability of the Pareto Principle to Source-Code Growth in Open Source Projects

Korneliusz Szymański
Email: korneliusz.szymanski@gmail.com

Mirosław Ochodek
0000-0002-9103-717X
Poznan University of Technology,
ul. Piotrowo 2, 60-695 Poznan, Poland
Email: miroslaw.ochodek@put.poznan.pl

***Abstract*—Context: research on understanding the laws related to software- project evolution can indirectly impact the way we design software development processes, e.g., knowing the nature of the code-repository content growth could help us improve the ways we monitor the progress of OSS software development projects and predict their future development Goal: our aim is to empirically verify a hypothesis that the OSS code repositories grow in size according to the Pareto principle. Method: we collected and curated a sample of 31,343 OSS code repositories hosted on GitHub and analyzed their content growth over time to verify whether it follows the Pareto principle. Results: we observed that, on average, monotonically growing OSS repositories reach 75% of their final content size within the first 25% revisions. Conclusions: the content size of monotonically growing OSS repositories seems to grow in size according to the Pareto principle with the 75/25 ratio.**

## I. Introduction

THE 80/20 rule is often referred to as a means of quantitatively modeling cause-effect relationships between real-world variables. A generalization of this rule is a well-known Pareto principle. The principle states that *roughly 80% of outcomes come from 20% of causes*. This phenomenon has been also studied in the context of Open Source Software (OSS) development. Most of the studies investigated the principle by studying the patterns in the OSS community ways of working—i.e., commits [1], communication [2], [3], issue trackers, while others focused on the modeling the distribution of code smells, architecture, data, and software defects [4], [5].

This research focuses on studying the applicability of the Pareto principle to code repository content growth over the lifespan of OSS projects. In particular, our goal is to empirically verify a hypothesis stating that *monotonically growing OSS code repositories increase their content size over time according to the Pareto principle*, which means that, on average, an OSS code repository reaches 80% of its final content size in the first quantile of the project's lifespan. We narrow our study to projects that have a natural monotonic tendency to grow in size over time (e.g., they are not subjected to significant code removal activities).

Research on understanding the laws related to software-project evolution can indirectly impact the way we design software development processes. For instance, knowing the nature of the OSS code-repository contents growth could help us improve the ways we monitor the progress of OSS software development projects and predict their future development.

This work was supported by the Poznan University of Technology within the project 0311/SBAD/0738.

This paper is organized as follows. Section II provides more details on the Pareto principle and discussed selected studies on the Pareto principle in Software Engineering. Section III presents the design of our research method to study the applicability of the Pareto principle to the code-repository growth in OSS. Section IV presents and discusses the results and main threats to the validity of our study. The main findings are summarized in Section V.

## II. Background and Related Work

### A. The Pareto principle

The Pareto principle, also known as the 80/20 or 80 by 20 principle, was formulated in the early 1950s by Joseph Juran [6], but it was based on a relationship that Vilfredo Pareto had noted before—namely that 80% of the world's wealth is owned by 20% of humanity. Since then, numerous studies have shown that this principle holds also for other variables. However, this is not always an exact 80/20 ratio, the values are rather illustrative and will not apply to every situation, but in many cases, the Pareto principle works perfectly. The Pareto principle not only works in many fields, such as resource management, people management, and time management, but also in scientific fields, such as economics, accounting, medical sciences, or computer science.

The Pareto principle is also a generalization for the Pareto distribution presented in Figure 1. This principle is also characterized by the Pareto index, denoted as the alpha coefficient in the figure, which is an indicator of the Pareto principle strength.

### B. The Pareto principle in Software Engineering

A large number of OSS projects available on platforms such as GitHub or SourceForge created an opportunity for researchers to study a massive corpus of software projects. Free access to code repositories based on version control systems (VCS) makes it possible to study the evolution of projects over time. This includes numerous studies on the applicability of the Pareto principle to different areas covered by Software Engineering. Most of them focused on studying the applicability of the Pareto principle to model the distribution

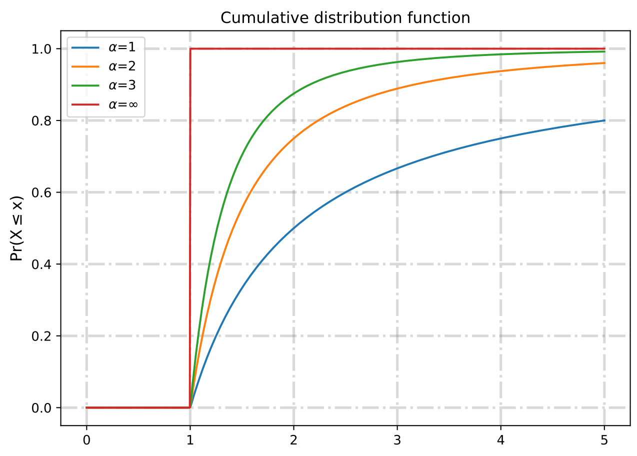


Fig. 1: Examples of cumulative density distribution functions for the Pareto distribution.

of defects, activity, or collaborators in software projects. Here, we briefly summarize the most relevant papers in this area:

- *Architecture smells and Pareto principle: A preliminary empirical exploration* [7] investigated to what extent architecture smell occurrences adhere to the Pareto principle. The authors analyzed 750 Java and 361 C# repositories and detected seven types of architecture smells. They reported that ca. 45% of the Java repositories and 66% of C# repositories followed the Pareto principle in this aspect. This study investigates the Pareto principle in open source, however, its focus is on architectural smells, while our study focuses on the size growth of OSS projects.
- *Long-term evaluation of technical debt in open-source software* [8] studied the evolution and characteristics of technical debt in OSS. In particular, the authors investigated the evolution of three large OSS Java applications (110 releases) with the use of the SonarQube technical debt detector. They reported that the Pareto principle was satisfied for the studied applications, as 20% of issue types generated around 80% of total technical debt. The focus of the study is different that ours, however, the main similarity is that both studies investigate release histories.
- *Towards a theoretical model for software growth* [9] studied 700,000 C source-code files of the FreeBSD operating system to compare different complexity measures in the context of measuring software growth. One of their observations was that all the measures followed a double Pareto distribution. This observation is convergent with the outcomes of our study, however, the original study did not consider the time-related aspect of code evolution.
- *Evidence for the Pareto principle in open source software activity* [3] focuses on analyzing the activity of users in mailing lists on a sample of three OSS software projects. This study is loosely related to our work since the studied object differs from ours.
- *On the central role of mailing lists in open source projects: an exploratory study* [2] regards communication in OSS projects and focuses on communication with the OSS projects' mailing lists. In particular, they empirically verified a hypothesis stating that a few key discussion participants are responsible for most of the messages posted on the mailing list. However, the study did not provide strong evidence confirming the applicability of the Pareto principle to this case.
- *Evaluation and application of bounded generalized Pareto analysis to fault distributions in open source software* [5] aimed at investigating distributions of faults in OSS software projects to see if it follows the Pareto distribution. Therefore, since the object of the study differs from ours, we consider it to be loosely related to our study.
- *Revisiting the applicability of the Pareto principle to core development teams in open source software projects* [1] studies the ratio of produced code size to developers' activity on a sample of 2,496 GitHub projects. This work is not directly related to our research since it focuses on the Pareto principle of code size in ratio to developers' activity and not code size growth over the project lifespan.
- *Empirical study of software quality evolution in open source projects using agile practices* [4] studies the evolution of software quality in two open source projects (Eclipse and Netbeans). They investigated the relationship between Object-Oriented metrics and defect proneness. They observed that a small percentage of compilation units hold most of the reported issues—i.e., it follows the Pareto principle.

As it follows from our literature analysis, most of the papers focus on studying the existence of the Pareto principle in different areas of software development, however, not in the context of repository content increments. Also, most of the studies focus on small samples of OSS projects and often have a form of case studies. On the contrary, our study is based on a massive dataset of OSS projects.

## III. Research Methodology

### A. Research goal, questions, and metrics

The goal of our study is to investigate a hypothesis that *monotonically growing OSS code repositories increase their content size over time according to the Pareto principle.* Since we aim to verify the hypothesis based on empirical evidence, we frame the problem using the Goal-Question-Metric framework [10]. Our goal (G) is to examine the applicability of the Pareto principle to OSS repository content size growth over the project lifespan. From the goal follows a research question (Q): **What is the average repository content size growth over time in OSS projects?**

Finally, we use the following metrics (M) as the basis for answering the research question:

- **M1: Repository contents size at a given point in time** — this metric presents repository contents size which is expressed by the total lines count of all files in the repository in the given revision. There is no distinction between the types of data that the files contain. Source code files, resources files, comments files, etc. are treated as contents files. We use the **wc** tool [11] to count the number of lines in the files. A point in time is determined based on the resolution defined as the M2 metric.

- **M2: Lifespan unit** defines a resolution unit of the project's lifespan. It is logically expressed as a % value but technically converted to a float in the range between 0 to 1. The lower the metric value, the higher the resolution. The base form of the Pareto principle is the 80/20 ratio. Therefore, it requires a maximum lifespan unit to be at the quintile level (20%). Based on the selected Lifespan unit, the project is sampled at unit-sized points in time, and the number of lines of code is calculated (M1).
- **M3: Repository final size** is the final size of the repository expressed in bytes. A too-small repository may not have visible differences in content growth. Conversely, a repository that is too large may indicate that it is a fixed-size dataset that does not change over time.
- **M4: Repository lifespan** is expressed as the number of commits. Repository lifespan could have a visible impact on the content growth analysis. A repository that has a too short commit history may bias the results since we analyze the distribution of the M1 measure in time. Unfortunately, in practice, some repositories may contain only a single commit. Also, this metric is related to the M2 metric. For instance, when M2 is set to 20%, the minimal repository lifespan (M4) is required to have minimum of 5 commits ($1/0.2 = 5$).

## B. Study design

Answering the research question requires a multi-stage procedure, starting from collecting a large corpus of OSS project repositories, analyzing how the code changes over time in these repositories, and drawing conclusions about the applicability of the Pareto principle to the distribution of code increments.

*1) Data acquisition and filtering:* In the first step, we collect a dataset of projects hosted on GitHub.[1] In particular, we use GitHub API to fetch the data from the projects. The service allows the database to be filtered for repositories that match the expected criteria, but the response output is limited to 1,000 rows. The sample of repositories could be too small for our use case. Also, the quality of projects hosted on GitHub could vary visibly, starting from what we would call "engineered" projects to some toy examples, code snippets, or even repositories that do not contain any code.

Our workaround to these issues is to use a curated list of 1,857,423 GitHub repositories created by Munaiah et al. [12] (RepoReapers[2]) and further curated by Pickerill et al. [13]. The filtered list contains GitHub repositories belonging to so-called "engineered" projects. Munaiah et al. define an engineered project as *"a software project that leverages sound software engineering practices in one or more of its dimensions such as documentation, testing, and project management."* Selecting such projects allow us to narrow the search and increase the relevancy of the obtained sample of OSS projects.

In the following step, we further filter the list of repositories based on the M2, M3, and M4 metrics to select only relevant code repositories. We do not pre-set any filtering thresholds for these measures, but instead, we determine them empirically by observing how making these criteria stricter influences the dataset properties.

Finally, we take into account projects whose content growth is incremental. Therefore, we use linear regression as a tool to identify projects with anomalies related to how their size grows over time. The linear regression is expressed as:

$$y = ax + b$$

where $a$ is a slope and $b$ is an intercept. The slope can be used to evaluate the repository-growth tendency. We are going to determine the $a$ threshold empirically by observing how it affects the presence/absence of outliers in our dataset. We use the PostgreSQL function regr_slope[3] to calculate the slope.

*2) Data analysis:* The analysis procedure is based on the Git version control tools. Git tools allow comparing changed, added, or deleted lines of code. An atomic set of changes recorded by Git is called a commit or revision. Git repositories consist of a tree of commits. Using the git-ls-tree command, the structure of the commit tree and its metadata can be obtained. The metadata contains information about changes to the file's lines. Using the wc [11] tool, it is possible to count the total amount of code at a specific point in time at which a commit was created.

The data is further normalized and aggregated to obtain the distribution of repository content growth. Each repository has a different project beginning and ending date. The duration of projects has to be expressed as normalized time. Let's define project duration normalized time as $NT$. $NT$ is a discrete variable with a distribution of every threshold $s$ (M2) which is expressed as a value from 0 which is the beginning date of a project and 1 which is the end date of a project.

$$NT = 0, s, 2s, 3s, ..., 1$$

Each project has a different content size at the end. This means a repository's content size also has to be expressed as a normalized value to a repository's final content size (M1). Let $L(p, NT)$ represents repository content size as a count of lines, where $p$ is the repository. The ending time then is equal to 1 ($NT = 1$).

$$L_{final}(p) = L(p, 1)$$

Having the final repository content size, the normalized repository content size at a given normalized point in time can be calculated:

$$NL(p, NT) = \frac{L(p, NT)}{L_{final}(p)}$$

[1]GitHub https://github.com

[2]https://reporeapers.github.io/

[3]https://www.postgresql.org/docs/9.0/functions-aggregate.html

Having projects lifespan as normalized points in time and normalized repositories content size makes it possible to calculate the distribution of repository content growth:

$$\overline{NL}(P, NT) = \frac{1}{n}\sum_{i=1}^{n} NL(P_i, NT))$$

where $P$ is set of repositories.

In order to ascertain the completeness of the Pareto principle in our study, the distribution of code increments should be close to the theoretical Pareto distribution. The relevant point of the distribution is the point of the searched 80/20 ratio. As a result of the study, the resulting graph may intersect the vicinity of the point, but the whole distribution may not converge to the Pareto distribution. The expected graph should be incremental with a significant increase in the first 20-30% and then a gentle increase up to 100%.

## IV. Results and Discussion

### A. Data fetching

We fetched code repositories from GitHub using the curated list of links to repositories belonging to so-called "engineered projects" [12], [13]. For each repository, we first downloaded the repository metadata from GitHub API. Next, each repository was downloaded and analyzed. We used the GitStats tool [14] to analyze the tree structure of Git commits. As a result, we collected data from **35,890** OSS software repositories.

The distribution of programming languages of projects is presented in Table I. This classification of projects is based on the GitHub metadata. The table presents the data for the programming languages used in over 100 projects. The distribution of the repository size for the code repositories in our sample is summarized in Table II. The average size of code repository is around 3,343 KB with a standard deviation of ca. 213 KB. The average size is influenced by very large repositories (see Figure 2), thus, the median size is visibly smaller and equal to ca. 180 KB.

TABLE I: The distribution of programming languages of projects

| **Language** | **Projects count** |
|---|---|
| Java | 8886 |
| Python | 5756 |
| Ruby | 4188 |
| PHP | 3313 |
| C++ | 3134 |
| C# | 2196 |
| C | 2153 |
| JavaScript | 633 |
| HTML | 380 |
| CSS | 188 |
| Objective-C | 124 |

TABLE II: A summary of code repository sizes (measured in KB).

| Average | Median | Standard deviation |
|---|---|---|
| 3,343.31KB | 180KB | 212.95KB |

### B. Calculating the measures

We used the GitStats and wc tools to analyze the repositories and calculate measures M1, M2, M3, and M4. An example of the output generated by the toolset for the Spring Boot repository[4] is as follows:

```
Project name
    spring-boot
Generated
    2022-08-13 00:02:29 (in 344 seconds)
Generator
    GitStats (version 55c5c28),
    git version 2.25.1, gnuplot 5.2
    patchlevel 8
Report Period
    2012-10-21 19:53:52 to 2022-08-12 17:47:38
Age
    3583 days, 3032 active days (84.62%)
Total Files
    8748
Total Lines of Code
    727599 (1726809 added, 999210 removed)
Total Commits
    39150 (average 12.9 commits per active day,
    10.9 per all days)
Authors
    1113 (average 35.2 commits per author)
```

In addition to the report, the toolset allowed us to collect information about the growth of the code repository in time. Figure 3 presents the growth of the Spring framework repository (M1) over time. As we can see, the size (M1) of this repository increases monotonically over time, therefore, it belongs to the population of the projects in the scope of our analysis.

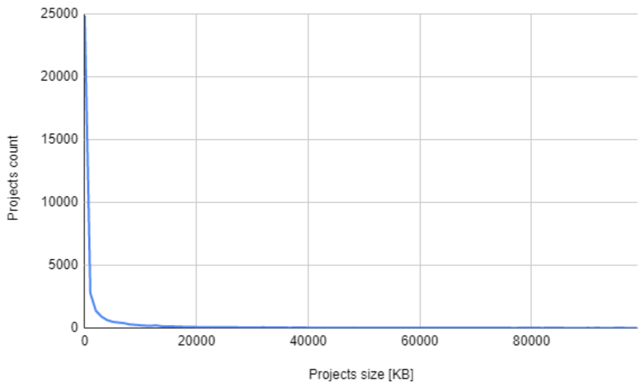
### C. Applying filtering criteria

We decided to set the project lifespan unit (M2) to $s = 0.05$, which allowed us to analyze the appearance of the Pareto principle with a minimum resolution of 5%. As it was stated in Section III-A, the maximum lifespan unit for studying the Pareto principle shall not exceed 20% ($s = 0.2$). We set the threshold to 0.05 to increase the resolution since it is rather unlikely to observe the exact 80/20 ratio in a sample of real-life data, but rather some minor variation of that ratio, e.g., 75/25, 70/30, or similar ratios.

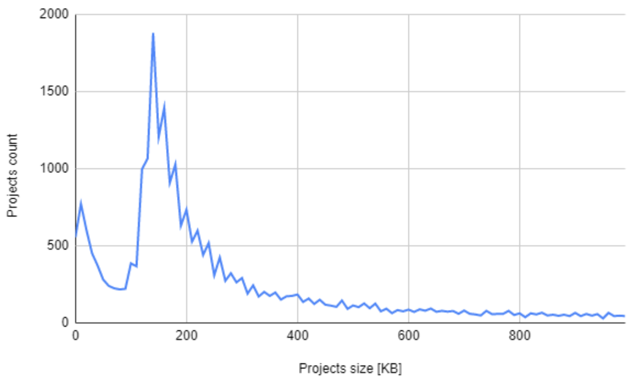
We set the following thresholds values of the M3 and M4 measures to initially filter the code repositories:

1) M3: Repository total size of up to 100 MB — we limited the size of the repository to exclude outliers (see Figure 2) and reduce the processing time of a single repository.

[4]Spring Boot – https://spring.io/projects/spring-boot



(a) All projects in the sample



(b) Projects with code repositories of up to 1,000KB

Fig. 2: The distribution of the sizes of the code repositories (KB).

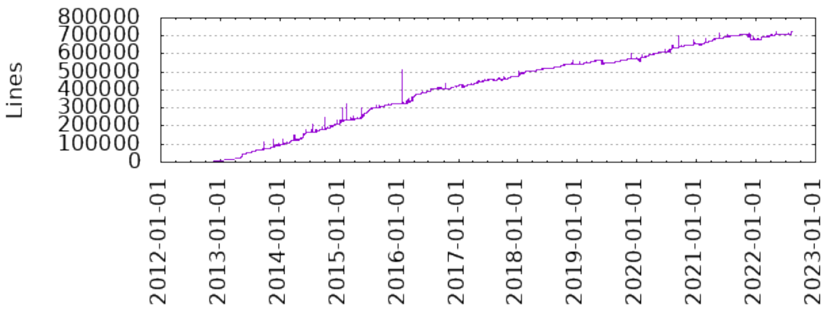


Fig. 3: An example of the repository growth plot for the Spring framework.

2) M4: Repository lifespan — we constraint the minimum repository lifespan to at least 20 commits, which follows from the selected lifespan unit (M2). A lifespan unit of 5% determines the minimum number of commits to 20.

In the following step, we used our toolset to calculate the count of lines (M1) over time ($L$). The results were stored in a PostgreSQL database. Next, we calculated the normalized repository content size ($NL$) and normalized project lifespan ($NT$). Finally, we used both measures to calculate the distribution of repository content growth ($\overline{NL}$).

Figure 4 shows $\overline{NL}$ for the sample of projects filtered based on the M3 and M4 measure thresholds. As the content growth is measured with respect to the final content size of the repository, the 100% content size is achieved at the last point in time. However, we can see that the average normalized size reaches up to 722% of the final size. The reason for that is the presence of outlier projects with visibly non-monotonic, non-incremental growth over time. An example of such a project could be Unity3d-Async-Task[5]:

```
Project name
    Unity3d-Async-Task
Generated
    2022-08-23 19:22:00 (in 0 seconds)
Generator
    GitStats (version 55c5c28),
    git version 2.25.1, gnuplot 5.2
    patchlevel 8
Report Period
    2015-03-07 19:24:45 to 2015-10-01 17:45:15
Age
    209 days, 21 active days (10.05%)
Total Files
    1
Total Lines of Code
    3 (48396 added, 48393 removed)
Total Commits
    58 (average 2.8 commits per active day,
    0.3 per all days)
Authors
    4 (average 14.5 commits per author)
```

[5]Unity3d-Async – https://github.com/NVentimiglia/Unity3d-Async-Task

As it is visible in Figure 5, the content of this repository was drastically reduced at a single point in time. As it stands from the README.md file, the source code from this repository was moved to another location. Such projects bias the general view of how the content of OSS code repositories grows over time and thus should be removed from the analysis. Unfortunately, manual filtering of such repositories was not feasible due to the size of the sample. Therefore, we decided to use linear regression (as we stated in Section III-B) to identify and remove outlying projects with respect to their

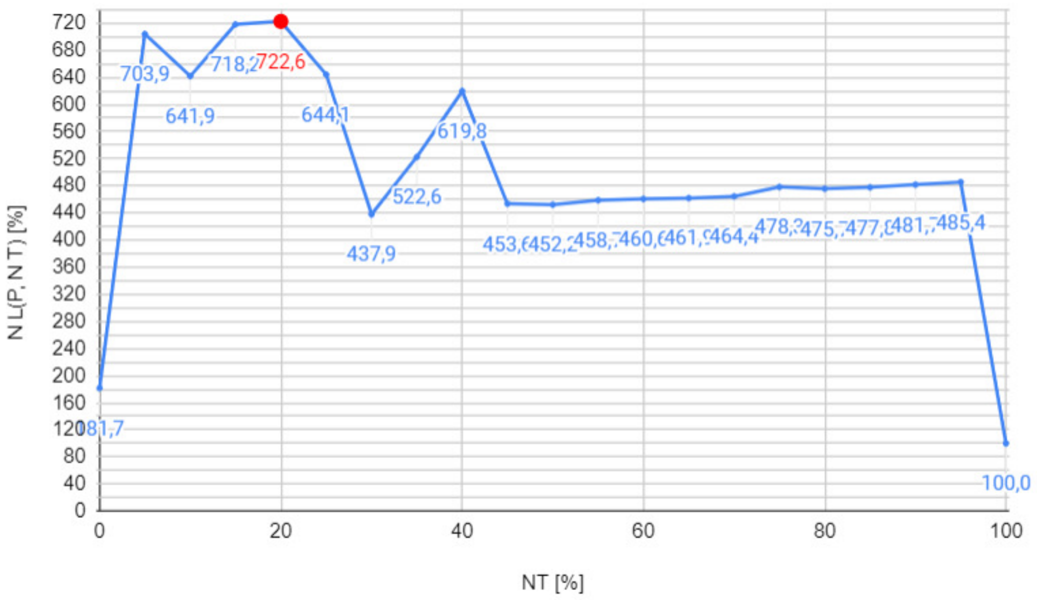


Fig. 4: Aggregated normalized lines count per normalized time expressed in percentage before filtering out the repositories with non-monotonic size growth over time.

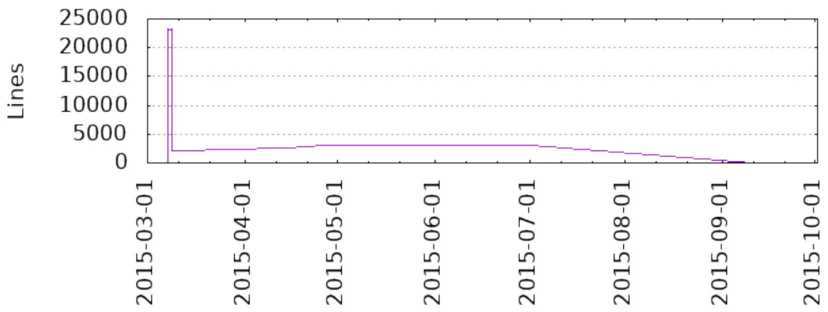


Fig. 5: Content growth in lines of code for Unity3d-Async-Task.

content growth. After applying linear regression and analyzing the slope of the regression functions, we accepted projects for which the slope was between 0 and 5. This allowed us to filter out projects with both suspiciously massive size reductions and suspicious bulk code addition operations in single commits.

The results of applying the filtering criteria are presented in Figure 4. The intensity of line-count growth became growing monotonically. Interestingly, as we can see in the figure (see $NT = 0\%$), projects, on average, are uploaded to GitHub when they contain 30% of their target size. After that, their content growth increases significantly, up to 20% of their lifespans when they reach 90% of their final size, and later, the growth rate flattens out. Unfortunately, we can also see a suspicious drop in size within the last 5% of the projects' lifespan, with a drop of 15% of their content (the pick point for a project content size is ca. 120% of its final size instead of the expected 100%). After investigation, it turned out to be, once again, the effect of outliers in the sample (e.g., moving repositories) that affected the overall picture. Therefore, we decided to introduce one more filtering criterion to identify projects with extensive code removal operations at the end of their lifespans and remove them from the sample. We applied linear regression to the last 10% of the projects' lifespans. Figure 7 shows the results of filtering the sample based on the slope of the regression function ($RE$). We decided to set the filtering threshold for the slope to $RE = -0.5$, however, as follows from the figure, the results were similar for all considered $RE$ values. The final sample included **31,343** out of the 35,890 initially fetched repositories.

## D. *The Pareto principle*

The analysis of the repository content growth for the curated sample of projects (see Figure 7) shows that, on average, monotonically growing OSS projects are uploaded to GitHub with 28.8% of their final content size. Later, their content

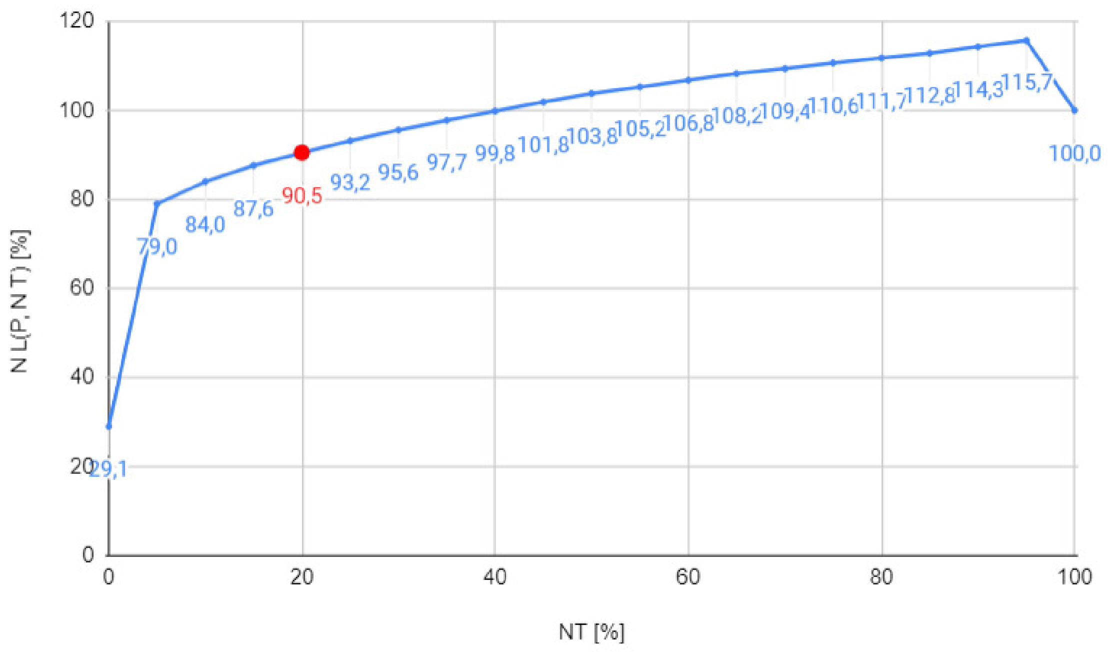


Fig. 6: Aggregated normalized lines count per normalized time expressed in percentage after filtering out the repositories with non-monotonic size growth over time.

size increases dynamically until it reaches ca. 60% of the final content size and over the remaining project lifespan, the repository content increases steadily.

The goal of our study was to investigate a hypothesis that repository content growth for monotonically growing OSS projects is subjected to the Pareto principle. If it is true, we would expect to observe that at the first quintile of the normalized project lifespan, the normalized content size of repositories should be around 80%. The ratio observed in the analyzed sample of projects is 75/25 (on average, 75% of the final repository content size was delivered within the first 25% of normalized project lifespan). Could we accept this observation as a manifestation of the Pareto principle?

As the literature on the Pareto principle suggests, the principle is not strictly bound to the 80/20 proportion. For instance, Dunford, Su, and Tamang [15] state that *"the Pareto Principle is a simplified version of the mathematics behind the Pareto distribution"* and *"the numbers 20 and 80 are not mathematically fixed, but are used as a rule of thumb"*; Boboia and Polinicencu [16] state that *"The 80/20 Rule is not a strict formula. Sometimes the cause-effect relation is closer to the 70/30 ratio than to the 80/20 ratio, but very rarely 50% of the causes lead to 50% of the results."* Therefore, taking into account the results of our study and literature regarding the Pareto principle, we conclude that **the observed 75/25 ratio fits the definition of the Pareto principle and allow us to accept the hypothesis that the content growth over time in monotonically growing OSS repositories is subjected to that principle.**

## *E. Threats to validity*

We identified several threats to validity of our study. The internal validity threats relate to the contents of repositories in the collected sample and their potential impact on the calculated measures, in particular:

- *Extreme sizes of repositories* — our sample may contain repositories that are too small or too large. The size of the repository may indicate inappropriate project content and purpose for our research. To reject such repositories, we used the filtering criteria M3 and M4.
- *Irrelevant repository content* — many of the repositories available on GitHub are toy examples, code snippets, or do not contain any source code. To mitigate the influence of such repositories, we used the curated list of GitHub repositories that are likely to contain "engineered" projects.
- *Non-monotonically growing repositories* — the content of repositories may not be incremental. We used linear regression models to automatically filter out projects with visible deviations in content growth from what we understand to be a monotonically growing project. However, since we were not able to manually inspect each and every project, we should accept the fact that some such projects could be still present in the curated sample.

The external validity regards the possibility of generalizing the results of our study to the whole considered population. First of all, we narrowed the population to monotonically growing OSS projects which gave us better control over the

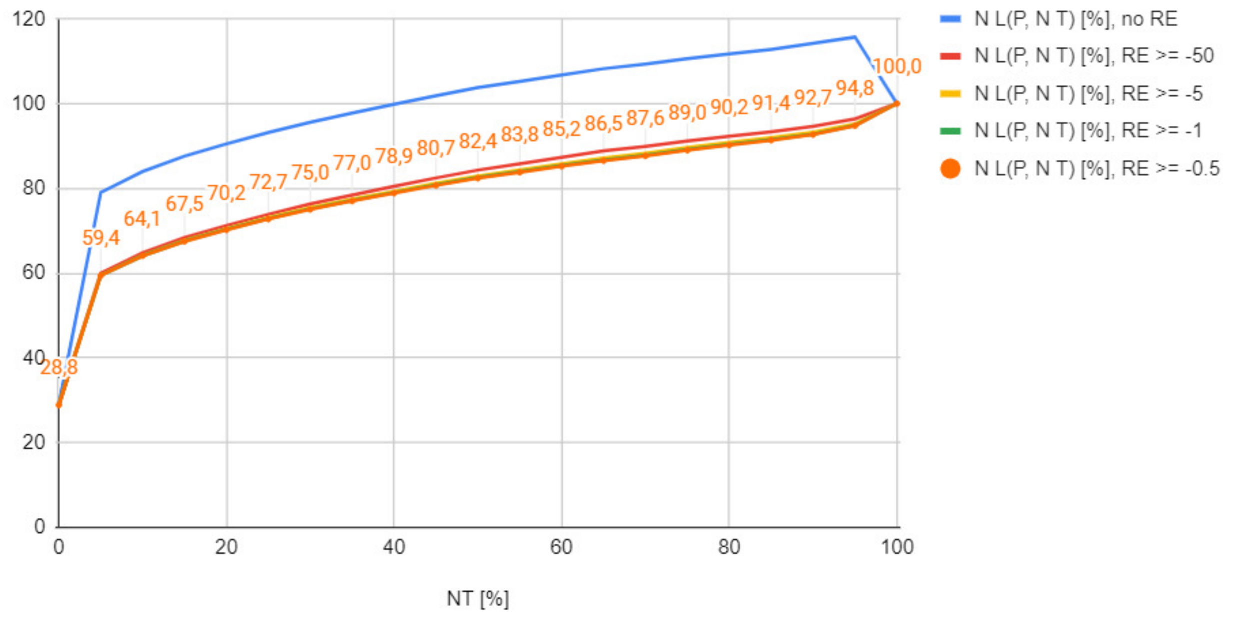


Fig. 7: Aggregated normalized lines count per normalized time expressed in percentage after filtering out the repositories with extensive code-removal actions at the end of their lifespans.

generalizability of our findings (but also narrows it). Secondly, we collected and curated a very large sample of OSS code repositories, which we believe to be a representative sample of the population under study.

## V. Conclusions

We analyzed the intensity of OSS repositories' content growth over time to verify a hypothesis that the number of lines in the monotonically growing repositories increases over the project lifespan according to the Pareto principle.

We studied a sample of 31,343 OSS repositories hosted on GitHub and observed that, on average, 75% of the final content size of the repositories is produced within the first 25% revisions in such repositories. Therefore, we claim that this phenomenon is a manifestation of the Pareto principle.

We also observed that monotonically growing OSS projects are, on average, become hosted on GitHub when they already contain 30% of their final code size[6]. Therefore, in many cases, it might be impossible to retrieve the history of code changes for the first ca. 30% of the project's final size. Although studying the causes of this phenomenon was outside of the scope of this study, we suspect that this might be caused by either beginning work on an OSS project as a private project/project hosted outside of the GitHub version control system, or due to the fact that many projects are based on framework skeletons or auto-generated stubs.

[6]Please note that we refer to the final size and not to the final content of the repository.

The observations made in this study could help to monitor the growth of OSS projects and help to evaluate the current state of such projects.

For future research, we recommend studying the applicability of the Pareto principle to content growth depending on the various sub-criteria such as programming language, community size, etc.

## References

[1] K. Yamashita, S. McIntosh, Y. Kamei, A. E. Hassan, and N. Ubayashi, "Revisiting the applicability of the pareto principle to core development teams in open source software projects," in *Proceedings of the 14th international workshop on principles of software evolution*, 2015, pp. 46–55.

[2] E. Shihab, N. Bettenburg, B. Adams, and A. E. Hassan, "On the central role of mailing lists in open source software projects: An exploratory study," in *New Frontiers in Artificial Intelligence: JSAI-isAI 2009 Workshops, LENLS, JURISIN, KCSD, LLLL, Tokyo, Japan, November 19-20, 2009, Revised Selected Papers 1*. Springer, 2009, pp. 91–103.

[3] M. Goeminne and T. Mens, "Evidence for the pareto principle in open source software activity," in *First International Workshop on Model-Driven Software Migration (MDSM 2011)*, 2011, p. 74.

[4] A. Murgia, G. Concas, S. Pinna, R. Tonelli, I. Turnu *et al.*, "Empirical study of software quality evolution in open source projects using agile practices," in *Proc. of the 1st International Symposium on Emerging Trends in Software Metrics*, vol. 11, 2009.

[5] C.-Y. Huang, C.-S. Kuo, and S.-P. Luan, "Evaluation and application of bounded generalized pareto analysis to fault distributions in open source software," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 309–319, 2013.

[6] J. M. Juran, "Pareto, Lorenz, Cournot, Bernoulli, Juran and others," in *Critical evaluations in business management*, J. C. Wood and W. M. C., Eds. Routledge, 2004, ch. 1, pp. 47–49.

[7] A.-M. Chaniotaki and T. Sharma, "Architecture smells and pareto principle: A preliminary empirical exploration," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 190–194.

[8] A.-J. Molnar and S. Motogna, "Long-term evaluation of technical debt in open-source software," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–9.

[9] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Towards a theoretical model for software growth," in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007).* IEEE, 2007, pp. 21–21.

[10] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (gqm) approach," *Encyclopedia of software engineering*, 2002.

[11] D. M. Paul Rubin, "word cound," https://linux.die.net/man/1/wc, lines word count tool, GNU General Public License.

[12] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Reporeapers," https://reporeapers.github.io/results/1.html, 2017, [Online; accessed 12-June-2022]".

[13] P. Pickerill, H. J. Jungen, M. Ochodek, M. Maćkowiak, and M. Staron, "Phantom: Curating github for engineered software projects using time-series clustering," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2897–2929, 2020.

[14] S. Byeon, "Gitstats, https://pypi.org/project/gitstats/."

[15] R. Dunford, Q. Su, and E. Tamang, "The pareto principle," *The Plymouth Student Scientist*, vol. 7, no. 1, pp. 140–148, 2014.

[16] A. Boboia and C. Polinicencu, "Application of the pareto analysis regarding the research on the value of preparations in community pharmacies from cluj-napoca, romania," *Farmacia*, vol. 60, no. 4, pp. 578–585, 2012.