DISSERTATION

TIME-DELTA METHOD FOR MEASURING SOFTWARE DEVELOPMENT CONTRIBUTION RATES

Submitted by

Vincil Chapman Bishop III

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2024

Doctoral Committee:

Advisor: Steven J. Simske

Marie Vans
Yashwant Malaiya
Indrajit Ray

v2024.07.24a

ABSTRACT

TIME-DELTA METHOD FOR MEASURING SOFTWARE DEVELOPMENT CONTRIBUTION RATES

The Time-Delta Method for estimating software development contribution rates provides insight into the efficiency and effectiveness of software developers. It proposes and evaluates a framework for assessing software development contribution and its rate (first derivative) based on Commit Time Delta (CTD) and software complexity metrics. The methodology relies on analyzing historical data from software repositories, employing statistical techniques to infer developer productivity and work patterns. The approach combines existing metrics like Cyclomatic Complexity with novel imputation techniques to estimate unobserved work durations, offering a practical tool for evaluating the engagement of software developers in a production setting. The findings suggest that this method can serve as a reliable estimator of development effort, with potential implications for optimizing software project management and resource allocation.

ACKNOWLEDGEMENTS

I would like to thank my mother and father for their hard work to provide me the academic foundation necessary to complete this research. I would also like to thank Dr. Steve Simske for giving me the courage to apply myself, and for all his valuable advice and support along the way. I would last like to thank the faculty and staff of the Systems Engineering Department at Colorado State University for fostering such a positive and constructive environment for all that endeavor to learn. Had it not been for the benevolence of those acknowledged, none of this would have been possible.

This research predates the author's employment with Amazon, Inc.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DEFINITION OF TERMS

Abbrev.	Term	Description
AC	Antecedent Commit	In a pair of commits by the same author, the earlier commit is referred to as the Antecedent Commit.
ACD	Antecedent Commit Date-time	The date-time value that the Antecedent Commit was created in the source code repository.
BC	Bound Contribution	Contribution that is preceded by an Antecedent Commit and likely represents continuous and uninterrupted work by a developer.
CA	Commit Author	The author of the commit.
CTD	Commit Time Delta	In a pair of commits by the same author, the difference between the antecedent commit and the subsequent commit measured in minutes. (SCD-ACD=CTD)
~CTD hours	Commit Time Delta Hours (rounded)	CTD seconds / (60 * 60) and rounded up/down to the nearest hour.
CD	Complexity Delta	A single change in Cyclomatic Complexity between one commit to the next.
C	Contribution	A measure that represents the size of a commit. Different methods can be used to calculate, e.g., Cyclomatic Complexity, Levenshtein distance, etc.
CR	Contribution Rate	The rate of change measured in words per minute (WPM) calculated as: (C÷CTD)
CRIM	Contribution Rate Imputation Method	A method that prescribes how mCR values can be "imputed" to commits with high CTD values. These imputed contribution rates can then be used to estimate the amount of hours a developer actually spent on the commit.

Abbrev.	Term	Description
LDBC	Levenshtein Distance Based Contribution	A measure of contribution that is based on the Levenshtein word distance between an old/new version of a file.
LDBCR	Levenshtein Distance Based Contribution Rate	A measure of contribution rate that is based on how much work distance between an old/new file can be generated over a period of time (words per minute, or hour).
LOC	Lines of Code	The count of lines of code in a software program or file. This can also be abbreviated SLOC for "Software Lines of Code".
MBCR	Mean Bound Contribution Rate	A method of estimating contribution rates based on an average of CR values taken from BC commits.
mCR	Model Contribution Rate	A Contribution Rate value used to estimate duration by (C / MCR) and applied to commits where the CTD is long in duration and does not represent continuous and Bound Contribution by the developer.
mCTD	Model CTD	A CTD period determined to contain natural CR values to be modeled for imputation.
NC	Natural Contribution	Contribution that was typed by a human as a part of the software development process.
NCR	NC Rate	A CR value observed below the maximum determined rate that a human programmer would reasonably write code.
SC	Subsequent Commit	In a pair of commits by the same author, the later commit is referred to as the Subsequent Commit.
SCD	Subsequent Commit Date-time	The date-time that the subsequent commit was created in the source code repository.

Abbrev.	Term	Description
UC	Unbound Contribution	Contribution that is not preceded immediately by an Antecedent commit. It's assumed that an Unbound Contribution is not worked on continuously, and therefore the CTD value of the commit does not represent the amount of time on which the commit was worked.
UnC	Unnatural Contribution	Contribution that was added by an automated process or otherwise not manually typed by a human.

CHAPTER 1: INTRODUCTION

In the evolving landscape of software development, the quantification of developer contributions represents a significant challenge, with implications for project management, team collaboration, and productivity assessment. Traditional metrics, while useful, often fail to capture the nuanced dynamics of software development processes. This study introduces the Time-Delta Method, a novel framework for estimating software development contribution rates that bridges the gap between existing theoretical models and practical application needs.

This research is motivated by the pressing demand for more accurate insights into software developer productivity and the effectiveness of their contributions. With the proliferation of complex software systems and the increasing reliance on collaborative development environments, the need for a robust, scalable, and accurate method to assess and optimize developer contributions has never been more critical. The Time-Delta Method responds to this need by leveraging Commit Time Delta (CTD) and software complexity metrics, offering a new lens through which software development efforts can be evaluated.

Grounded in the statistical analysis of historical data from software repositories, this methodology advances the field by providing a comprehensive framework that integrates Cyclomatic Complexity with innovative imputation techniques. This approach not only enhances our understanding of developer work patterns but also facilitates a deeper analysis of the factors influencing software project outcomes.

This study unfolds the theory, experimental methodology, and empirical validations of the Time-Delta Method. Through a detailed examination of software development activities

across both commercial and open-source environments, it demonstrates the method's efficacy in capturing the essence of developer contributions. The findings not only contribute to the academic discourse on software engineering metrics but also offer actionable insights for industry practitioners aiming to optimize project management strategies and resource allocation.

As the digital economy continues to expand, the implications of this research extend beyond the immediate domain of software engineering, offering potential applications in project management, team dynamics analysis, and the broader field of systems engineering. By providing a novel tool for assessing developer contributions, the Time-Delta Method represents a significant step forward in our quest to enhance the efficiency and effectiveness of software development practices.

CHAPTER 2: THEORY

This chapter explores the theoretical underpinnings of a novel methodology aimed at quantifying software development productivity through the analysis of Commit Time Delta (CTD) and complexity metrics. Theories proposed by this study are based on the ability to measure the size and time period associated with a software contribution (commit). The measure of size associated with a software contribution is termed “Contribution”. The time period measure used in this research is termed Commit Time Delta (CTD). With these two associated data points, a first derivative Contribution Rate (CR) can be estimated. The ability to measure a CR value forms the basis of other extrapolative techniques that allow for the durations of other less observed contributions to be estimated. The development of these theories support research goals by offering repeatable methods to estimate and impute CR values to contributions. These imputed CR values can eventually be used to estimate the actual time a developer spent on a contribution. The following sections propose theories that will be explored in this study.

2.1 Time-Delta Method

The foundational theory of the dissertation, the Time-Delta Method, is designed to estimate software development contribution rates. This method utilizes Commit Time Delta (CTD) alongside software complexity metrics to assess developer productivity and work patterns. By analyzing software repository data through statistical methods, the Time-Delta Method aims to provide a reliable estimation of development efforts.

2.2 Contribution

The concept of contribution is dissected into several methodologies to quantify the impact of a developer's work. The study explores traditional measures like Levenshtein Distance and introduces Cyclomatic Complexity-based contribution as a preferred method. Chapter 5: Contribution evaluates various techniques for measuring contribution, ultimately advocating for complexity-based measures due to their ability to reflect developer behavior more accurately.

2.3 Contribution Rate

The study explores the calculation of the Contribution Rate (CR), presenting it as the rate of change in contribution over time. The study compares Levenshtein Distance-based and Complexity-based CRs, favoring the latter for its stability and reliability. The Contribution Rate is essential for understanding developer productivity in relation to time spent on tasks.

2.4 Statistical Significance

The study assesses the statistical significance of contribution rates within software development repositories. Through statistical testing, experiments identify patterns and significance in CR values, affirming the Time-Delta Method's utility in reflecting real-world developer behavior.

2.5 Contribution Rate Imputation Method

The study introduces methods to estimate the contribution rates for commits with long CTDs, where direct measurement is challenging or impossible. The study outlines a hypothesis for Model Contribution Rates (mCR) and explores techniques like Gradient Boosted Tree

Regression and Mean Bound Contribution Rate (MBCR) for imputation. The study focuses on devising a practical approach to assign reasonable contribution rates to such commits.

2.6 Evaluating Imputed Contribution Rates

The final theory addresses the evaluation of imputed contribution rates, emphasizing the importance of sanity checking and statistical analysis. The study presents a methodology for assessing the plausibility and accuracy of estimated contribution rates, ensuring the reliability of imputed values.

CHAPTER 3: METHODOLOGY

The study acknowledges that it may not be possible to determine precisely what went on over long periods of time when human software developers were unobserved. To deliver value in the face of this reality, the research methodology takes an approach of estimation instead of precise measurement, indirect evidence instead of verifiable proof. Despite these approximations, it is still possible for these methods to deliver business value, as precise measurement is not required in all cases for feedback to be valuable.

3.1 Experimental Approach

- A time series duration is assigned to commit contributions by developers.
- Contribution is quantified using two methods: Levenshtein Distance and cyclomatic complexity.
- Where possible, a rate is estimated using the time series and contribution information.
- Behavior between the Levenshtein and complexity-based contribution rates are compared, relationships are inferred.
- Model contribution rates are sampled, and imputed to commits that were likely worked on over a long period of time: days, weeks, or longer.
- These imputed contribution rates are then multiplied over the contribution measures to estimate how long a developer may have spent on a task.
- These estimations are then sanity checked against the plausibility that an individual might have worked these hours.
- These results are then used to support indirectly the validity of the contribution rate measures.

3.2 Experimental Dataset

Table 1: Experimental Dataset Details

repo_code	MIN(commit_date)	MAX(commit_date)	COUNT_DISTINCT(author_email)	COUNT_DISTINCT(commit_sha)	SUM(git_patch_count)	SUM(git_hunk_count)
commercial-repo-1	2022-11-28	2023-12-04	295	8k	31.1k	89.7k
commercial-repo-2	2021-12-06	2023-12-04	53	2.17k	12.1k	49k
commercial-repo-3	2022-02-21	2023-11-22	167	8k	34.8k	152k
commercial-repo-4	2021-11-22	2023-11-22	128	2.47k	8.48k	28.8k
commercial-repo-5	2022-08-30	2023-11-22	72	3.47k	14.8k	53.2k
open-source-repo-1	2023-06-13	2024-01-27	224	8k	30.6k	122k
open-source-repo-2	2022-01-26	2024-01-26	29	4.76k	19.1k	103k
open-source-repo-3	2022-01-27	2024-01-27	547	7.55k	29.8k	105k
open-source-repo-4	2022-11-23	2024-01-27	313	8k	61.4k	232k
open-source-repo-5	2022-12-15	2024-01-27	162	8k	50k	211k
Totals	**2021-11-22**	**2024-01-27**	**1.8k**	**60.4k**	**292k**	**1.15M**

- Commercial and open-source repositories were chosen to highlight possible differences in contribution patterns.
- Repository details have been de-identified for anonymization purposes.
- The study began with a target of a 730-day (~2 years) history for each repository. Certain repositories could not finish processing 730 days' worth of commits in a reasonable amount of time. In these cases, the repositories were limited to 8,000 commits.
- The 730-day history started from the latest commit. The latest commit may have taken place on different days; therefore, the min/max commit dates vary between repositories.
- Commits were examined from a single default branch.

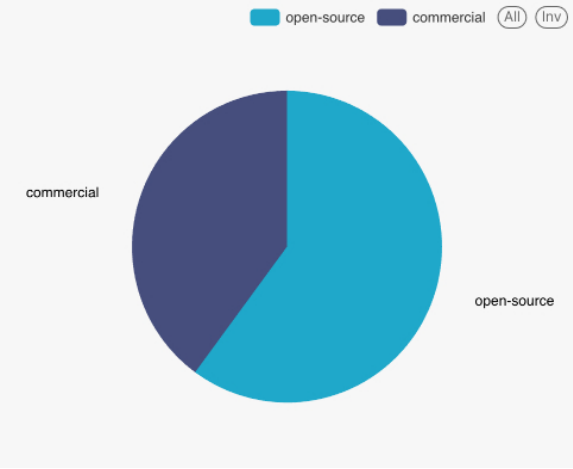

Figure 1: Commercial vs. Open-Source Commits

Figure 1: Commercial vs. Open-Source Commits depicts the proportion of open-source (36,305, ~60%) to commercial (24,104, ~40%) commits in the dataset for a total of 60,409 commits in total.

3.3 Experimental Environment

During research, a technology stack emerged that achieved success in extracting and processing the dataset. Challenges included slow retrieval times from the file-based Git database, as well as over 1 million individual objects that had to be processed. The file-based nature of the Git database presented performance challenges because of disk wait times. The large number of commit, patch, and diff hunk objects that must be extracted from Git require a scalable strategy for processing. Big data tooling was required to process the dataset as it quickly grew to a size that caused out of memory exceptions while processing. This section discusses a few notable tools that were used.

3.3.1 Apache Spark

Apache Spark [1] was used to process data at scale. In the beginning stages of research, commodity application servers were used to process data, and this worked well until exploration necessitated examining repository data at the DiffHunk[1] level. Processing quickly exhibited out of memory exceptions due to the size of the datasets pulled into memory. After some amount of experimentation Apache Spark, specifically Amazon's EMR Serverless product was used to process the datasets. Apache Spark excels at processing large data sets because it can distribute data across a cluster of worker nodes using a relatively familiar API interface. Amazon's EMR

[1] See: https://www.pygit2.org/diff.html#pygit2.DiffHunk

Serverless [2] product conveniently allowed processing to scale using low-cost virtual servers far beyond what the project would have been able to achieve relying on physical servers. Within the Apache Spark ecosystem, PySpark [3] libraries specifically were used to map/reduce the large dataset.

3.3.2 PyGit2

To extract low level objects from Git [4] source control repositories, the PyGit2 [5] library was used. PyGit2 is based on the LibGit2 [6] library, and offers a convenient Python based interface to interact with Git objects. It should be noted that LibGit2, and therefore PyGit2 does not maintain feature parity with the command line version of Git present on most systems. The command line version of Git is usually more advanced in terms of features and functionality when compared to LibGit2 or PyGit2.

3.3.3 Jupyter Notebooks

Python Jupyter [7] notebooks were used as a general programming interface when developing experimental code to process the data set. Notebooks were used not only to develop code that was eventually promoted to EMR/Spark, but also for experimentation and proof of concept for untested approaches.

3.3.4 Postgres SQL

The project experimented with several different data stores including Amazon's S3 with SQL queries provided by Amazon Athena, Amazon Redshift, Apache Druid and others [8] [9] [10] [11]. The experimentation started with PostgreSQL [12] as one of the first data stores, and in the end returned to the technology as a primary data store because of its ease of compatibility with advanced analytic tools such as Tableau and Apache Superset [13] [14]. The NoSQL nature

of S3, Druid, and other products from a development perspective was very attractive as hard relationships and predefined schemas don't have to be adhered to in many cases. This reduces development overhead and encourages velocity when it comes to delivery. Overall, it turned out to be less work to absorb the overhead of PostgreSQL, migrations, and a relational schema to get the project to a place where advanced analytics and visualizations could be easily achieved using readily available and mature tools. While it is technically possible to use tools like Tableau, Superset, or even Jupyter notebooks to interface with NoSQL based tooling, the solutions required excessive mechanisms, adapters, or other technology that made the integration laborious, error prone, and distracting from primary goals.

3.3.5 Apache Superset

Apache Superset was eventually settled on for project data visualization. A variety of tools including the commercial Tableau business intelligence package, as well as Matplotlib [15] and Jupyter notebooks was used along the way. In the end Apache Superset proved to deliver high quality visualizations and offered a mature user interface and feature set. Some users may find Superset more approachable and straightforward to use than Tableau. It is worth noting that Tableau does seem to have more features than Superset. For this project it was possible to work around missing features and still deliver complete and meaningful visualizations.

CHAPTER 4: THE TIME-DELTA METHOD

The Time-Delta method is a technique used to determine the time elapsed since the last commit made by a specific author. The technique assigns a duration to a commit by measuring the amount of time that has passed since the author's prior commit. The Time-Delta method can be used to estimate the amount of time that an author may have worked on a task associated with a commit, in the case that the duration between commits is short enough. Conversely, the method is also useful to identify commits where the author probably did not work continuously on a task.

Central to the Time-Delta method is the Commit Time Delta or CTD. CTD is the amount of time since the last commit by the same author in the same repository. Kolassa et al. has called this measure the “commit interval” and emphasized its use for characterizing developer behavior. The same study also identified that different individuals may exhibit different commit patterns depending on their position on the software team as some individuals seem to have a higher commit frequency than others [16]. Gote et al. references the commit timestamp not only to track an author’s activity, but also as a way to model the interactions and relationships between authors and their contributions [17].

It is important to note that the Time-Delta method is not always precise. This is because not all commits will have an Antecedent Commit by the same author that is necessary for calculation. Therefore, this technique may not always provide an accurate representation of the duration between two specific commits. Nonetheless, it is still a useful tool for downstream calculations when trying to estimate how long a developer may have spent on a task. The method also shows promise as an individual efficiency tool as the index of a developer’s performance against herself would be a particularly relevant comparison.

4.1 Calculation

CTD in a repository is calculated by these steps:

1. Group commits by author.
2. Starting with the latest commit by an author, the amount of time in seconds since the last commit by the same author (Antecedent Commit) is measured.
    a. If there is no Antecedent Commit, then the CTD is None/null.
3. The CTD value, if any, is associated with the commit.

4.2 Distribution

4.2.1 CTD Weekly Distribution

To visualize the distribution of CTD values at a macro level, rounded CTD values are converted to week buckets (168 hours, rounded up/down to the nearest week). The x-axis is the number of weeks rounded; the y-axis is the count of unique commits with CTD values falling within the corresponding week. The visualization was limited to approximately 8 weeks or less somewhat arbitrarily in order to illustrate the right skewed [18] nature of the distribution.

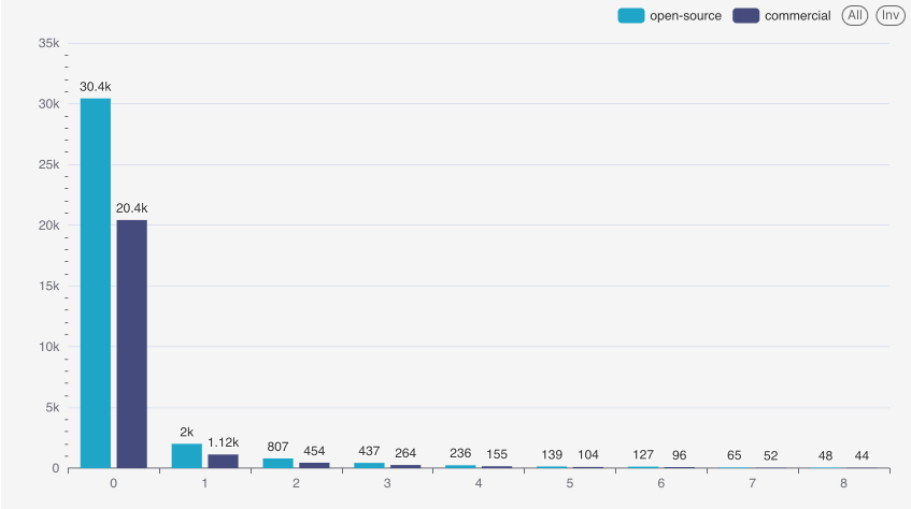

Figure 2: CTD Weeks Bucket Distribution

Figure 2: CTD Weeks Bucket Distribution shows that most commits (approximately 50,891 out of 60,409 in total, ~84%) have a CTD value of less than one week. The distribution of CTD values was proportionally similar across open source and commercial repositories.

4.2.2 CTD Daily Distribution

Continuing the examination of CTD value distribution, commits were grouped into days by rounding the CTD values up/down to the nearest 24 hours. Figure 3: CTD Days Bucket Distribution depicts the count of unique commits grouped by day CTD values, less than 7 days.

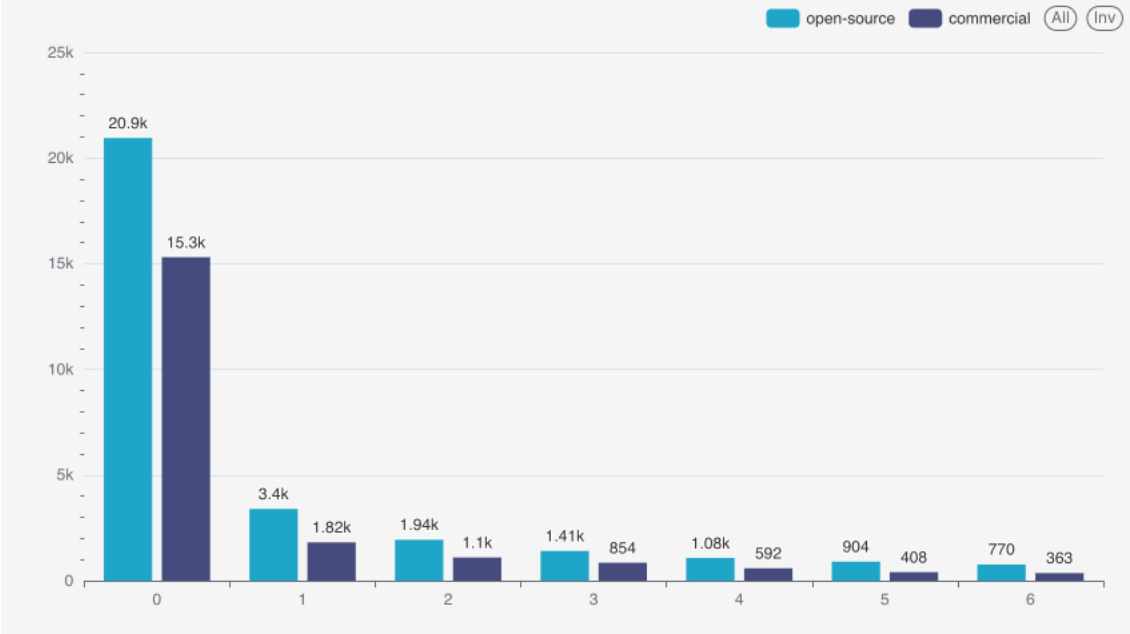

Figure 3: CTD Days Bucket Distribution

In Figure 3: CTD Days Bucket Distribution it can be observed that a significant number of commits (37,386 out of 60,409 in total, ~62%) have a CTD value of less than 24 hours. This CTD distribution would indicate that most commits are started and completed in a single day.

4.2.3 CTD Hourly Distribution

Understanding that a significant number of commits have a CTD value of approximately 24 hours or less, CTD distribution is visualized hourly.

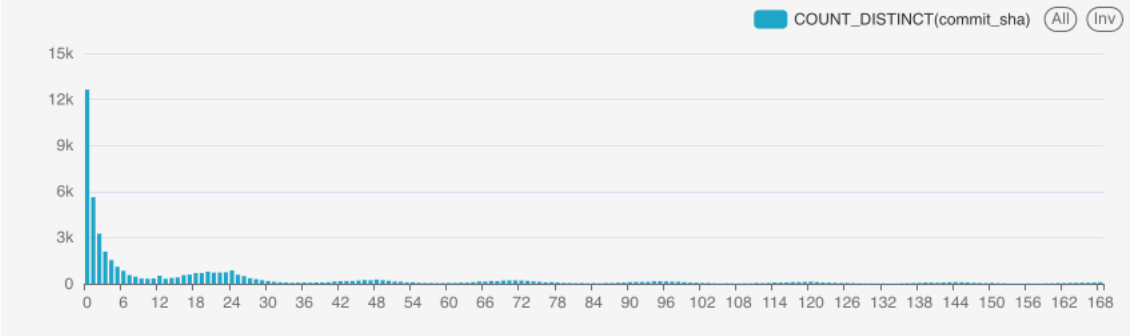

Figure 4: CTD Hours Bucket Distribution

Figure 4: CTD Hours Bucket Distribution is shared to demonstrate the hourly distribution of commits across an entire week (< 168 hours, rounded to the nearest hour) further reinforcing the right-skewed nature of the dataset.

Figure 5: CTD < 8 Hours Bucket Distribution shows the distribution of open source vs. commercial commits across an 8-hour period, rounded up/down to the nearest hour. 8 hours is a significant period because for many software developers it represents a normal workday.

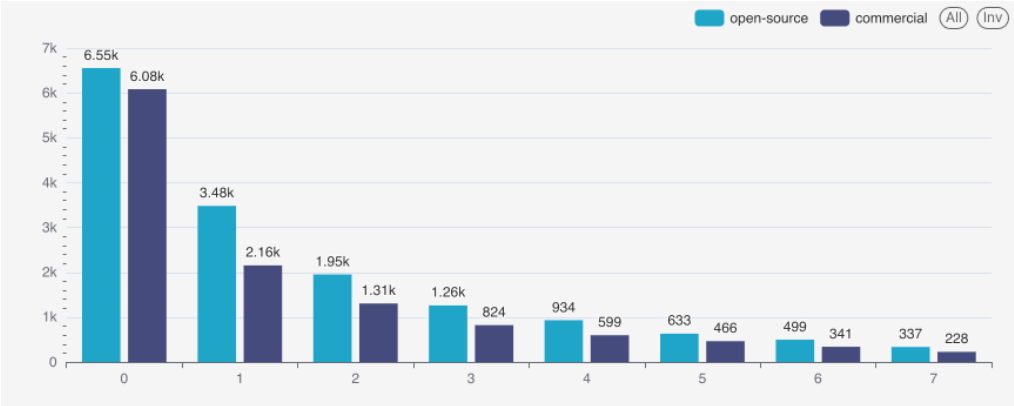

Figure 5: CTD < 8 Hours Bucket Distribution

Figure 5: CTD < 8 Hours Bucket Distribution indicates that almost half of all commits (28,605 out of 60,409 in total, ~47%) have a CTD value of less than 8 hours. Similarly, 25,180 out of 60,409 in total (~42%) have a CTD value of 4 hours or less. 16,725 commits out of 60,409 in total (~28%) have a CTD value less than 1 hour.

4.2.4 CTD Quarter Hour Distribution

Figure 6: CTD < 1 Hour in 15 Minute Buckets illustrates the distribution of CTD values over the first hour CTD window.

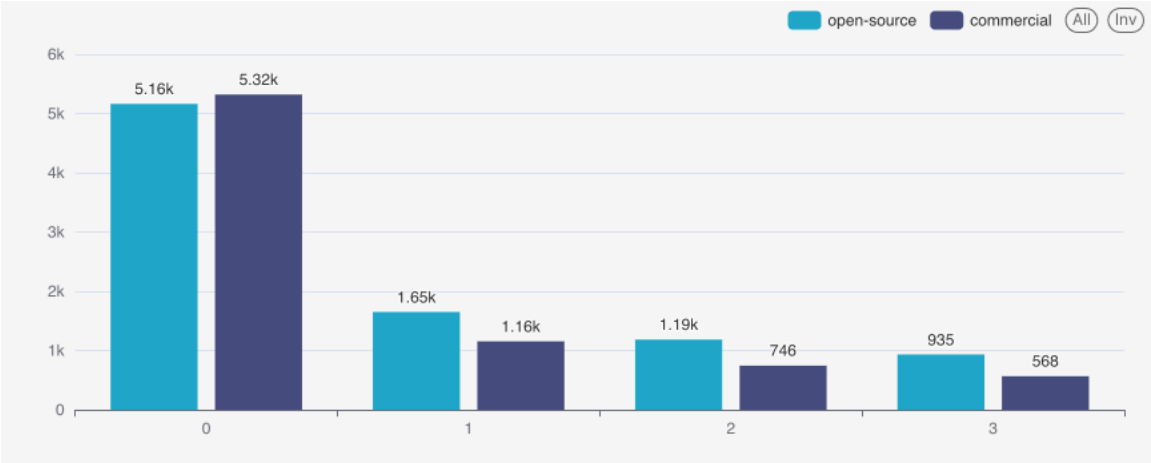

Figure 6: CTD < 1 Hour in 15 Minute Buckets

10,482 out of 60,409 in total (~17%) have a CTD value of 15 minutes or less. It can be noted that a greater number of commits from commercial repositories were made within the first 15-minute CTD window. This is interesting because commercial commits make up a minority in the data set population, which might indicate that developers in a commercial setting are more likely to perform quick remedy commits, shortly after an antecedent commit was performed. The remainder of the 6,239 commits are distributed over the other 3 quartiles.

4.3 Issues

Some issues with the method are that not all commits have a CTD value. This can lead to null values when using the method for calculations. Missing CTD values cause downstream contribution rates to not be calculated, further contributing to the approximate nature of dependent calculations. An opportunity for future improvement could impute CTD values to commits with missing values.

4.4 Conclusions

There seems to be a definite bias towards the first “0” hour CTD bucket. This is explained by quick remedy commit, with focused additions of logic and less time spent idle with more time spent generating textual change. While not as profound as this bias for that first “0” hour CTD bucket, later hour buckets see a continual decline in the number of commits. This trend was observed across open source as well as commercial repositories. In conclusion, this data set indicates there is a bias for shorter CTD values. It's likely that this behavior would also be normally observed in most active repositories.

CHAPTER 5: CONTRIBUTION

In the context of a software source code repository, contribution is a term used to describe the size or impact of a particular commit. In software development, a commit is an atomic set of changes made to the codebase which are then saved and tracked as a new version. Contribution quantifies the size of these changes and can be measured in several different ways.

5.1 Research Evolution

This development of contribution methods was a challenging and complex undertaking that required the utilization of a variety of techniques progress. Throughout the course of the investigation, numerous obstacles, setbacks, and technical challenges were encountered. A range of contribution approaches were experimented with, each with its own strengths and weaknesses. Some of these techniques proved successful, allowing significant strides towards the goals to be made. Others were less effective, requiring reevaluation and repeated tries. During research into contribution quantification methods, three main approaches were explored. The first was Levenshtein-Based Contribution, which involves calculating the difference between the old and new versions of a file using the Levenshtein distance algorithm. The resulting value represents the size of the changes made, with higher values indicating larger contributions.

The second method investigated was LLM Logic Score-Based Contribution, which considers not only the size of the changes made, but also their complexity and potential impact on the codebase.

Finally, Complexity-Based Contribution was examined, which focuses on the inherent complexity of the code being changed. This method assigns a score based on the complexity of

the code before and after the commit, with higher scores indicating more significant contributions. The Complexity-Based method was ultimately chosen as the preferred method for measuring contribution for experiments because its stability and results that more reasonably reflect a developer's potential behavior. The following sections provide an overview of some of the approaches explored, both successful and unsuccessful.

5.1.1 Levenshtein Distance Base Contribution

Experimentation was performed with a Levenshtein[2]-based method of measuring software contribution. This involved assessing contribution by calculating the distance between the words used in the codebase. Although this approach did not yield the expected results, it provided valuable insights that helped refine the approach.

It is worth noting that Levenshtein distance is a useful technique for measuring changes in source code text files. However, one of its drawbacks is that it cannot differentiate between a logic change and a simple renaming or refactoring; in other words, the type of cosmetic changes that someone trying to "game" the system might perform. As a result, significant fluctuations were observed in the derived contribution rate when this method was solely used to measure contribution.

[2] Levenshtein Distance: https://en.wikipedia.org/wiki/Levenshtein_distance

5.1.2 LLM Logic Score Based Contribution

A Large Language Model (LLM[3]) was employed to evaluate source code logic. This method involved providing details of file changes and using generative AI[4] to analyze the change in logic. Although at first this method showed a lot of promise, roadblocks were encountered, and research could not continue with this approach.

Initially, the LLM could differentiate between innocuous refactoring, white space changes, and actual logic changes. However, this approach proved unfeasible for a few reasons. First, the number of inferences required to analyze a codebase for any meaningful period easily reached into the millions, causing significant costs. Second, analysis took a long time because of the number of inferences required (one per file change). Although current cloud based generative AI services offer batch inference interfaces allowing many requests to be sent at once, the experiments could not achieve enough successful inferences to form a large enough dataset for meaningful analysis. Last, it was difficult to get the LLM model to differentiate and quantify many types of logic changes, further limiting the quality of results.

Due to these limitations, the generative AI-based contribution approach was abandoned. Although not suitable for this round of research, evaluation of source code by generative AI is an area of application that shows a lot of promise.

---

[3] Large Language Model: https://en.wikipedia.org/wiki/Large_language_model

[4] Generative Artificial Intelligence: https://en.wikipedia.org/wiki/Large_language_model

5.1.3 Cyclomatic Complexity Based Contribution

The technique that was ultimately settled on was a technique based on the measurement of contribution using cyclomatic complexity[5] deltas. This approach involved measuring the cyclomatic complexity of a file and its methods before/after a change to calculate a cyclomatic complexity delta. Despite encountering initial difficulties, progress was eventually made, and the study achieved a stable measure that reliably reflected a developer's contribution. The cyclomatic complexity-based contribution approach is heuristic in nature and sits somewhere between the wholesale Levenshtein distance-based approach, and the ultra-intelligent but non-performant LLM approach.

5.2 Performance Limitations

The methods used in this research filter certain outliers that may affect the processing of commits in a repository. One cause of these outliers was found to be the branching strategy used by the development team. In certain scenarios, when the team collects and merges commits to a branch and then subsequently merges that branch to the main branch, it can lead to unnaturally large commits. Such large commits can cause out-of-memory errors and failures during processing. To address this issue, commit files having an excess (>200) number of changes are omitted from processing. This avoids some errors and ensures that processing stays performant.

[5] Cyclomatic Complexity: https://en.wikipedia.org/wiki/Cyclomatic_complexity

5.3 Candidate Methods

While not all methods attempted delivered the desired performance, these experiences are included here for background and reference.

5.3.1 Levenshtein Based Contribution

Levenshtein Based Contribution is calculated by measuring the Levenshtein Distance [19] [20] [21] in words (whitespace excluded) between the before and after versions of a commit file. Levenshtein Distance has some advantages over traditional lines of code measures as it provides a higher fidelity of measure than assuming all lines contain equal change. The notion to use a Levenshtein distance to measure changes in source code is not original, Gote et al. (2019) [17] has also proposed use of the metric in their experimentation.

Calculation Method

Levenshtein Based Contribution is calculated by measuring the Levenshtein Distance in words between the original and changed text in a commit's DiffHunks. The Levenshtein Distance between each hunk is taken because it is computationally less intensive to perform the calculation on a small block of change as opposed to an entire file. The Levenshtein Distance of each DiffHunk is then summed at the Patch/File level to arrive at the total Levenshtein Distance for a commit.

Distribution

Table 2: Levenshtein Distance Dataset Statistics

repo_code	MIN(levenshtein_distance)	AVG(levenshtein_distance)	MAX(levenshtein_distance)
commercial-repo-1	1	246.49	45.8k
commercial-repo-2	1	363.62	30.9k
commercial-repo-3	1	222.9	64.7k
commercial-repo-4	1	206.52	10.5k
commercial-repo-5	1	235.92	30.5k
open-source-repo-1	1	171.14	15.2k
open-source-repo-2	1	339.43	32.6k
open-source-repo-3	1	536.61	436k
open-source-repo-4	1	509.28	27.1k
open-source-repo-5	1	565.86	93.4k
Totals	**1**	**372.5**	**436k**

Table 2: Levenshtein Distance Dataset Statistics shows minimum, mean, max, and standard deviation statistics about Levenshtein distance values measured in the dataset.

Issues

The Levenshtein Distance is not intelligent. The method can't identify logic or machine generated code. The calculation must rely on heuristic mechanisms to discern natural and unnatural contributions. This makes the calculation less suitable for calculating a derivative contribution rate, as the core measure underlying the rate is less representative of the human software developer's contribution.

5.3.2 LLM Logic Score Based Contribution

The Large Language Model (LLM) Logic Score Based Contribution measure attempts to address the lack of intelligence in the Levenshtein based method by offloading the evaluation of logic to a LLM / Generative AI.

Calculation Method

Experimentation with LLM based complexity evaluation was done primarily using Anthropic's Claude LLM though Amazon's Bedrock service [22] [23].

- Retrieval Augmented Generation
- A prompt is crafted that asks the LLM to make several evaluations.
- In the prompt is provided the new version of the source code file, as well as the before and after text of each DiffHunk that was changed.
    - The LLM seemed to get less confused and avoid hallucinations if the entire before/after file was not supplied, and only the changes were highlighted.
- The LLM is asked to assign a "Logic Score" by supplied criteria on a scale of 0-3.
- These Logic Score values were recorded at the DiffHunk level, rolled up to the Patch/File, and finally summed at the Commit level.

Issues

LLM Inference are expensive. It's prohibitively expensive to run self-hosted instances of an LLM capable of performing calculations like summarizing file changes that require such a large context window. Even performing the inferences using a cloud provider can be expensive.

LLM Inferences are time consuming, and often throttled by the cloud provider. Normal multithreading or other horizontal scaling techniques are not effective at accelerating performance in these environments.

To avoid errors from throttling, Amazon Bedrock's Batch Inference endpoint was employed. At the time of this writing, the endpoint was in developer preview, and would fail with an unexplained error for any batch size over about 100 inferences. One subject repository required over 23k inferences for just 30 days of repository history.

Although intriguing, and this method shows an incredible amount of promise, this research could not utilize the technique at scale.

5.4 Complexity Based Contribution

Complexity based contribution works by measuring the Cyclomatic Complexity delta generated by a developer between commits. The method is less about measuring Cyclomatic Complexity as it is about establishing a stable measure of change that accurately reflects human developer behavior. Cyclomatic Complexity was identified as a candidate for this measure when attempts to use LLM techniques to measure change failed for performance and cost reasons. Cyclomatic Complexity emerged as an effective balance between performance and results. Because Cyclomatic Complexity is measured by a local library, the system can determine Cyclomatic Complexity deltas exponentially faster than LLM based methods for determining logic changes. Measuring Cyclomatic Complexity deltas delivers more consistent results than a pure Levenshtein based method for determining contribution because only real logic changes are considered, and it's much less common for UnC to be mistakenly counted as natural contribution. Future research efforts could identify other alternative methods for measuring developer contribution.

5.4.1 Cyclomatic Complexity

Cyclomatic Complexity is a measure of the number of logic paths that a program might take during execution. The measure was developed by Thomas J. McCabe, Sr. in 1976 [24]. Cyclomatic Complexity is a popular alternative to Software Lines of Code (SLOC) measures because it considers the actual logic in the program flow and not just where there happened to be instructions occupying a line in a file. In contrast, Cyclomatic Complexity has been criticized as redundant to SLOC for its strong linear correlation to the measure [25]. Cyclomatic Complexity works by "measuring the number of linearly independent paths through a function in the source code" [26] [27]. In these experiments, only positive Cyclomatic Complexity deltas (CD) are

considered. Negative CD values are discarded to prevent credit being given for mass deletion of logic. The downside is that in the number of cases where Cyclomatic Complexity is reduced because of thoughtful programming improvement, the developer is not given credit for these values. An in-depth analysis for the number of instances where this happens and the effect that it might have on complexity-based contribution values is left for a future research opportunity.

5.4.2 Calculation Method

The Cyclomatic Complexity Based Contribution relies on heuristic methods provided by the Lizard Python library [28]. The Lizard Python library is a popular tool used to measure the Cyclomatic Complexity

- Commits are examined on a file-by-file basis.
- The methods in each file are identified and evaluated for a before/after Cyclomatic Complexity value.
- Based on the difference between the before/after complexity values for each method, a CD measure results.
- Only positive CD measures are counted. Negative CD values are discarded.
- This Complexity Delta is then rolled up from the Patch/File level and summed for the entire Commit.

5.4.3 Distribution

Table 3: Complexity Delta Dataset Statistics

repo_code	MIN(complexity_delta)	AVG(complexity_delta)	MAX(complexity_delta)
commercial-repo-1	-279.67	11.17	373.28
commercial-repo-2	-3.03	1.58	43.92
commercial-repo-3	-32	5.97	363.43
commercial-repo-4	-9.08	3.09	124.54
commercial-repo-5	-20.09	8.81	241.17
open-source-repo-1	-5.75	1.28	117.34
open-source-repo-2	-35.56	0.2658	338.88
open-source-repo-3	-72.73	0.7854	1k
open-source-repo-4	-26.24	2.02	169.71
open-source-repo-5	-35.78	0.5798	56.34
Totals	**-279.67**	**3.06**	**1k**

5.4.4 Advantages

During experimentation with Levenshtein and LOC contribution measurement it was observed that there were more outliers and anomalous instances of contribution being measured. These anomalies became especially apparent when using contribution results to calculate CR values. The “unnatural” contributions were more common and required complex and error prone heuristic processing to factor out. A quantitative analysis of the quality differences between the two measures is left as an opportunity for future research.

5.4.5 Issues

Unlike Levenshtein distance, Cyclomatic Complexity calculations are language dependent. Although the Lizard Cyclomatic Complexity library does support many languages, there are some such as Perl, Dart, and others that are not supported. This limitation was not an issue in these experiments, but on a broader scale having a language agnostic tool for analyzing software performance would be preferred. In this case, the value and performance delivered by using a measure based on Cyclomatic complexity was enough to introduce this limitation onto the tool.

It could be argued that giving credit for Cyclomatic Complexity encourages developers to write more complex code. Technically this is not false, but such an attempt to game the system would be very difficult to keep up over a long period of time and many commits. The theory works on averages and so aggregates many examples over a long period of time. In practice, development shops working on any sort of system of value will have a process by which code is peer reviewed in the context of a pull request prior to being merged. This review process would prevent the gratuitous introduction of complexity by developers wishing to game the system.

5.5 Conclusions

In conclusion, various methods of quantifying contributions within a software source code repository were explored. While the journey was marked by challenges and extensive experimentation, it yielded valuable insights into the nature of software development contributions and their measurement.

The Levenshtein-Based Contribution method offered a straightforward metric based on textual changes but fell short in discerning between cosmetic and logical changes. The LLM Logic Score-Based Contribution method attempted to incorporate an understanding of code logic into the evaluation but was hindered by high costs and computational demands. Ultimately, the Complexity-Based Contribution method, particularly one that focuses on cyclomatic complexity deltas, emerged as the most viable, striking a balance between capturing meaningful contributions and computational efficiency.

The study indicated that while no single metric perfectly encapsulates a developer's contributions, cyclomatic complexity offers a compelling proxy by focusing on the logical pathways through code rather than purely textual alterations. This complexity-based approach

aligns more closely with the qualitative impact of code changes, though it's not without its own set of limitations, such as language dependency and potential incentives for more complex code.

Future research should continue to refine these metrics, looking for ways to accurately represent developer contributions while mitigating the drawbacks identified throughout this investigation. It may involve combining elements from different approaches or exploring entirely new methodologies that can handle the nuances of software development contributions without prohibitive costs or complexity.

The development of contribution methods lay the foundation for continued exploration and innovation in the field of contribution quantification, providing the starting point from which more nuanced and revealing methods will be developed in the following chapters.

CHAPTER 6: CONTRIBUTION RATE

The Contribution Rate (CR) is the first derivative of any Contribution measure. The CR value is an approximate measure as the actual contribution rate is not possible to quantify precisely with only information from a source code repository. Extra information would be needed, speculation on which is beyond the scope of this research. The rate cannot be calculated for all commits as not all commits have the requisite Antecedent Commit by the same author needed to calculate the CTD required by Contribution Rate.

6.1 Calculating Contribution Rate

To calculate contribution rate, the Contribution measure (by any method, Complexity or Levenshtein) is divided by the CTD value:

$$\frac{C}{CTD}$$

For ease of comparison all CTD values are reported in hours, to the 2nd decimal precision. This result is a Contribution Rate that is "per hour". Levenshtein based CR values are reported in distance (words) per hour, Complexity based CR values are reported in Complexity Delta (CD) per hour.

6.2 Levenshtein Distance Based Contribution Rate

Levenshtein Distance Based Contribution Rate (LDBCR) takes the cumulative Levenshtein Distance of all files in a commit and divides this Contribution measure by the CTD value on the commit if present. Early experimentation was done using Levenshtein Distance Based Contribution Rate, but it was found that the contribution rate could vary widely, and the

resulting measures were found to be highly suspect and unusable when estimating hourly durations for commits. This was due to unnatural contributions being included in contribution and rate calculations, causing the deviations. LDBCR was not suitable for Contribution Rate Imputation because the estimated hours that result vary widely and in many cases are not plausible (> 168 in a week, more hours than a person would likely work in a period, or a greater hour estimate than the CTD value).

Table 4: Levenshtein Rate Dataset Statistics

repo_code	MIN(levenshtein_rate)	AVG(levenshtein_rate)	MAX(levenshtein_rate)
commercial-repo-1	0.01	1.75k	1.29M
commercial-repo-2	0.01	1.31k	719k
commercial-repo-3	0.01	1.92k	698k
commercial-repo-4	0.01	2.01k	449k
commercial-repo-5	0.01	1.52k	335k
open-source-repo-1	0.01	670.81	326k
open-source-repo-2	0.01	710.83	565k
open-source-repo-3	0.01	4.45k	4.18M
open-source-repo-4	0.01	791.82	240k
open-source-repo-5	0.01	1.49k	1.2M
Totals	**0.01**	**1.66k**	**4.18M**

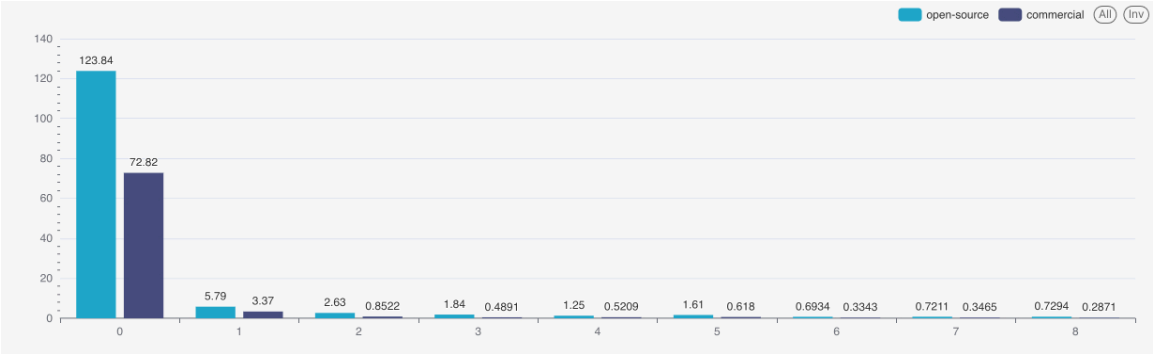

Figure 7: Mean Levenshtein Rate 0-8 Hour CTD Commercial vs. Open-Source

Although the study will not present an exhaustive demonstration of Levenshtein rate behavior in the pages, Figure 7: Mean Levenshtein Rate 0-8 Hour CTD Commercial vs. Open-Source is provided as an example to demonstrate the approximate behavior of CR values

based on this measure, especially to highlight the similarity in behavior to Complexity based rates discussed in later sections. The similarity between the two is important, and although Complexity based rates are generally perceived to be more accurate than Levenshtein based rates, the fact that bother exhibit a similar behavior with relation to CTD hour buckets supports that a Time-Delta based method for measuring CR values reflects actual developer behavior.

6.2.1 Words per Minute vs. Words per Hour

It could also be practical to report Levenshtein-based Contribution Rates in "words per minute". The well-known "w.p.m" measure allows the measure to be easily compared with a human's typing speed. As a rule of thumb, 40 wpm is be a rate that would be hard for a programmer to maintain that speed of contribution for any length of time [29]. 216 wpm is widely regarded as the fastest typing speed recorded by a human. Greater than 200 wpm is used in this example as an upper limit of a humanly possible Contribution Rate [30] [31].

This scale was used in early research efforts, but when focus was switched to Complexity based methods, the wpm significance was lost, and it seemed more straightforward to report both rate measures by the hour.

If Levenshtein distance were retained as a primary metric preferred for downstream calculations, it's likely that the experiment would have retained the per minute granularity when reporting contribution.

6.3 Complexity Based Contribution Rate

Like LDBCR, Complexity Based Contribution Rate takes the cumulative Complexity Delta on all files in a commit and divides this Contribution measure by the CTD value on the

commit if present. Because Complexity Based Contribution Rate is calculated from the Cyclomatic Complexity Delta that is more grounded in developer contributed logic, the derivative is more stable and is less likely to have spikes not related to natural development activities.

Table 5: Complexity Rate Dataset Stats

repo_code	MIN(complexity_rate)	AVG(complexity_rate)	MAX(complexity_rate)
commercial-repo-1	0.01	162.17	12.2k
commercial-repo-2	0.01	13.23	545.5
commercial-repo-3	0.01	101.65	16.1k
commercial-repo-4	0.01	87.26	4.73k
commercial-repo-5	0.01	127.71	14.7k
open-source-repo-1	0.01	37.45	6.67k
open-source-repo-2	0.01	12.26	5.65k
open-source-repo-3	0.01	40.4	10.5k
open-source-repo-4	0.01	9.22	4.28k
open-source-repo-5	0.01	8.55	1.99k
Totals	**0.01**	**70.62**	**16.1k**

6.4 Contribution Rate Behavior

Contribution rates measured by Levenshtein, Complexity or other methods all share a similar characteristic relationship with the CTD value. Commits with a shorter CTD value tend to have a higher CR value. This is generally caused by quick remedy commits where developers have a short and focused amount of code to add. Because time is not lost on thinking. The higher mean CR values in the first 15-minute bucket may be explained in part by “quick fix” remedy commits where developers have a very specific task that they are coding [32] [33]. When a developer requires less time to think or other non-coding tasks when implementing a targeted fix for a relatively small issue, it stands to reason that the textual change introduced per minute may be higher, even though the number of minutes is fewer.

6.5 Contribution Rate Distribution

Both Complexity and Levenshtein based CR values are right skewed towards having lower contribution rates. In the charts that follow, results with less than 1 Complexity or Levenshtein units are omitted as there are so many results in those buckets, it would make the chart less readable.

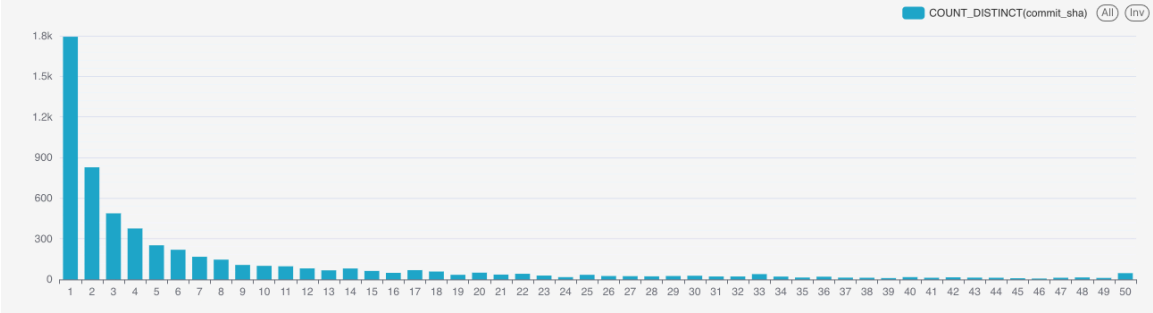

Figure 8: Complexity Rate Distribution Buckets >0

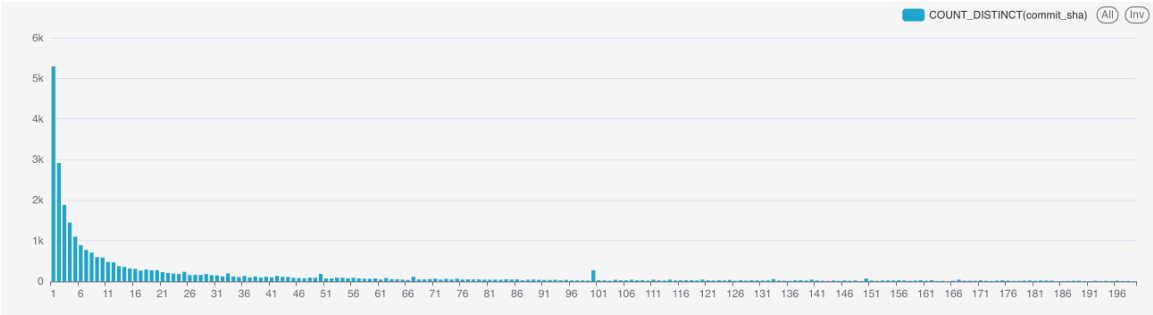

Figure 9: Levenshtein Rate Distribution Buckets >0

6.6 CR vs. CTD Distribution

Contribution Rate seems to be not normally distributed in most cases. CR values were observed to be overwhelmingly right skewed towards the higher CR values and lower CTD

buckets. Because CR values were not observed to be normally distributed around a mean, an inter-quartile-range (IQR) method was used to evaluate outliers.

Following are a series of visual charts included to communicate the behavior of CR values over CTD hour buckets. Complexity-based CR values are used in these examples as these values exhibited more stability with fewer outliers and are generally the preferred method of measure for experimentation. These visuals and the distribution of CR values are significant because they highlight the observed behavior of CR values, and some correlation can be drawn between human developer behavior and observations in the data set population. The disparity in contribution rate in the 0-hour CTD bucket is especially significant, as it's important to understand for downstream calculations like CR value imputation, and the estimation of commit hours.

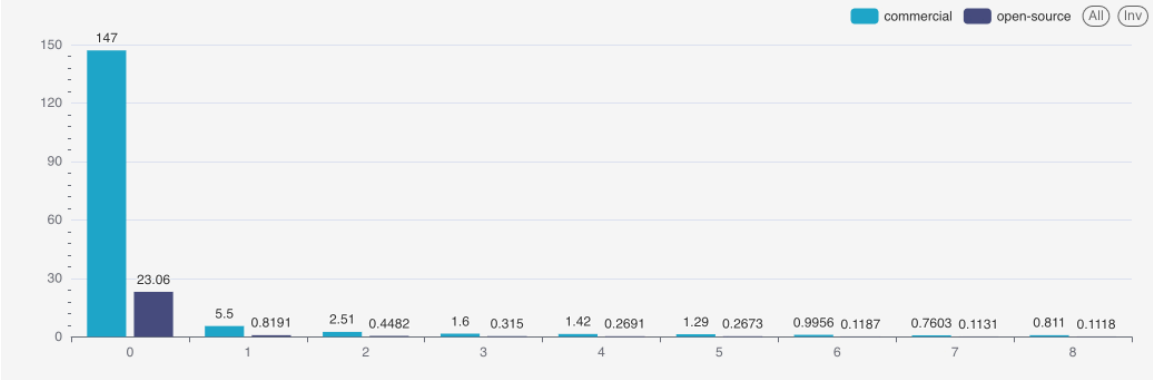

Figure 10: Mean Complexity Rate 0-8 Hour CTD Commercial vs. Open-Source

The Figure 10: Mean Complexity Rate 0-8 Hour CTD Commercial vs. Open-Source chart describes the mCR value of open-source projects over the first 9 CTD hour buckets. The x-axis represents the CTD hour bucket (rounded), and the y-axis measures the mCR value for that CTD bucket. Data in the chart indicates that Commercial projects have a significantly higher

mCR value during the 0-hour CTD bucket with an mCR of 147 CD/hr. complexity deltas per hour, in contrast to Open-Source projects that show a lower rate of 23.06 CD/hr.

As the hourly CTD buckets increase from 1-8, the CR value for both Open-Source and Commercial repositories fall. The mCR values for Commercial repositories fall from 5.5 CD units per hour in the 1-hour CTD bucket, to 0.811 in the 8-hour CTD bucket. Across all CTD hour buckets, Open-Source projects show lower CR values than Commercial repositories.

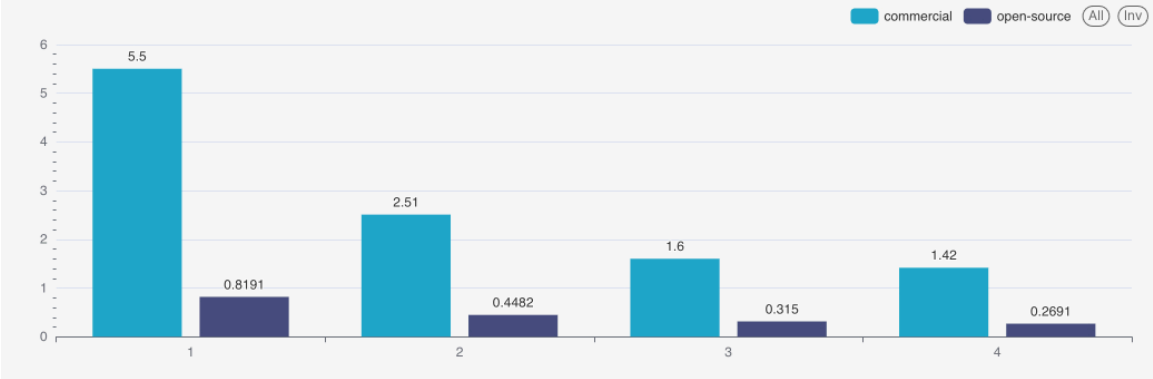

Figure 11: Mean Complexity Rate 1-4 Hour CTD Commercial vs. Open-Source

To illustrate the behavior of CR values more clearly in the 1–4-hour CTD, results are shown where the scale is restricted to the values in these buckets without the higher rates from the 0-hour CTD bucket dwarfing the readings. It can be observed that the Commercial repository values in this population have a steadily declining rate from 5.5 CD/hr. in the 1-hour CTD bucket, down to 1.42 CD/hr. in the 4-hour CTD bucket.

The study highlights these CTD buckets, in part because of their “Goldilocks” value position in the observations (not too high, not too low), but also deriving from industry experience. Anecdotally a 1–4-hour programming session could be considered normal and “as best as one could hope for” in a normal workday. The tendency for a developer to leave the keyboard for breaks or other activities could also be explained by occupational fatigue in some

cases [34] [35]. For a variety of reasons, it's common for a developer to not even achieve this level of productivity as most professionals that have worked in a commercial setting can attest to. Such a fantastic programming session might be one where the developer works without interruption and continuously contributes productive Complexity Delta stopping only for the occasional coffee refill, bathroom break, or pensative reflection on the logic at hand. Identifying these optimal rates of contribution will allow other less than productive sessions to be identified and compared. It's probable that if these methods were commercially applied, they may not assert a precise down to the minute estimation of activity, but rather a range potentially declaring a tolerance of +/- 2 hours. While greater accuracy is better, even a tolerance of a few hours would be telling for resources that performed on the low end of the spectrum. This interesting range of CTD buckets is discussed again in later sections when experimentation turns to identifying a "model" CR value for the Contribution Rate Imputation Method.

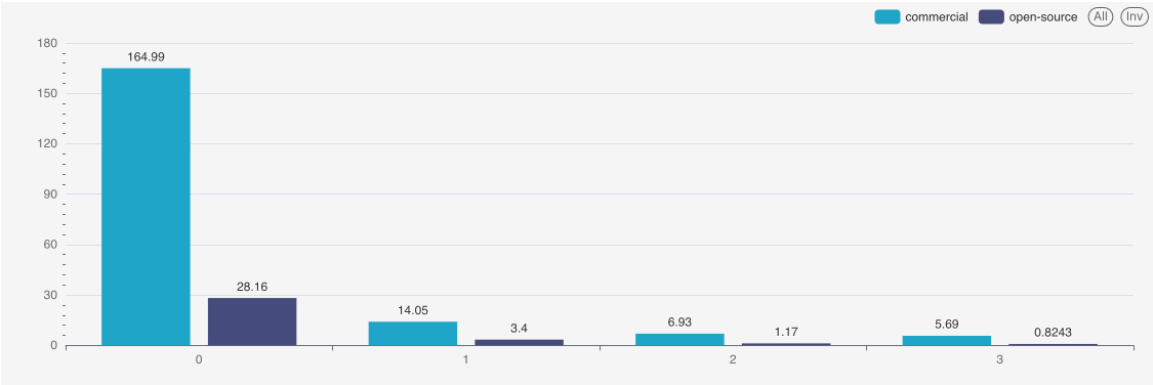

Figure 12: Mean Complexity Rate < 1 Hour CTD in 15 Minute Buckets

Figure 12: Mean Complexity Rate < 1 Hour CTD in 15 Minute Buckets compares the mCR between Commercial and Open-Source repository groups. Specifically in the first 0-hour CTD bucket broken down into four 15 minute/quarter hour intervals. The slide of data is represented to this granularity, to further detail the distribution of CR values within the first hour.

The chart shows noticeably higher CR values in the first quarter hour reporting 164.99 CD/hr. for Commercial repositories, and 28.16 CD/hr. for Open-Source. In the second quarter hour the values drop to 14.05 and 3.4 CD/hr. for Commercial and Open-Source repositories respectively. These values continue to fall for the rest of the 0-hour CTD bucket. This data indicates that while commits within any period of the first 0-hour CTD bucket tend to exhibit higher CR values, the first 15 minutes exhibit higher CR values by far.

6.7 Conclusions

Significant findings regarding the nature of Contribution Rates (CR) in source code repositories are synthesized. Notably, it was observed that CRs, as the first derivative of contribution measures, demonstrate a downward trend relative to the time between commits (CTD). This consistent decline suggests a strong correlation with common developer behaviors, where shorter CTDs typically correlate with higher CR values. This phenomenon could be attributed to the focused nature of quick remedy commits, which require minimal conceptualization time, thus elevating the measure of contribution per unit time.

The exploration delineated two primary methods of calculating CR: the Levenshtein Distance and Complexity-based approaches. Despite the initial utilization of the Levenshtein Distance, its propensity to produce erratic and implausible estimates rendered it less reliable for imputing CR values. Conversely, the Complexity-based method proved more stable, echoing the more systematic aspects of programming tasks and providing a more accurate reflection of developer logic contributions.

The implications of these findings are multi-faceted. First, they affirm the utility of Time-Delta based metrics for reflecting actual development practices. Second, the insights gained from examining various CR distributions bolster the argument that longer CTD values are likely to

yield lower CR values. This is indicative of a developer’s work being interrupted and not necessarily of tasks requiring greater contemplation or complexity.

Despite the insights offered by this research, it is essential to recognize the inherent limitations in CR as a metric. Given that CR is fundamentally an approximate measure due to the incomplete data derivable from source code repositories alone, the results necessitate cautious interpretation. It remains clear that CR's utility as a metric lies in its ability to provide a directional, rather than absolute, understanding of developer contribution patterns.

In conclusion, while neither Levenshtein nor Complexity-based methods deliver an infallible measure of developer productivity, the latter's consistency and alignment with observed developer behaviors make it the superior choice for estimating CR values. As such, this research not only contributes to the existing body of knowledge on developer productivity measurement but also underscores the potential for further refinement in methods to approximate contribution rates more accurately.

CHAPTER 7: CONTRIBUTION RATE STATISTICAL SIGNIFICANCE

To further explore the behavior of CR values in the experimental environment, a series of tests for statistical significance were performed. The hypothesis is that if CR values are statistically significant, or they are likely to come from a population with a similar distribution, then it indicates that rates are approximately repeatable, stable under some conditions, and it does not invalidate the Time-Delta based method that derives the rate. If statistically significant values can be measured from a population sample, these model rates would be candidates for contribution rate imputation (see: Contribution Rate Imputation).

7.1 Repository Population Statistical Significance

7.1.1 Experimental Approach

The approach to evaluating statistical significance was to evaluate contribution rates from a single repository against other commits in the population. This comparison was performed in several permutations to explore where statistical significance amongst the population may exist. Not all permutations yielded statistically significant results in all cases. Commercial and Open-Source populations were examined separately in case either set exhibited distinct behavior.

Permutations

The different permutations applied to test statistical significance are summarized in Table 6. These permutations are referenced when results are reported, as they affect statistical significance.

Table 6: Permutations for Evaluating Statistical Significance.

Permutation	Description
Repository	The repository being evaluated.
Reference Organization	The organization against which the repository data was tested for statistical significance. (commercial, open-source, or All)
IQR Source	The source from where the IQR range was referenced, for the purposes of eliminating outliers. (commercial, open-source, or All)
CTD Hour Bucket	The rounded hour bucket of the CTD value (0-4)
Rate Type	The type of rate being examined, Levenshtein or Complexity.
Bounds	These bounds are used to eliminate outliers. • iqr_outliers = +/-IQR * 1.5 • iqr_inclusive = IQ1 =><= IQR3 • iqr_exclusive = IQ1><IQ2

Statistical Methods

To evaluate statistical significance, select methods from the SciPy Python library were used. Multiple tests were run on all permutations to identify instances of statistical significance. For brevity, Kruskal-Wallis results are generally reported. Comparisons of select methods are offered for demonstration. Additional scenarios are also enumerated in appendices. Results from ANOVA tests were excluded from the report due to the fact that the method requires samples to come from populations that are normally distributed [36]. The observed contribution rate data was not normally distributed.

1.1.1.1.1 Kruskal-Wallis Test

The SciPy documentation quotes: “The Kruskal-Wallis H-test tests the null hypothesis that the population median of all of the groups are equal. It is a non-parametric version of ANOVA” [37] [38].

1.1.1.1.2 Mann-Whitney U Test

The SciPy documentation quotes: “The Mann-Whitney U test is a nonparametric test of the null hypothesis that the distribution underlying sample x is the same as the distribution underlying sample y. It is often used as a test of difference in location between distributions.” [39]

7.1.2 Commercial Repositories

Commercial repository values are tested separately for statistical significance because of potential differences in development and commit behavior between Commercial and Open-Source repository contributors. The following sections explore different permutations and their effect on statistical significance amongst CR values taken from Commercial repositories.

1.1.1.1.3 IQR Source: Commercial

IQR Source: Commercial results take the IQR used to filter outliers from the IQR of only Commercial repositories in the population.

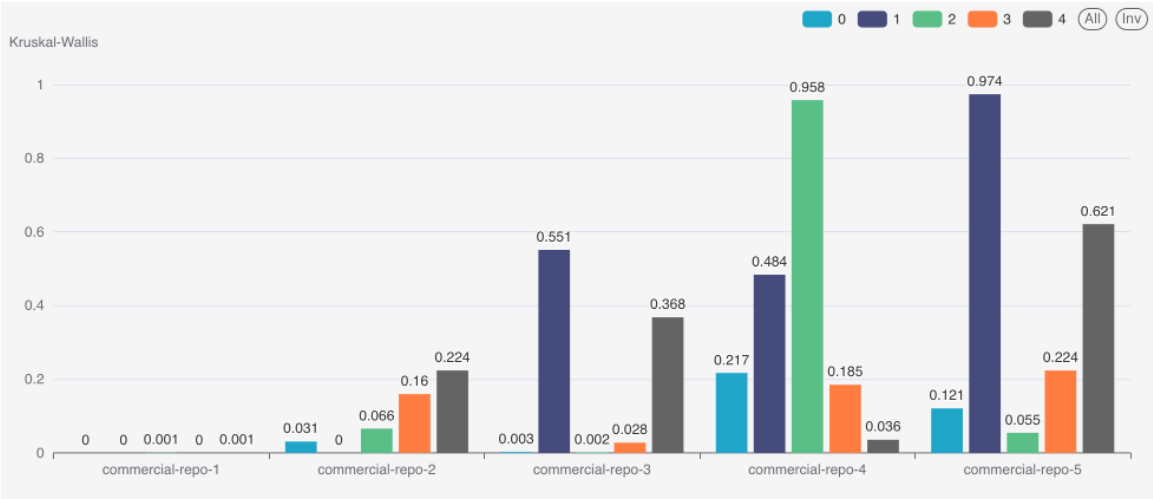

Figure 13: Commercial Complexity Rate Kruskal-Wallis 0-4hr CTD (IQR Source: Commercial)

Table 7: IQR Source Commercial Permutation Details.

Permutation	Description
Repository	Commercial repositories 1-5
Reference Organization	Commercial
IQR Source	Commercial
CTD Hour Bucket	0-4
Rate Type	Complexity
Bounds	iqr_inclusive = IQ1 =><= IQR3

Table 8: IQR Source Commercial IQR Details.

CTD Hours	IQR	IQR1	IQR3
0	366.515	13.485	380
1	14.06	1.228	15.288
2	6.8	0.51	7.31
3	4.24	0.33	4.57
4	3.21	0.28	3.49

Figure 13 shows p-value results from a Kruskal Wallis test for statistical significance for the 0–4-hour CTD CR values when considering those values within the IQR range of only Commercial repositories. While the results do indicate statistical significance especially in commercial-repo-4 and commercial-repo-5, other results are less than significant. When comparing the differing IQR values between the IQR Source: Commercial and IQR Source: All, the values in Table 8 cover a greater range of CR values than Table 10. The wider CR range invites more data points into the comparison, reducing statistical significance.

1.1.1.1.4 IQR Source: All

IQR Source: All results take the IQR used to filter outliers from the IQR of only Commercial repositories in the population.

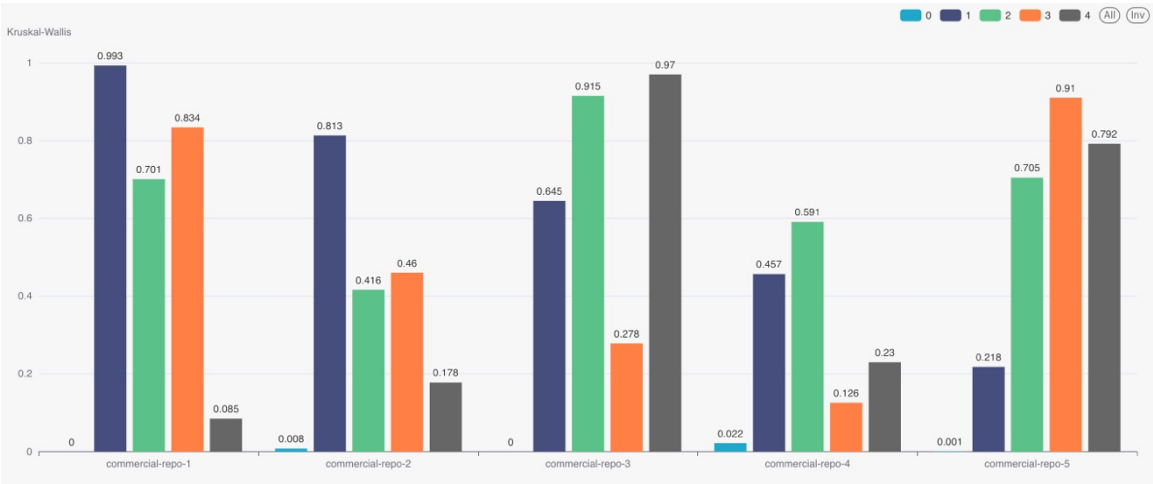

Figure 14: Commercial Complexity Rate Kruskal-Wallis 0-4hr CTD (IQR Source: All)

Table 9: Commercial Repositories, IQR Source All Permutation Details

Permutation	Description
Repository	Commercial repositories 1-5
Reference Organization	Commercial
IQR Source	All
CTD Hour Bucket	0-4
Rate Type	Complexity
Bounds	iqr_inclusive = IQ1 =><= IQR3

Table 10: Commercial Repositories, IQR Source All Details

CTD Hours	IQR	IQR1	IQR3
0	183.578	3.598	187.175
1	7.26	0.23	7.49
2	3.672	0.13	3.802
3	2.71	0.08	2.79
4	1.937	0.073	2.01

Figure 14 shows p-value results from a Kruskal Wallis test for statistical significance for the 0–4-hour CTD CR values when considering only those values within the IQR range of all commits in both Commercial and Open-Source repositories. For this sample, the test indicated that there is a statistically significant possibility the CR values in the 1–4-hour CTD buckets for each repository came from a population with the same mean. Not surprisingly, the 0-hour CTD bucket had low to no statistical significance. This CTD bucket seems to have a wider range of potential CR values due to quick remedy commits and the greater potential for developers to have focused additions of logic in a very short period, sometimes in multiple succession, mixed with commits with much lower CR values.

7.1.3 Open-Source Repositories

Open-Source repositories like Commercial repositories are tested in isolation for statistical significance. As noted in prior sections, commit and development patterns in Open-Source repositories tend to be different than that of Commercial repositories, generally reporting lower CR values for the same CTD windows. The difference in the size of these windows measured has an effect on the statistical significance measured.

1.1.1.1.5 IQR Source: Open-Source

IQR Source: Open-Source reports statistical significance results with values filtered by the IQR range of Open-Source repositories only.

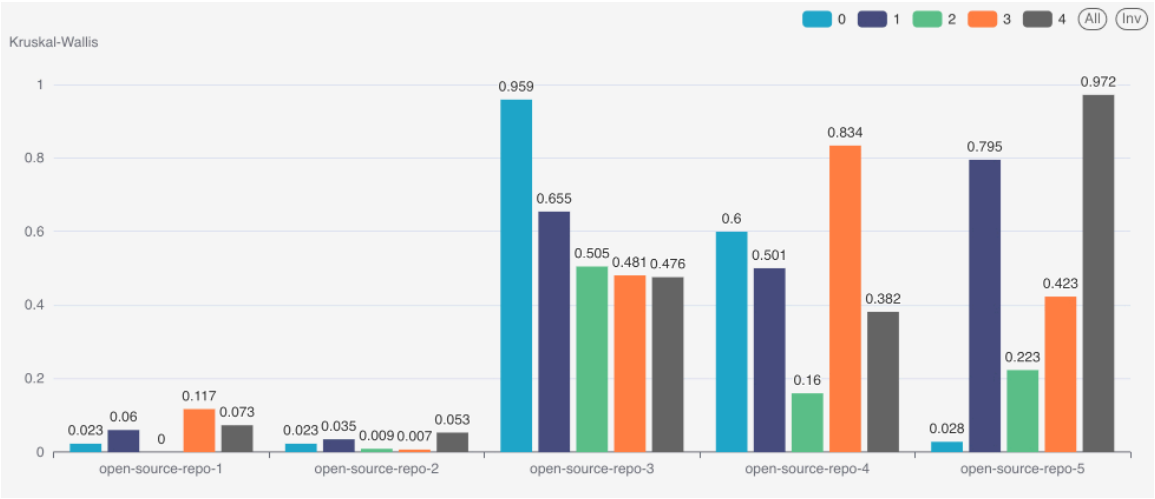

Figure 15: Open-Source Complexity Rate Kruskal-Wallis 0-4hr CTD (IQR Source: Open-Source)

Table 11: Open-Source Repositories, IQR Source All Permutation Details

Permutation	Description
Repository	Open-Source repositories 1-5
Reference Organization	Open-Source
IQR Source	Open-Source
CTD Hour Bucket	0-4
Rate Type	Complexity
Bounds	iqr_inclusive = IQ1 =><= IQR3

Table 12: IQR Source Open-Source IQR Details

CTD Hours	IQR	IQR1	IQR3
0	33	1	34
1	2.93	0.12	3.05
2	1.16	0.07	1.23
3	0.71	0.04	0.75
4	0.6875	0.04	0.7275

Figure 15 shows a statistical significance in repositories open-source-3, open-source-4, open-source-5. The CR distribution of the open-source-1 and open-source-2 repositories show

almost no significance across any of the CTD windows when compared with other Open-Source repositories. The difference in distribution could have more to do with repository culture and developer behavior, signifying that the contribution rate distribution on significant repositories have more in common with each other, leading to a higher statistical significance when the rates are tested against samples from the other Open-Source repositories. Conversely the open-source-1 and open-source-2 repositories exhibit CR distributions that are not likely to be statistically significant with the other Open-Source repositories. A deeper examination of these distributions and how/why they might differ is left as an exercise for future research efforts.

1.1.1.1.6 IQR Source: All

IQR Source: All reports statistical significance results with values from all other repositories, Open-Source and Commercial included.

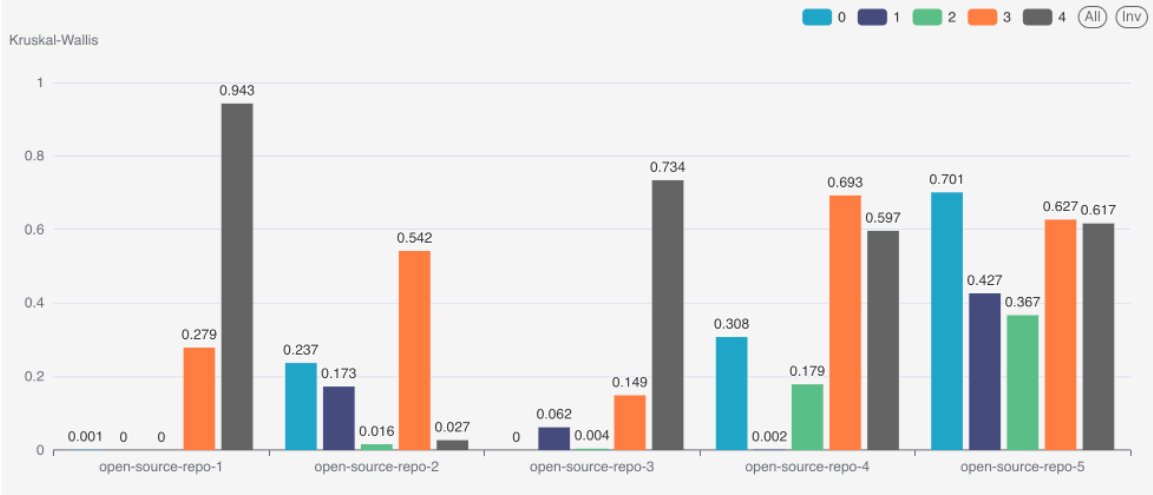

Figure 16: Open-Source Complexity Rate Kruskal-Wallis 0-4hr CTD (IQR Source: All)

Table 13: Open-Source Repositories, IQR Source All Permutation Details

Permutation	Description
Repository	Open-Source repositories 1-5
Reference Organization	Open-Source
IQR Source	All
CTD Hour Bucket	0-4
Rate Type	Complexity
Bounds	iqr_inclusive = IQ1 =><= IQR3

Figure 16 reports slightly higher p-values across certain CTD windows for open-source-1 and open-source-2 repositories when using an IQR value taken from the entire population of repositories. For the rest of the repositories, p-values are reduced in most cases.

7.1.4 Complexity vs. Levenshtein Rates

The study holds the hypothesis that Levenshtein based CR values are less accurate than Complexity based CR values. To support that hypothesis, the statistical significance of CR values calculated using these two methods was tested. It was expected that the statistical significance of Levenshtein based CR values would be less or non-existent. The results were surprising, and it was observed that this was not always the case.

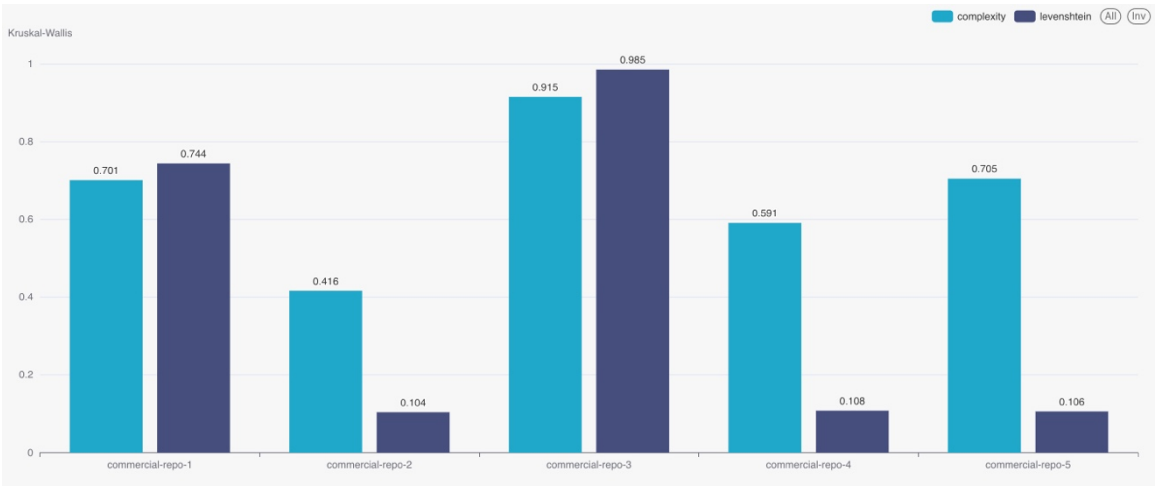

Figure 17: Complexity Rate vs. Levenshtein Rate Kruskal Wallis Comparison, Commercial Only

Table 14: Complexity Rate vs. Levenshtein Rate Kruskal Wallis Comparison, Permutation Details

Permutation	Description
Repository	Commercial
Reference Organization	Commercial
IQR Source	ALL
CTD Hour Bucket	2
Rate Type	Complexity vs. Levenshtein
Bounds	iqr_inclusive = IQ1 =><= IQR3

Figure 17: Complexity Rate vs. Levenshtein Rate Kruskal Wallis Comparison, Commercial Only shows that not only do Levenshtein based CR values exhibit a high likelihood to come from a population with the same mean, in some cases these values were showed the potential for more significance than their Complexity based counterpart. The results of this test are inconclusive and a deeper examination of sub populations in the dataset could yield different results.

7.1.5 Repository Population Conclusions

In the assessment of repository population statistical significance, the investigation reveals as yet unpredictable patterns within the population of contribution rates. It is evident that the underlying distribution of these rates—whether originating from Commercial or Open-Source repositories—exhibit variations that reflect distinct behavioral and operational conditions within each repository's "development micro-culture".

Through permutation analysis and application of non-parametric statistical tests, such as the Kruskal-Wallis and Mann-Whitney U, the study observed that statistical significance is not uniform across all scenarios. Notably, certain Commercial repositories demonstrate a marked statistical significance, particularly within specific CTD windows. While other repositories from

the same “organization” do not share these same characteristics. Conversely, Open-Source repositories present a varied picture; while some show significant results, others do not. This common observation on Commercial as well as Open-Source repositories underscores the potential influence of unique repository cultures and developer behaviors on contribution rates.

The fluctuations in p-values across different permutations suggest that while there may be common threads in CR value distributions within certain repositories, statistical significance, at least as measured by these experiments, is not to be assumed. This variability can be attributed to a number of factors—ranging from individual developer habits and task-specific factors to broader considerations such as regional holidays and operational practices.

The hypothesis positing that Levenshtein based CR values are less accurate than Complexity based CR values requires re-evaluation due to surprising findings. Contrary to expectations, Levenshtein based CR values did not uniformly exhibit lower statistical significance. In fact, they sometimes matched or even exceeded the significance of Complexity based CR values in comparisons limited to commercial repositories, as demonstrated by the Kruskal-Wallis comparison in Figure 17.

This suggests that, at least within the Commercial context of this experiment, Levenshtein measurements can yield statistically relevant data that could be as indicative of trends as the Complexity measurements. However, the results were not consistently in favor of one method over the other, indicating the necessity for a more nuanced approach, perhaps considering specific subpopulations within the dataset for a more detailed analysis.

The conclusion here must acknowledge the complexities of the dataset and the limitations of a one-size-fits-all approach to measuring CR. Levenshtein rates have shown potential where complexity measures were expected to dominate, but the overarching narrative is that the search

for statistical significance in contribution rates is nuanced and depends heavily on the context of the repository and the nature of the commits. This study lays a foundation for further experimentation with more granular focus on repository subpopulations continued in section 7.2.

7.2 Repository Temporal Statistical Significance

This section introduces the concept of Repository Temporal Statistical Significance within the context of CR values. This section follows previous explorations of Repository Population Statistical Significance, which did not yield deterministic results, and thus invites further investigation into repository subpopulations. The emphasis is on temporal periods, under the hypothesis that they are more likely to display statistical significance when compared against samples CR values from the same repository population.

This hypothesis is based on the understanding that temporal periods inherently aggregate various subpopulations within a repository, such as developers and files, which may exhibit erratic CR values for reasons beyond this study's current scope. This focus is further justified by the lower computational and time resource requirements, owing to prior coding of the dataset with temporal identifiers to facilitate time-based analysis.

The experimental approach delineated in 7.2.1involves extraction and filtering of commit CR values, followed by comparison across different permutations. The permutations are succinctly described in Table 15, providing a clear framework for the experiments conducted.

In section 7.2.2, the calculation method centers on establishing baselines for outlier detection using the Interquartile Range (IQR) and employing non-parametric statistical tests to evaluate the significance of contribution rate variations within these temporal periods.

The Select Results outlined in section 7.2.3 present compelling findings, indicating high rates of statistical significance across temporal periods within repositories. The consistency of

these results suggests inherent patterns in contribution rates over time, which is further evidenced by the visualizations in Figure 18 through Figure 21.

Finally, the conclusions drawn in section 7.2.4 reinforce the notion that statistically significant subpopulations of CR values are indicative of a repeatable distribution, lending credibility to the Time-Delta method for assessing software development contribution rates. The systematic analysis emphasizes the robustness of the study's methodology and the relevance of its findings in the field of software development metrics.

7.2.1 Experimental Approach

The process begins with the extraction of all commit contribution rates from the repository, filtered by specific variables such as rate type (Complexity or Levenshtein), the rounded hours to the nearest CTD hour, and the repository code. The logic employs queries to retrieve data from a database, with filters based on the defined permutations. In the greater experiment, values for all permutations were tested against all other permutation values yielding an exponential number of results. A healthy selection of results allows the experiment the opportunity to identify variations between permutations should any exist.

Table 15: Repository Temporal Statistical Significance Permutations

Permutation	Description
Repo Code	The repository being examined. All repositories in the population were examined individually.
Rate Type	The rate type being examined. Both Complexity and Levenshtein based rates were examined.
Temporal Period Type	The type of time period being examined. Day, Week, and Month temporal periods were examined independently.
CTD Hour (Rounded)	The CTD window being examined. The CTD hour values are rounded to the nearest hour, +/- 0.5 hours.

7.2.2 Calculation Method

To establish a baseline for identifying outliers, the experiment calculates the IQR. This involves querying the subpopulation to determine the first quartile (Q1) and the third quartile (Q3) of contribution rates. The IQR is the difference between these quartiles, and the upper and lower bounds for outlier detection are set at 1.5 times the IQR above Q3 and below Q1, respectively.

Following this, the logics queries the database to obtain a filtered dataset of contribution rates that exclude outliers. Commits are then grouped by their temporal periods (days, weeks, or months). Each temporal period is individually tested for statistical significance against the rest of the repository's contribution rates, excluding the current period being examined. This is done using the Kruskal-Wallis and Mann-Whitney U tests, which are non-parametric methods suitable for comparing two samples that are not normally distributed.

Finally, the p-values from these tests are calculated and compared to a predetermined alpha level of 0.05 to determine if the differences in contribution rates are statistically significant. The results, including statistical measures and test outcomes, are persisted for subsequent analysis.

This systematic approach allows for a granular examination of statistical significance in contribution rates over time within repositories.

7.2.3 Select Results

The following sections detail select results from the examination of the permutations of temporal periods in repository subpopulations. The values reported in Select Results are the ratio of temporal periods within the subpopulation that reported statistically significant. A value of 1.0 reports all members are statistically significant.

CTD Window Monthly Rate Comparison

CTD Window Monthly Rate Comparison details the statistical significance of Complexity-based contribution rates tested using the Mann-Whitney U test. Results are visualized across the 0-4-hour CTD windows and are divided per repository and Commercial vs. Open-Source classifications respectively.

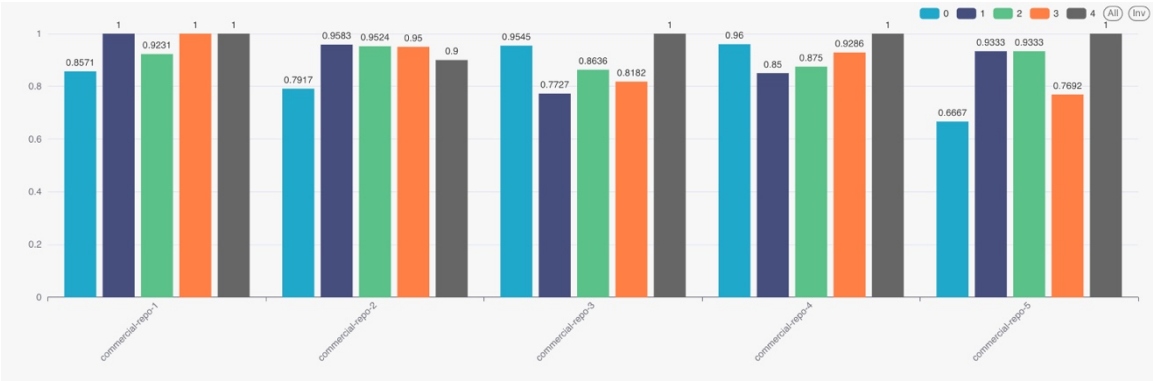

Figure 18: Commercial Repository Monthly Complexity Rate Significance Mann-Whitney 0-4hr CTD.

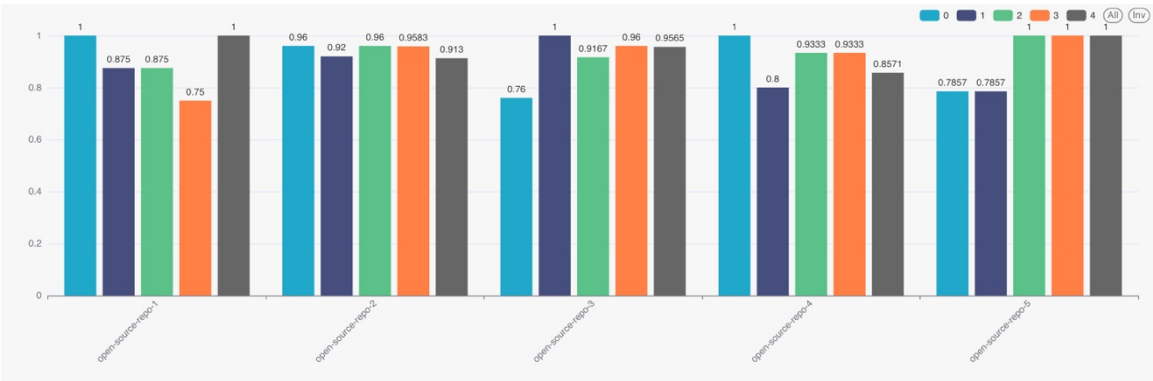

Figure 19: Open-Source Repository Monthly Complexity Rate Significance Mann-Whitney 0-4hr CTD.

Both Figure 18 and Figure 19 reports high rates of statistical significance across the 0–4-hour CTD windows for all repository permutations. This high rate of statistical significance gives a strong indication that within a repository, there are significant commonalities repeated across temporal periods with regard to contribution rate.

Temporal Period Comparison

Temporal Period Comparison details the statistical significance of Complexity-based contribution rates tested using the Mann-Whitney U test across Day/Week/Month temporal periods. Results are divided per repository and Commercial vs. Open-Source classifications respectively.

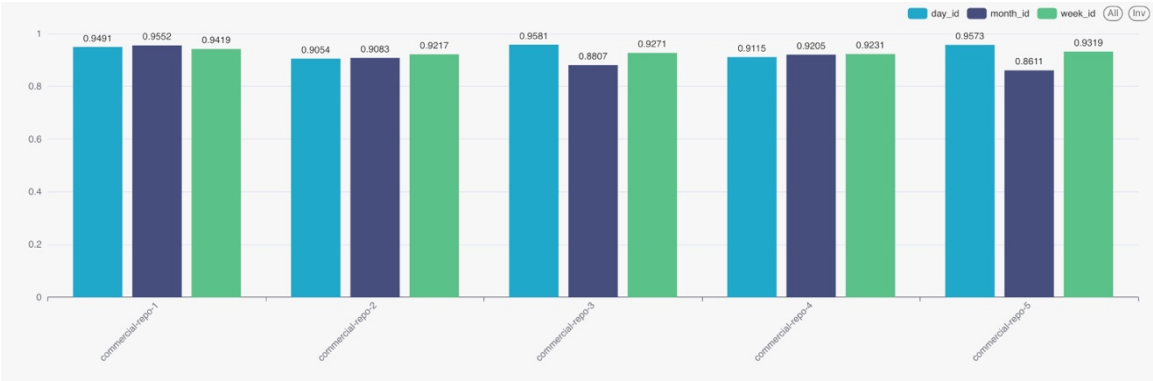

Figure 20: Commercial Repository Day/Month/Week Period Complexity Rate Significance Mann-Whitney

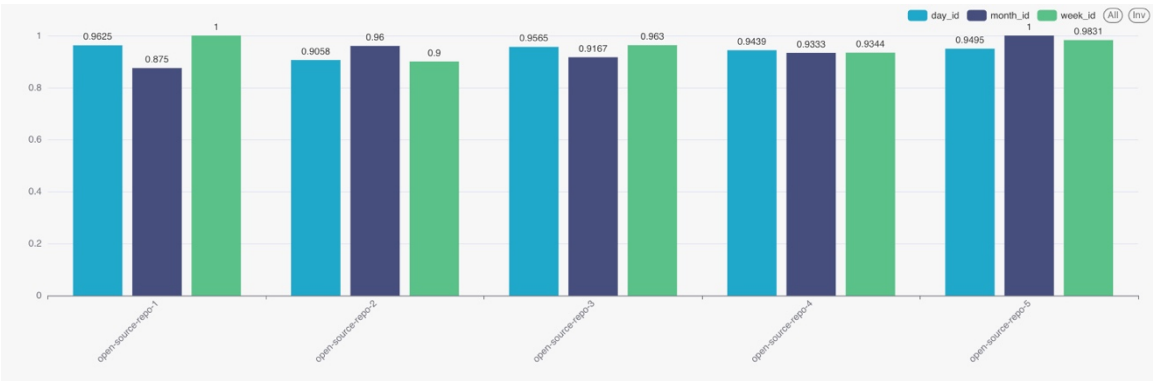

Figure 21: Open-Source Repository Day/Month/Week Period Complexity Rate Significance Mann-Whitney

Continuing the high rates of statistical significance reported in the CTD Window Monthly Rate Comparison section, each Day/Month/Week temporal period in Figure 20 and Figure 21 report undeniably high rates of statistical significance.

7.2.4 Conclusions

The investigation into Repository Temporal Statistical Significance has yielded insight into the behavior of CR values within software repositories. The experimental procedures and statistical analyses applied in this section have demonstrated a robust framework for examining the temporal distribution of CR values, validated by the high rates of statistical significance reported in the examined subpopulations.

The use of temporal identifiers in the dataset proved advantageous, lending a time-based analysis that was not only computationally efficient but also effective in revealing sub-patterns of contributions. By employing non-parametric statistical tests, namely the Kruskal-Wallis and Mann-Whitney U tests, the study addressed the non-normal distributions typically found in CR value sample.

The visualizations and quantitative results presented from Figure 18 to Figure 21, particularly the consistently high rates of statistical significance across various temporal periods, reinforce the validity of temporal periods as significant factors in the analysis of CR values. These findings suggest the presence of inherent patterns in contribution activities over time, indicating a potential for these patterns to serve as reliable indicators of developer engagement and project dynamics.

The success of the Time-Delta method, as evidenced by the high statistical significance of CR distributions, underlines its potential as a repeatable and valid measure of software development contribution rates. The significance of these results extends beyond the immediate context of the study, hinting at broader applications for the Time-Delta method in evaluating development efforts and productivity. By highlighting the temporal consistency in CR values, the research contributes a compelling method to measuring software development contribution rates.

CHAPTER 8: CONTRIBUTION RATE IMPUTATION

Because Contribution Rates tend to decline with increasing CTD windows, this indicates that the longer the period between commits by a developer, the more likely it is that the developer will work on another task, and therefore the Contribution Rate is less likely to represent a constant and uninterrupted work session. The Contribution Rate Imputation Method (CRIM) refers to a technique where the Contribution Rates of commits that are believed to come from relatively consistent work session are then assigned to those commits whose contribution rates do not represent a consistent and uninterrupted work session. Like the calculation of Contribution, there are multiple approaches that can be considered. This chapter discusses two that were explored.

8.1 Model Contribution Rate Hypothesis

The mCR hypothesis is that if a CTD window is short enough, it is mor likely that the developer worked continuously on the task without interruption. To calculate the mCR, only commits within the same “Organization” are considered. In the context of this experiment, commercial repositories make up one separate and distinct organization, and open-source repositories make up the other. When calculating a Model Contribution Rate values determined to be +/- 1.5 * IQR, are dropped to eliminate outliers. Eliminating outliers removes anomalies and other unnatural influences on the Model Contribution Rate. This hypothesis is useful in both the Gradient Boosted Tree (GBT) and Mean Bound Contribution Rate (mBCR) methods. For machine learning /GBT based methods, the mCR values can serve as training values to support

unsupervised learning on a population. For the MBCR methods, the "mean" is taken from these mCR values.

8.2 Candidate Method: Gradient Boosted Tree Regression

Although Gradient Boosted Tree Regression was not the method ultimately used to calculate mCR in the final data set, some important lessons learned from experimentation are included here for future reference. Gradient Boosted Trees are a popular type of machine learning model used in a wide variety of applications. Some varied examples are predicting travel time [40], wind pressures [41], and concrete strength [42], just to name a few. Wang et al. (2018) says: "The Gradient Boost method builds a strong learner by combining weak learners through iterative methods and the Decision Tree is a basic classification and regression method. As an ensemble machine learning algorithm, the Gradient Boost Decision Tree algorithm can offer higher forecast accuracy than one single learning algorithm." [43]

GBT Regression was used to train a machine learning model on past mCR values as they relate to commit features. Once the model was trained, predictions were made on commits that have long CTD durations and artificially low Contribution Rate values.

8.2.1 Features

Gradient Boosted Tree Regression introduces an interesting opportunity to consider different features that might well predict Contribution Rates Table 16: Gradient Boosted Tree - Candidate Features enumerates a few that were explored.

Table 16: Gradient Boosted Tree - Candidate Features

Feature	Description
Author	Do authors tend to contribute at different rates, and therefore the feature of

	"Author" on the commit might well predict the CR value of the contribution?
Filename	Do certain files tend to have different CR values? For example, are the contents/logic of some certain files more difficult (slower) to work on than others?
Commit Time	Does the time a commit was pushed to the repository correlate with the CR value of the overall contribution in any way?
Number of Commit Files	Do commits with a larger number of files exhibit higher or lower CR values?

These are just a few examples of features that could correlate with commit CR values. While these features and the GBT regression method in general all showed promise, it quickly became clear that the effort necessary to fully explore this candidate technique was beyond what was possible in the scope of this experiment. Opportunities exist to introduce other features based on repository regression and the likelihood that the commit in question would conform to the predicted value. This opportunity is left for future research.

8.2.2 Issues

In the end the results of the GBT because hard to trace and defend. The resulting contribution rates could vary widely with little explanation. The method was abandoned in experimentation because of the difficulty in defending the resulting values. Rates and the resulting hours would vary widely with little traceability. Some studies have suggested that more and higher quality features may improve GBT performance [44]. It's likely if more correlated features could be identified, then a Gradient Boosted, or other ensemble type algorithm might be more effective. Other candidate methods like the "Mean Bound Contribution" method had results that were easier to trace and defend, with arguably less complexity than a GBT

regression-based method. Further feature engineering and validation are identified as areas for future research opportunity.

8.3 Actual Method: Mean Bound Contribution

The Method used in these research calculations is a method named the "Mean Bound Contribution" method. This method takes the mean of a reasonable CTD window and assumes that any commits that that place within this distance of each other, were worked on relatively continuously, or at least "continuous enough" to fit within a window of tolerance for the application. E.g., a 3-hour CTD window like the one used in these examples, might offer an approximately +/- 3-hour margin of error when performing imputation of the Model Contribution Rate and estimating the hours.

8.3.1 Method

- Complexity Based Contribution Rates, greater than zero and with a CTD of > 1 and < 5 hours are collected.
- The IQR between the 25 – 75 percentile is taken.
- An upper bound and lower bound of 1.5 * IQR are determined. Commits with rates that are outside of these upper/lower bounds are discarded.
    - Future work might include assigning the upper/lower values to instances that fall outside of the range to further reinforce the weights of these occurrences.

8.3.2 The Model Complexity Rate (mCR)

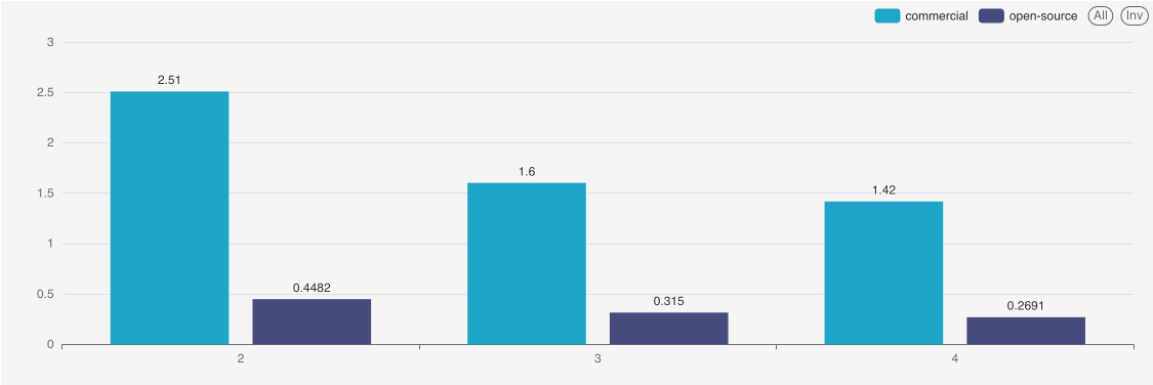

Figure 22: Model Complexity Rate Buckets, 2-4 Hour CTD

Figure 22: Model Complexity Rate Buckets, 2-4 Hour CTD shows an example of mCR values per hourly CTD bucket for commercial vs. open-source repositories. To arrive at an mCR value to be imputed across "unbound" commits, the mCR for the 2-, 3-, and 4-hour CTD buckets would be averaged to arrive at a single rate (A delta of approximately 1.84 complexity units per hour for commercial repositories, according to this example).

8.3.3 Scattered Results

Representing imputed Complexity Rates on a scatter chart visualizes the distribution of values and draws attention to outliers. Following prior analysis (See: Figure 8: Complexity Rate Distribution Buckets >) it can be observed that CR values are heavily skewed towards lower values. The following charts demonstrate all values within the mCTD range and go on to visualize this slice of rates with outliers filtered.

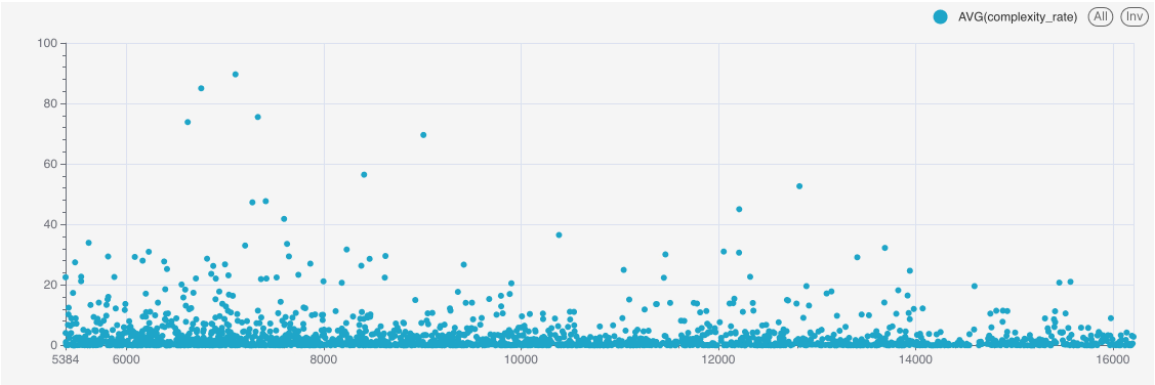

Figure 23: Complexity Rates > 1 and < 5 CTD.

Figure 23: Complexity Rates > 1 and < 5 CTD shows Complexity based CR values that occur in the 2-, 3-, and 4-hour (rounded) CTD window. This CTD window is the "model" (mCTD) window from which CR values are referenced for imputation. In this chart, outliers are not filtered and both Commercial and Open-Source values are included.

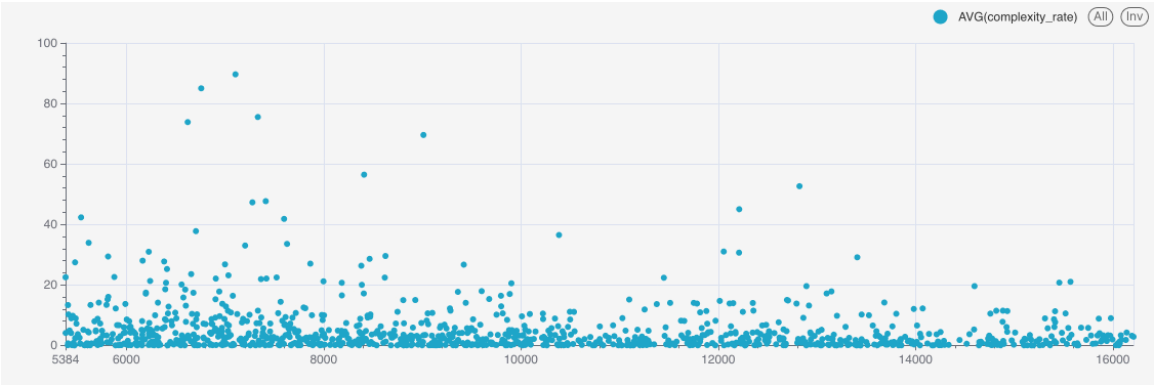

Figure 24: Complexity Rates > 1 and < 5 CTD Commercial Only

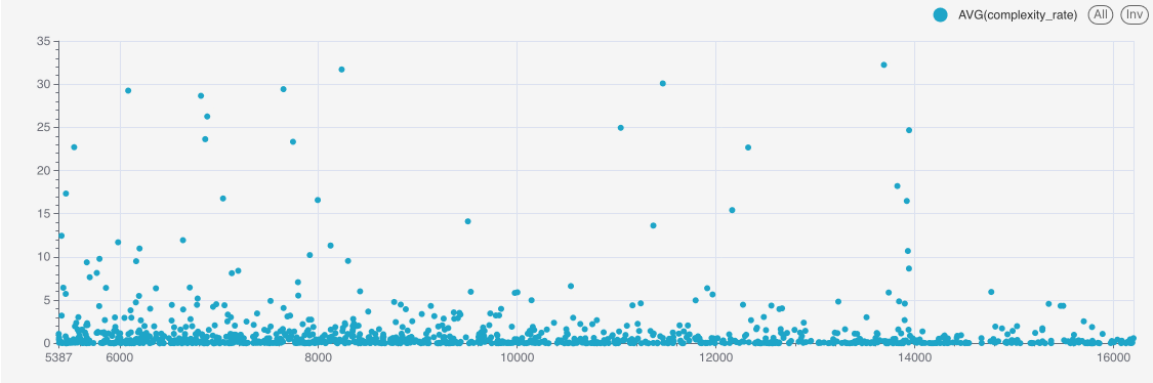

Figure 25: Complexity Rates > 1 and < 5 CTD Open-Source Only

Figure 24: Complexity Rates > 1 and < 5 CTD Commercial Only and Figure 25: Complexity Rates > 1 and < 5 CTD Open-Source Only show the same mCTD window but divide values from Commercial and Open-Source repositories into different charts to visualize nuances between the two populations.

Figure 26: Complexity Rates > 1 and < 5 CTD, Filtered Outliers, Commercial Only depicts complexity rates with a CTD of 2, 3, or 4 hours (rounded to the nearest hours) falling within 1.5 * IQR (1.5 * 5.185 = 67.7775 max rate)

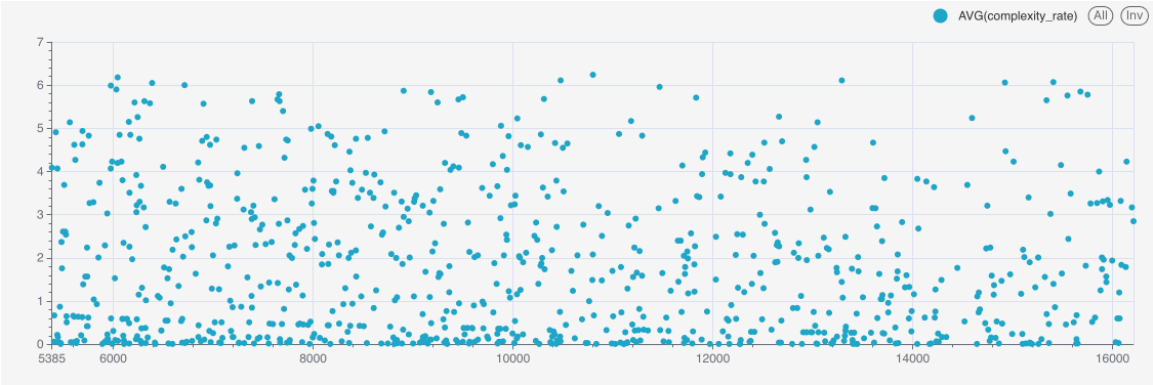

Figure 26: Complexity Rates > 1 and < 5 CTD, Filtered Outliers, Commercial Only

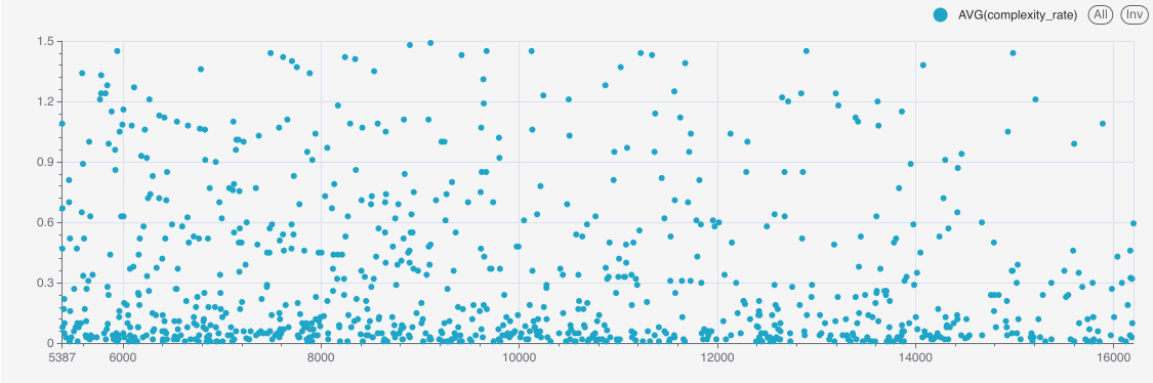

Figure 27: Complexity Rates > 1 and < 5 CTD, Filtered Outliers, Open-Source Only

8.3.4 Issues

Mean Bound Contribution outlines a computational approach for evaluating developer contributions based on commit timing and complexity, leveraging a concept called the "Mean Bound Contribution" method. While innovative, this methodology encounters several noteworthy issues, primarily around its assumptions and potential for imprecision. The method's reliance on an average Commit Time Delta (CTD) window to infer continuous work effort introduces a significant margin of error, potentially misrepresenting the actual developer behavior. This is particularly problematic as it assumes a uniformity of work patterns across different commits without accounting for the variability inherent in software development tasks.

Further compounding the method's limitations is its procedure for handling outliers. By discarding commits that fall outside predetermined upper and lower bounds, valuable data regarding atypical but potentially insightful developer contributions might be lost. While future research opportunities do include the possibility of better addressing these outliers, the current methodology's exclusion of such data points may skew the results, offering a potentially oversimplified view of contribution complexities.

The method's derivation of a Model Complexity Rate (mCR) from averaged mCR values across different CTD buckets (as shown in Figure 22) risks oversimplification. This averaging process may not accurately capture the nuanced differences in complexity across different time frames or between commercial and open-source repositories, as suggested by the distinct charts for each repository type in Figure 24 and Figure 25.

Lastly, the study acknowledges the method's approximate nature and its potential to inaccurately represent developers' contributions. This admission underscores the method's limitations in precisely capturing and reflecting the multifaceted and dynamic nature of software development activities. The presentation of scattered results, intended to visualize the distribution of complexity rates and identify outliers, serves to highlight these methodological challenges further. Without a more nuanced approach to accounting for the wide range of developer behaviors and commit complexities, the "Mean Bound Contribution" method may offer only a partial, and at times misleading, picture of developer contributions.

8.4 Conclusions

Contribution Rate Imputation is a novel method for attributing developer efforts in a more accurate and consistent manner, particularly when standard contribution metrics fail to capture the continuous nature of work. The study presents a rationale behind the need for imputation, proposing that increasing Commit Time Delta (CTD) windows often lead to diminishing Contribution Rates, thereby necessitating an imputation methodology.

The exploration of the Model Contribution Rate (mCR) hypothesis and its applications in Gradient Boosted Tree (GBT) and Mean Bound Contribution Rate (mBCR) methods provides a foundational strategy for addressing issues around quantifying developer performance. Despite the potential of the Gradient Boosted Tree Regression as a predictive tool, its practical

application encountered challenges in producing defensible and traceable results. This ultimately led to its dismissal in favor of the mBCR method.

The Mean Bound Contribution method, which was ultimately chosen, stands out for its simplicity and the defensible nature of its imputed rates. This method leverages a defined CTD window to approximate continuous developer efforts and compensates for a margin of error inherent in the imputation process. Although this method represents an advancement in the field, the study acknowledges the potential imprecision of its assumptions and the elimination of outliers as areas ripe for future inquiry.

The study also acknowledges that exceptions and anomalies in the data are significant and should be integral to any rigorous Systems Engineering approach. When evaluating developer contributions through computational methods such as the mBCR method, it's crucial to consider not only the average or expected patterns of behavior but also the deviations from these patterns. Anomalies may indicate unique and innovative work approaches, attempts to game the system, or data collection errors. Each of these has implications for the trustworthiness of the data and, by extension, the people, and the organization.

Incorporating this perspective into the assessment process enriches our understanding of developer behavior. It may reveal systemic issues, highlight areas where the model's assumptions do not hold, or uncover novel patterns of effective performance. Such insights can inform the refinement of the model, making it more robust and reflective of the true complexity of software development activities.

Moreover, by not discarding these outliers but instead analyzing them to understand their nature, the model can become more inclusive of the diverse ways in which work is conducted

and contributions are made. This approach recognizes that innovation often occurs at the margins, outside the central tendencies of data.

In conclusion, while the Mean Bound Contribution method provides a structured approach to impute contribution rates, it is the nuanced analysis of all data—including exceptions and anomalies—that truly enhances the model's utility and reliability. This not only aids in crafting a more accurate picture of individual contributions but also in fortifying the trust in the data and the systems that rely on such metrics. Thus, these anomalies should not be merely seen as outliers to be discarded but as valuable data points that can lead to a deeper understanding of the developer contributions and the health of the organizational ecosystem in which they occur.

CHAPTER 9: EVALUATING IMPUTED CONTRIBUTION RATES

In this chapter the study explores imputed contribution rates and the estimation of hours contributed by developers in the realm of software engineering. This evaluation is crucial as it serves to validate the robustness and reliability of metrics used to gauge the contributions and productivity of software developers.

Given the impracticality of direct observation, the study has resorted to post hoc analytical methods to assess the fidelity of the estimated hours derived from imputed contribution rates, along with the Time-Delta method employed for rate determination. The methods employed a "Sanity Check" to evaluate estimated hours, and then compare the statistical significance between actual vs. estimated hours.

9.1 Sanity Checking Estimated Hours

Being able to estimate the hours that a developer spends on a commit unobserved could be a very useful software engineering management tool. Estimating time spent on commits opens the ability to estimate the time spent by an individual or a team during a period. With this estimate, the productivity of a team could be evaluated against expected time values. Proving the accuracy of estimated hours is left as an opportunity for future research.

9.1.1 Calculation Method

- Commits that have a CTD Hours (rounded) value of > 4 are selected.
- The Contribution is divided by the imputed CR rate.
- The calculation can be applied with Complexity and/or Levenshtein based CR values.
- For experiments in this exploration, mean imputed CR values were used.

9.1.2 Complexity vs. Levenshtein Sanity Check

Estimated commit hours can be useful as a form of sanity check for CR values. If the estimated commit hours are longer than the CTD, or longer than a developer would have reasonably spent on the commit then the CR is incorrect. This approach provides an alternate way to evaluate methods used to arrive at the CR value.

To demonstrate the difference in quality difference between Complexity and Levenshtein based CR values, estimated hours calculated from both measures are compared against commit CTD values. If estimated hours from the CR value (Complexity of Levenshtein based respectively) are greater than the CTD value, the commit is counted in the total as “unnatural”. Should one method, Complexity of Levenshtein have more unnatural instances, then one contributing factor could be the accuracy of the underlying method used to measure contribution.

Exceptions

The presumption is that the CTD value is the maximum actual time period that a commit could be worked on. Of course, there are exceptions to this such as if a developer works on more than one commit at a time, and stashes the results of one, the CTD measure would not be accurate in those cases.

The consideration of exceptions in data underscores the inherent complexity of accurately measuring contributions in software engineering. This complexity is amplified by the potential for individuals to manipulate contribution metrics for personal benefit, thus compromising the integrity of the data.

From a Systems Engineering perspective, acknowledging, and analyzing exceptions is critical, not solely for refining the reliability of CR calculations but also for evaluating the overall trustworthiness of the data, the contributors, and the organization itself. The “Sanity

Check" described herein serves as a methodological safeguard to validate the estimated hours against actual hours worked, which could potentially reveal discrepancies and unnatural patterns in the data.

When anomalies are detected, they should not be dismissed as mere outliers; instead, they provide valuable insights into the operational dynamics of the software development process. By meticulously documenting and understanding these data anomalies, an organization can develop a better understanding of developer behaviors, thereby enhancing the robustness of its management tools and the fidelity of its metrics.

When assessing organizational trustworthiness and reliability, the consistent application of such sanity checks, coupled with the investigation into exceptions and gaming attempts, could reveal systemic issues within the organization's culture or processes that may require attention. Thus, exceptions to standard CR calculations, rather than being viewed as a hindrance to data purity, should be seen as an opportunity to improve Systems Engineering practices and to reinforce the integrity of software engineering management.

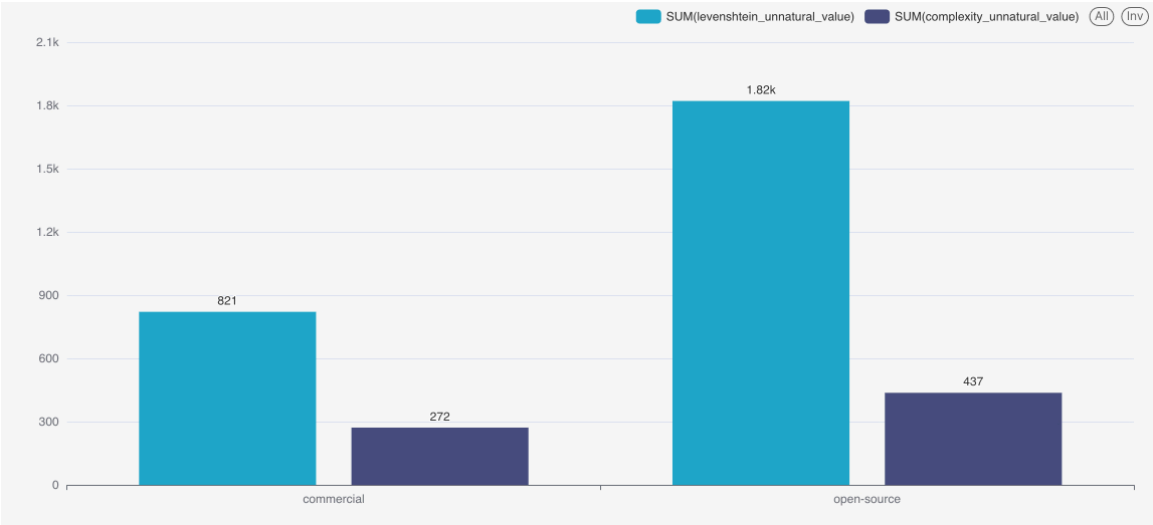

Figure 28: Count of Levenshtein vs. Complexity based Unnatural Contributions in commercial and open-source repositories.

Figure 28: Count of Levenshtein vs. Complexity based Unnatural Contributions in commercial and open-source visualizes commits in the dataset that had hours imputed that totaled more than the commit's CTD value. Because imputed CR values are only used to estimate hours on commits with a CTD hour value (rounded) of > 4, these results include CTD hour values of 5 or greater. In both cases of commercial or open-source repositories, the count of commits with an unnatural number of estimated hours is greater when using Levenshtein based CR values. One likely explanation for this difference is the fact that Complexity based CR values count actual logic paths added, and Levenshtein based CR values only consider textual change. This means that things like automated refactoring or other non-code changes could be included in the developer's contribution calculation, and therefore CR value. A more accurate measure of contribution will yield a more accurate CR value for the time a developer spends contributing code. A more accurate CR value will lead to more plausible results when estimating commit hours.

The following charts demonstrate a measurable difference in those commits whose estimated hours exceed the CTD value on the commit. The values reported in these charts come with some qualifications, first these values only represent "non-model" commits, that is, commits with a CTD hour (rounded) value of < 4 hours. This is important to remember when interpreting the values, as model contribution values are assumed to have been worked on for the entire CTD duration.

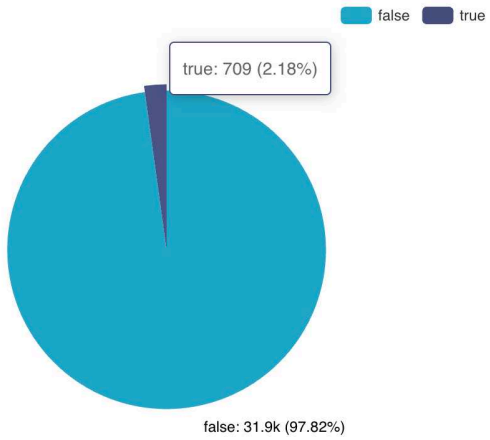

Figure 29: Unnatural vs. Natural Complexity Based Hour Commit Estimates

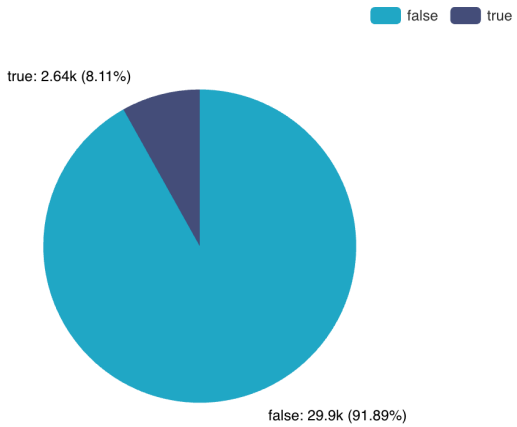

Figure 30: Unnatural vs. Natural Levenshtein Based Hour Commit Estimates

Figure 29: Unnatural vs. Natural Complexity Based Hour Commit Estimates details that for estimated hours using an imputed Complexity based CR value, only 2.18% of those commits had estimated hours that exceeded their CTD values. This contrasts with Figure 30: Unnatural vs. Natural Levenshtein Based Hour Commit Estimates which reports 8.11% of commits had estimated hours that exceeded their CTD value.

9.1.3 Select Results

Scatter charts provide a visualization of commits with Estimated Hours and their position relative to others in the population. Illustrated in these charts are commits with imputed values, i.e. commits with a CTD hours (rounded) value of > 4 hours.

Estimated Hours & Outliers

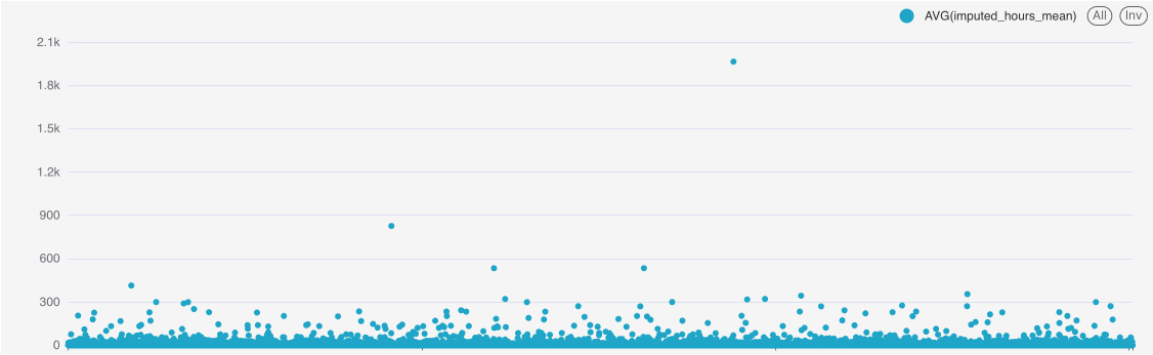

Figure 31: All Estimated Hours, with Outliers

Table 17: All Estimated Hours, Dataset Facts

Measure	Value
MIN	0
MEAN	7.68
MAX	1,964.71
Q1	0.27
Q3	5.748
IQR	5.47
Outliers	IQR * 1.5 = 5.47 * 1.5 = 8.205

Figure 31: All Estimated Hours, with Outliers reflects that the hours imputed are right skewed such that most Estimated Hours are clustered towards lower estimated hour values. This macro view is provided to illustrate the extreme outlier values. Appendix B: Table 20 provides an analysis of commits with greater than 300 hours imputed. Most of the Estimated Hours for these outliers are probably inaccurate. Almost all of the commits detailed in Table 20

are the result of merges or otherwise unnatural and wholesale introduction of complexity values into the codebase. Future development efforts will likely introduce heuristic or other filtering methods to better identify unnatural conditions. There exists the opportunity to implement a hybrid Levenshtein/Complexity based mechanism for identifying outliers and better estimating hours for those contributions that exhibit abnormally high CD values.

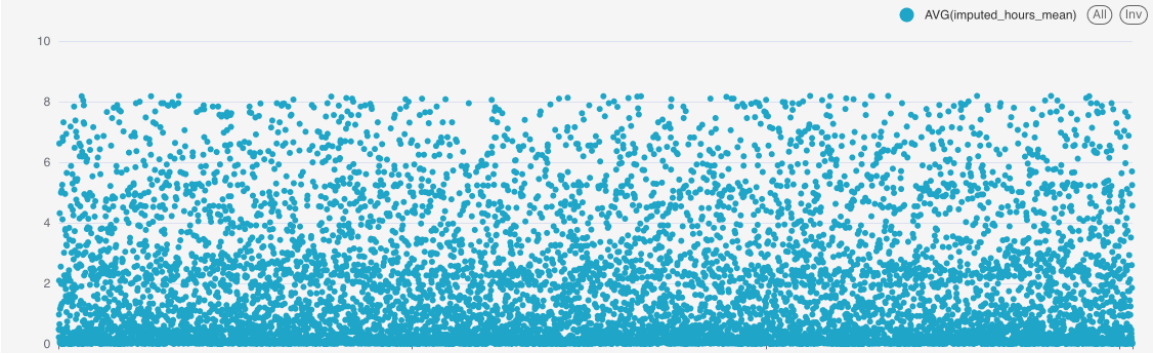

Figure 32: Estimated Hours, No Outliers

Figure 32: Estimated Hours, No Outliers illustrates the population of commits with estimated hours with the "Outliers" quoted in Appendix Table 20 omitted. The fact that the IQR of commits with estimated hours falls within a comparable range to CTD values observed in the entire dataset population does not contradict the potential accuracy of these estimations (See: Figure 5: CTD < 8 Hours Bucket Distribution).

Both Commercial and Open-Source repositories exhibited similar inter quartile ranges. Anecdotally, some of the differences observed between Commercial and Open-Source might be explained by more consistent and longer programming sessions being present in the Commercial dataset. It may be more likely that Open-Source contributions are contributed by a developers' own volition and not mandated by a commercial setting. For this reason, programming sessions may be characteristically shorter. Bosu et al. (2017) offers that there could

be present process differences between open source and commercial software teams such as code review practice that could influence work sessions and contribution size [45].

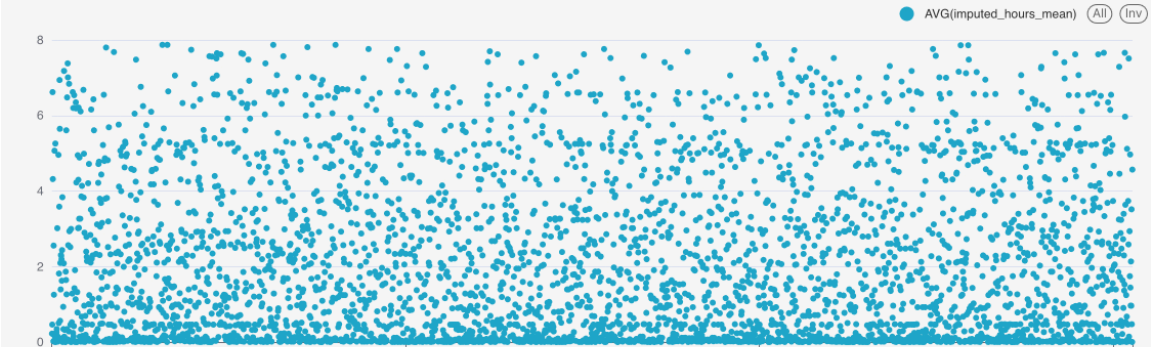

Figure 33: Estimated Hours, No Outliers, Commercial Only

Table 18: Commercial Estimated Hours Dataset Facts

Measure	Value
MIN	0.1
MEAN	5.17
MAX	169.83
Q1	0.41
Q3	5.25
IQR	5.47
Outliers	IQR * 1.5 = 5.25 * 1.5 = 7.875

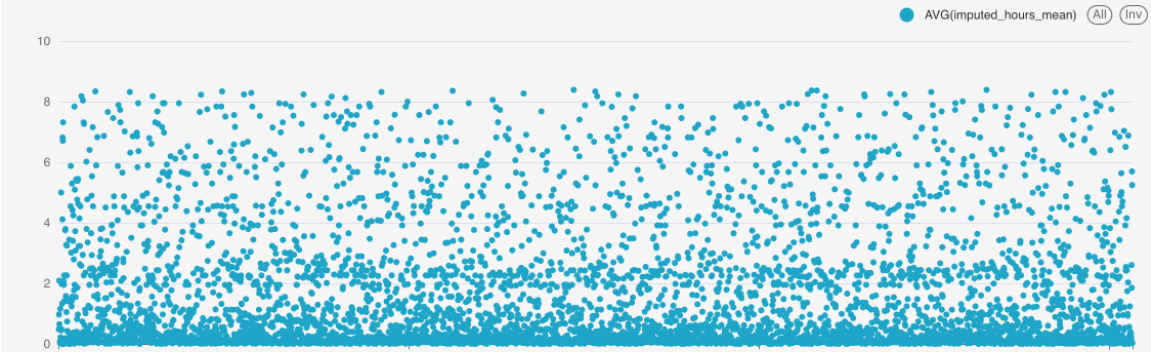

Figure 34: Estimated Hours, No Outliers, Open-Source Only

Table 19: Estimated Hours Dataset Facts

Measure	Value
MIN	0.01
MEAN	9.59
MAX	1964.71
Q1	0.24
Q3	5.86
IQR	5.62
Outliers	IQR * 1.5 = 5.62 * 1.5 = 8.43

Estimated Hours Distribution

The distribution of Estimated Hours is comparable to the distribution of commit CTD values observed in the dataset. The majority of instance falling into the 0-hour range mirrors well the observed CTD values in the overall dataset population (See: Figure 5: CTD < 8 Hours Bucket Distribution).

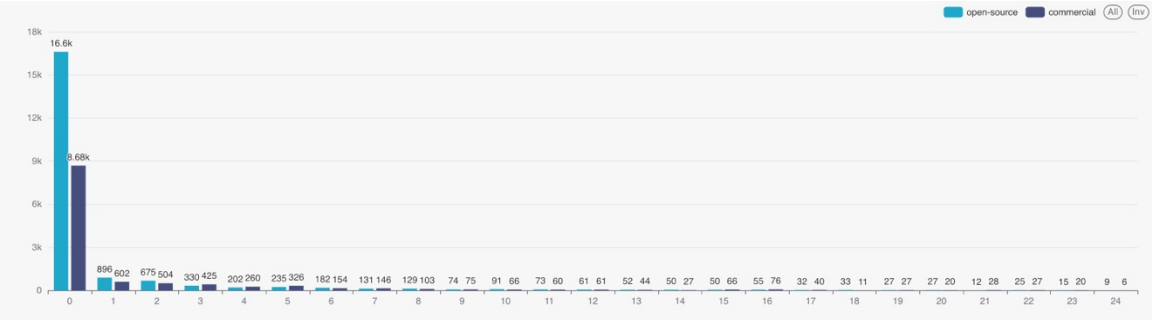

Figure 35: Unique Commits with Less Than 24 Estimated Hours

Figure 35 displays a series of hours that looks visually similar to the temporal series of commits counts shown in Appendix Figure 43. The population is right-skewed in a similar way, where a disproportionate number of commits are estimated to fall in the 0–1-hour CTD.

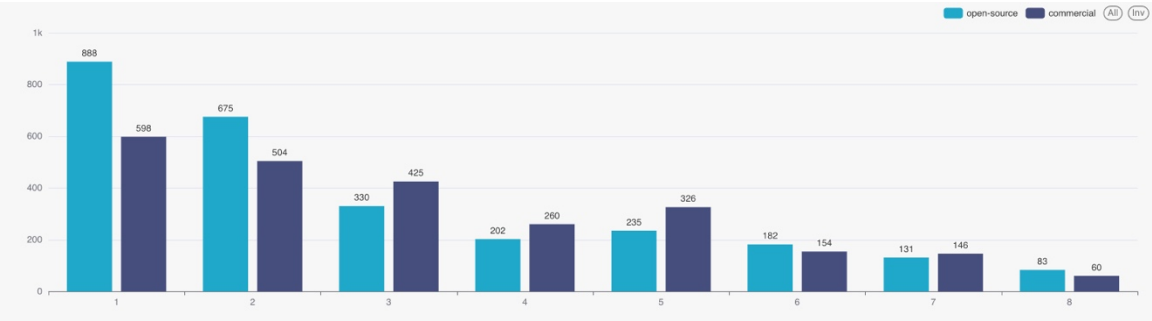

Figure 36: Unique Commits, Greater than 0 and Less Than 8 Estimated Hours

By omitting the hours reported in the 0-1-hour CTD, the correlation between actual CTD values and estimated hours in Figure 36 becomes more visually apparent. See Figure 5: CTD < 8 Hours Bucket Distribution for a comparison.

9.1.4 Sanity Check Conclusions

The parallel between observed CTD values and hours estimated from imputed contribution rates has prompted a closer scrutiny of the statistical significance between the distributions of both populations. The comparison between Complexity and Levenshtein based CR values against CTD hours indicates a marked difference in the reliability of estimated unnatural contributions, with Complexity-based CR values exhibiting greater fidelity. This enhanced reliability is quantitatively supported by a lower incidence of unnatural estimates in Complexity-based assessments. The correlation between the distribution of estimated hours and the CTD values further corroborates the validity of the estimation method. Although the evidence is not conclusive, the alignment of commit behavior with estimated hours lends credibility to the potential of the estimation techniques. The subsequent section expands on this topic by examining the statistical significance and exploring the accuracy of the underlying

techniques employed to estimate contribution hours, presenting an intriguing technique to evaluate the quality of resulting CR and estimated hour measures.

9.2 Statistical Significance of Estimated Hours

The study investigated the statistical significance between estimated hours and actual CTD measures, prompted by their correlation detected while sanity checking estimated hours. When compared to actual hours, those estimated hours for commits beyond the mCR range were also scrutinized for their statistical significance against actual CTDs within the mCR. A finding of significance would suggest a uniform distribution of developer effort over time. Neither the actual nor the estimated hours conformed to a normal distribution, as established by the Shapiro-Wilk [46] [47] [48] and D'Angostino-Pearson [49] [50] [51] tests, thereby negating the use of ANOVA in our results. The investigation instead applied the Mann-Whitney U and Kruskal-Wallis H tests to determine significance, with a recognized p-value threshold (alpha) of 0.05 or more. Initially, the results were inconclusive; however, through iterative recalibrations of the mCR calculation method, we obtained statistically significant values. These findings provide a new potential for tuning the mCR calculation, enhancing the robustness and credibility of the estimated hours.

9.2.1 Experiment-1: Initial Tests

In Experiment-1, the focus was on assessing the statistical significance between actual versus estimated hours of work. A CTD window between 2-4 hours was used to select the experiment population. Only contribution rates (CR) that were above the first interquartile range (IQR1) and below the third interquartile range (IQR3) were included in the analysis to avoid outliers affecting the central trend of the data.

Figure 37 illustrates the distribution of contribution rates within the specified interquartile ranges across the CTD windows tested. This visualization highlights how contributions are dispersed across the central portion of the data and how they differ as the CTD window changes.

Figure 38 displays the outcomes in terms of statistical significance for the repositories under consideration. The results indicates that there is promising statistical significance for half of the repositories examined, suggesting that for these repositories, the difference between actual and estimated hours is not due to random chance, thus, the estimates may be considered reliable.

It should be noted that 'promising statistical significance' implies that the results are likely to have a low probability of occurring by chance, and thus, might reflect true differences or effects being tested. This could be indicative of the potential utility of the model or method being used to estimate hours for software development tasks within the selected repositories. Further investigation would be warranted to confirm the results and potentially refine the estimation model.

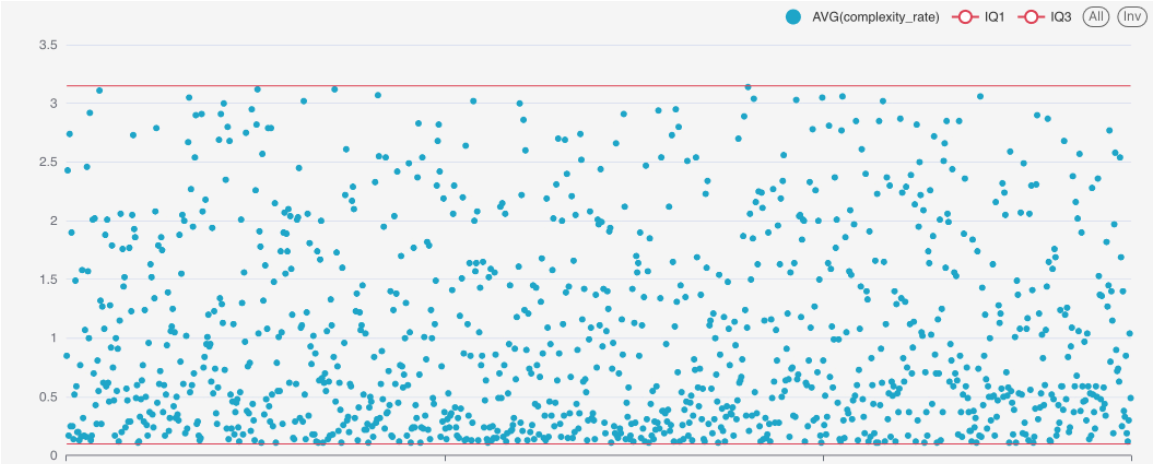

Figure 37: Experiment-1 IQR Contribution Rates, 2/3/4 Hour CTD

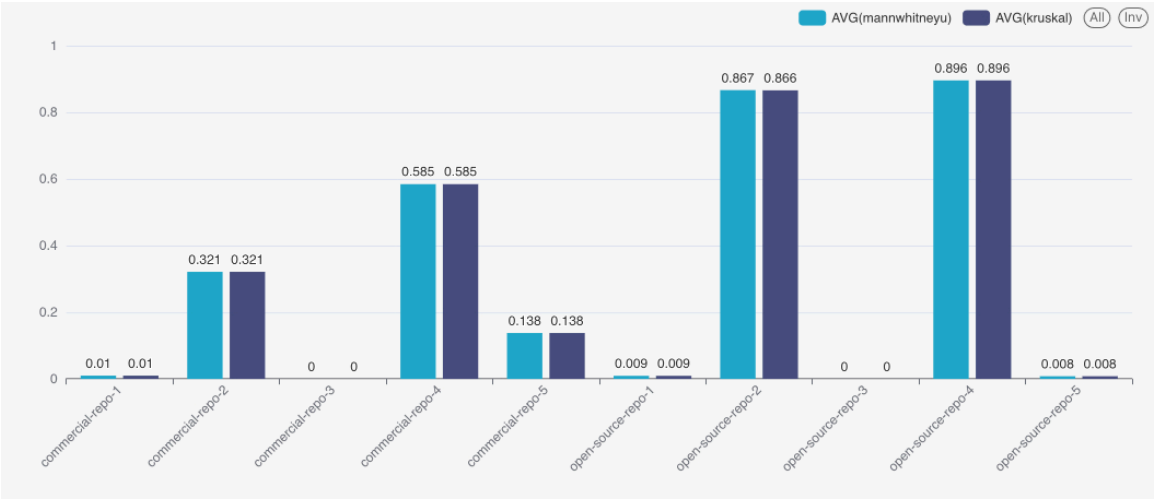

Figure 38: Experiment-1 Repository Statistical Significance

9.2.2 Experiment-2: Narrowing the model CTD Window

Experiment 2 aims to refine the prediction of developers' work contributions by focusing on a specific 2-hour CTD window. The experiment involves narrowing down the range of CR values to those between the first and third interquartile ranges (IQR1 and IQR3), effectively excluding outliers. The underlying hypothesis is that by concentrating on a 2-hour CTD, the imputed contribution rates and resulting estimated hours will more accurately reflect a developer's consistent work patterns.

Figure 39 visualizes contribution rates within the specified 2-hour CTD window, with data points representing instances that fall within the interquartile ranges. The goal is to identify the central tendency and dispersion of contribution rates during this timeframe.

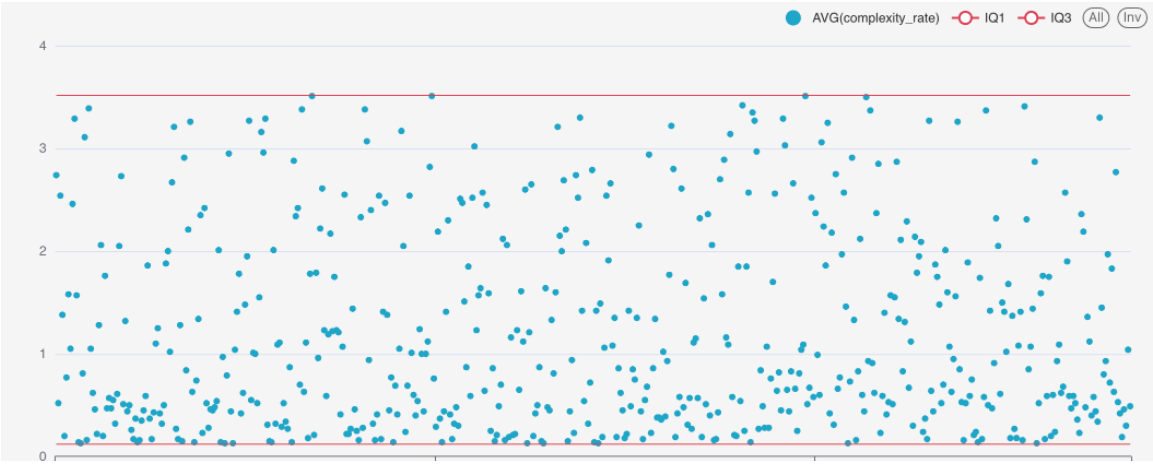

Figure 39: Experiment-2 IQR Contribution Rates, 2 Hour CTD

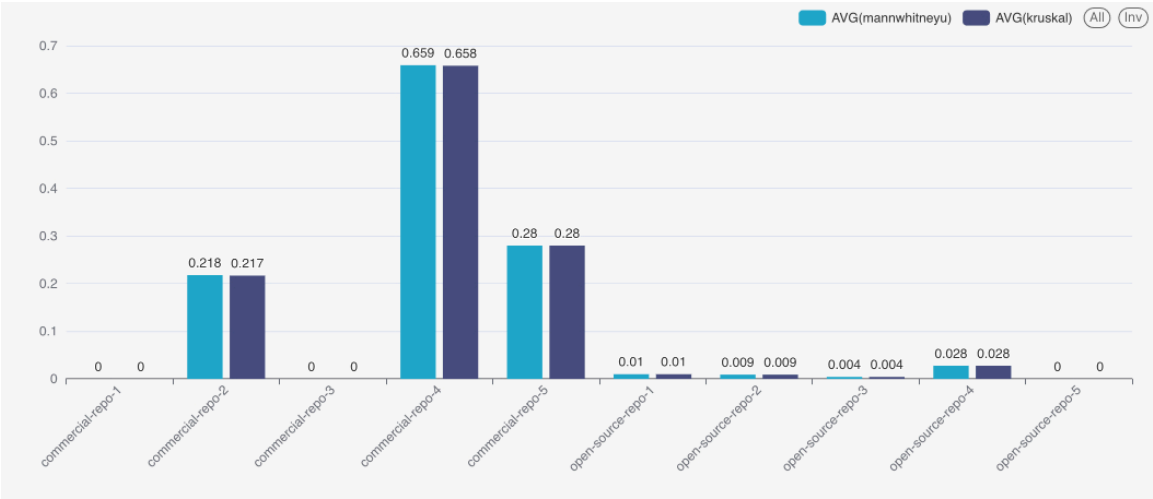

Figure 40: Experiment-2 Repository Statistical Significance

Figure 40 displays the statistical significance of results across different repositories or datasets. Bars represent the p-value of tests run to assess the hypothesis. Shorter bars with a p-value of less than 0.05 indicate a lack of statistical significance, whereas taller bars indicate stronger evidence that the narrowed CTD window provides higher statistical significance.

The expectation is that by narrowing the CTD to only the 2nd hour window and focusing on the middle 50% of data regarding contribution rates, the study can eliminate extreme values

that could skew the analysis, leading to more reliable estimates of work patterns that align with actual measured CTD values. Unfortunately, this was not the case with Experiment-2 results, and statistical significance did not improve. While these parameters did not produce the statistical significance that the study hoped for, it was suspected that by further focusing the window to only consider higher CR values that results might improve. This presumption lays the groundwork for Experiment-3 that follows.

9.2.3 Experiment-3: Focusing the mCR Range.

Experiment-3 is designed to refine the analysis of coding contributions by setting a specific interquartile range (IQR) threshold for CR values. The experiment builds upon a previous experiment, Experiment-2, by using the same 2-hour CTD window but adjusting the criteria for selecting data points to analyze.

Experiment-3 considers CR values that are greater than the second IQR (IQR2) and less than the third IQR (IQR3). This filtering narrows down the range to higher CR values that occurred during the 2-hour CTD window when compared to Experiment-2. The hypothesis is that by isolating the higher CR values, the analysis is more likely to capture work sessions that were sustained over a larger portion of the CTD window. Such higher CR values would suggest a greater 'complexity delta', which implies that the developer spent less time on non-coding activities and more time on actual coding tasks during the 2-hour window.

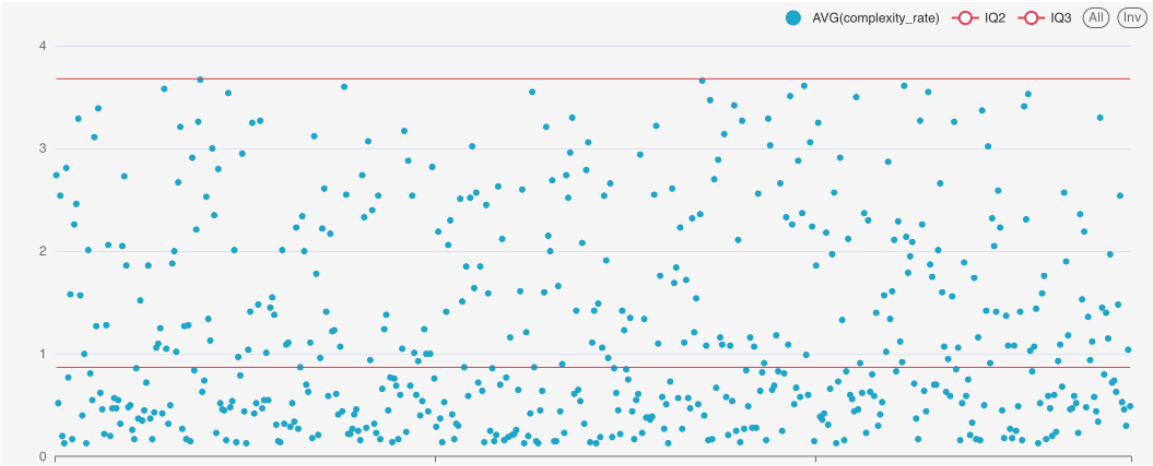

Figure 41: Experiment-3 IQR Contribution Rates, 2 Hour CTD

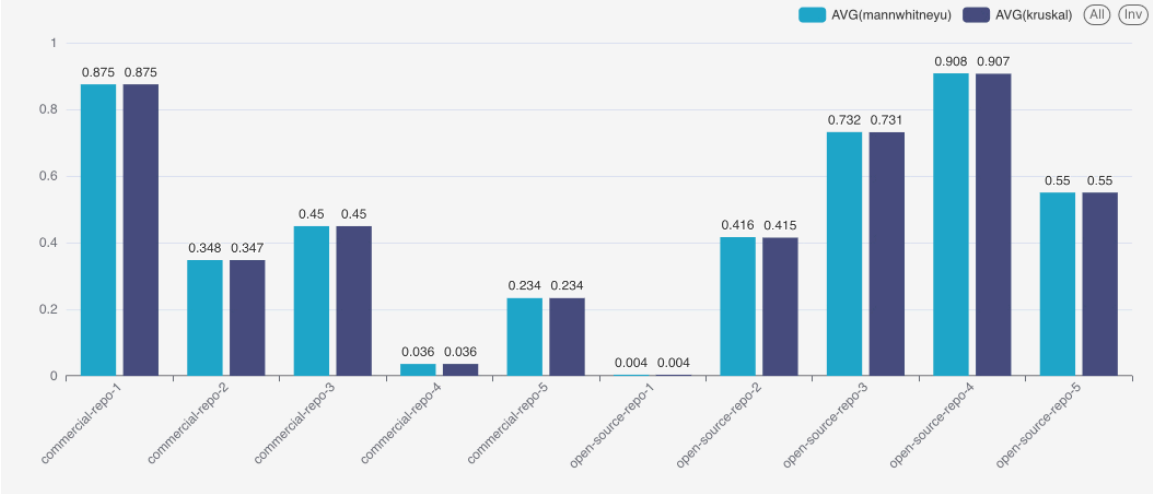

Figure 42: Experiment-3 Repository Statistical Significance

Figure 41 shows the distribution of IQR Contribution Rates for the 2-hour CTD window in Experiment-3. Figure 42 shows that the statistical significance of actual CTD values measures and estimated hours from imputed contribution rates increased as a result of focusing the mCR window to rates representing higher complexity deltas.

Experiment-3 establishes a correlation between higher CR values and prolonged coding sessions during the CTD window by demonstrating that if these more continuous and

uninterrupted values are used to estimate commit durations, that the time distribution of those estimations exhibit more statistical significance when compared to those values actually measured.

Sanity Check Contradictions

In the interest of full disclosure, the study will report that although the statistical significance reported in Experiment-3 are very promising, when the same imputed hours are run through the tests in section 9.1 Sanity Checking Estimated Hours, the results actually don't improve. This is demonstrated in Appendix Figure 44 where results show that unnatural estimates for commit hours increased to 4.94% from 2.18% (See: Figure 29: Unnatural vs. Natural Complexity Based Hour Commit Estimates). It is entirely possible that the rate of unnaturally estimate commit hours did increase, but the commit hours that were not estimates at unnatural values, did indeed more accurately represent a developer's behavior. Before declaring that Experiment-3 delivers improved results, any study would be obliged to continue this line of experimentation and reconcile the difference between the statistical significance of actual vs. estimates hours, and the sanity check results for those estimated hours. While further raising and narrowing the mCR window to achieve greater accuracy in contribution rate imputation is an opportunity left for future research, the study does have some speculation as to why the sanity check tests for estimate hours indicated less accuracy. It's possible that by increasing the mCR value used in Experiment-3, that the values for the 2-hour CTD used were indeed more statistically significant. It is also possible that the values taken from this narrow and low window do not correspond correctly to every commit complexity delta when attempting to estimate hours. This leads the study to believe that further refinement of the CRIM could involve an array of mCR values intelligently assigned based on the size of the complexity delta measured in the

commit. However, this study endeavors only to establish a method for measuring software development contribution rates, and not proving the estimated hours resulting from the CRIM. While the results of Experiment-3 are inconclusive, the fact the experiments described in this chapter were possible and yield results at all, support the Time-Delta method for calculating software development contribution rates.

9.2.4 Statistical Significance Conclusions

The study's analysis on the statistical significance of estimated hours against actual CTD measures has yielded noteworthy insights. Despite initial challenges due to the non-normal distribution of the data, which precluded the use of ANOVA, alternative non-parametric tests, namely the Mann-Whitney U and Kruskal-Wallis H, have successfully been employed. These tests revealed significant results following iterative adjustments to the mCR calculation method. Experiment-1's preliminary investigation provided a baseline for significance across various CTD windows, indicating a substantial variance in contribution rates when looking at a broader spectrum of estimated vs. actual hours.

Further refinement in Experiment-2, by narrowing down the CTD window to a more precise 2-hour frame, suggested that a tighter focus yields estimated hours that better reflect the actual distribution of developer effort, enhancing the fidelity of the estimated hours to actual work patterns.

Experiment-3's focus on a higher IQR range within the same 2-hour CTD window lent credence to the hypothesis that higher contribution rates are indicative of more continuous coding activity and complexity, leading to a more accurate estimation of developer hours.

These findings underscore the value of fine-tuning estimation methods for developer hours, which could significantly improve the accuracy of these methods. The statistical

significance obtained by controlling for the mCR suggests a pathway to more accurate representations of developer effort.

The study's commitment to integrity necessitates reporting that the significant results of Experiment-3 are contradicted by subsequent sanity checks. As detailed in Appendix Figure 44, when the imputed hours derived from Experiment-3 were subjected to these checks, they exhibited an increase in unnatural estimate rates, which does not corroborate an improvement in estimating actual developer behavior. This discrepancy between the observed statistical significance and the results of the sanity checks on estimated hours indicates that while the mCR may capture statistical significance, they may not universally translate to more accurate developer hour estimations.

The emerging hypothesis is that the array of mCR values may need to be dynamically calibrated to the complexity delta of each commit to refine the CRIM. This nuanced approach could potentially reconcile the significant statistical findings with the practical application of estimating developer effort. While Experiment-3 has provided inconclusive results regarding the absolute accuracy of developer hour estimations, the overall research supports the potential of the Time-Delta method in calculating software development contribution rates. Therefore, future investigations should focus on experimenting with a spectrum of mCR values and further scrutinizing their alignment with commit complexity to validate the CRIM's effectiveness.

CHAPTER 10: CONCLUSIONS

This chapter consolidates the significant conclusions derived from the study. The sections that follow correlate with the findings and implications detailed in the respective chapters of the dissertation, offering a structured synthesis of the contributions made to the body of knowledge concerning software development performance management.

10.1 Time Delta Method: Calculation and Distribution

The Time-Delta Method hinges on the Commit Time Delta (CTD), calculated by measuring the time elapsed between consecutive commits by the same author in a repository. It revealed a bias toward shorter CTD values, suggesting developers' propensity for making quick, focused changes. This method proved useful for estimating the duration a developer may have spent on tasks associated with commits, particularly when the CTD is short, indicating continuous work.

10.2 Cyclomatic-Complexity Based Contribution

This study established Cyclomatic Complexity-based Contribution as an accurate method for quantifying the size of a software commit. Cyclomatic Complexity offers a more stable and accurate reflection of a developer's behavior than other methods by focusing on the logical complexity of code changes, rather than mere textual changes (e.g., via LOC or Levenshtein Distance).

10.3 Complexity-Based Contribution Rate

The Contribution Rate (CR) analysis centered on the Complexity-based Contribution Rate, highlighting its superiority over the Levenshtein Distance-based rate due to its focus on the logical changes in code. The findings indicated that CR values decrease as the CTD increases, aligning with expectations that shorter intervals between commits are associated with more concentrated work efforts.

10.4 Contribution Rate Statistical Significance

This study evaluated the statistical significance of CR values within repositories, employing methods like the Kruskal-Wallis and Mann-Whitney U test. Results demonstrated consistency in CR values' statistical significance, suggesting that the utility of the Time-Delta method for reflecting real-world developer behavior is context-dependent.

10.5 CRIM: Mean Bound Contribution Rate

The chapter introduced the Mean Bound Contribution Rate (MBCR) as a method for estimating CR for commits with lengthy CTDs, where direct observation is not feasible. This approach utilizes average CR values from commits deemed to represent continuous work sessions as a baseline for imputing CR values to other commits.

10.6 Evaluating Imputed Contribution Rates

The final theoretical piece involved evaluating the plausibility and accuracy of imputed CR values. Through sanity checks and statistical analyses, the study affirmed the reliability of estimated contribution rates, underscoring the method's practicality for assessing software developer engagement and productivity.

10.7 Contribution to the Field of Systems Engineering

This study delivers advancement in the field of systems engineering, particularly in the areas of verification and validation, test and measurement, sensitivity analysis, risk assessment, and rate/anomaly detection. By introducing and rigorously evaluating the Time-Delta method and associated metrics for quantifying software development contributions, this research contributes to a deeper understanding of software engineering processes within complex systems. The novel methodologies developed herein address a critical gap in the existing body of knowledge, offering both theoretical and practical insights that can be leveraged to enhance the efficiency and effectiveness of software development teams.

10.7.1 Verification and Validation

The study introduces the Time-Delta Method as a novel framework for assessing software development contribution rates, leveraging CTD measures and software complexity metrics. This method provides a statistical basis for validating developer contributions and productivity, enhancing the verification and validation processes within software engineering projects.

10.7.2 Test and Measurement

Through the empirical validation of the Time-Delta Method across commercial and open-source software environments, the study advances the field's capacity for test and measurement. Also outlined is a rigorous experimental methodology for evaluating software developer contributions, employing statistical techniques to infer productivity and work patterns. This contributes a practical tool for measuring the engagement and efficiency of software developers in a production setting.

10.7.3 Sensitivity Analysis

The study performs a sensitivity analysis on the factors influencing software development contribution rates. By analyzing the statistical significance of contribution rates within software development repositories, the study provides insights into how different metrics (e.g., Cyclomatic Complexity versus Levenshtein Distance-based measures) influence the estimation of developer contributions. This analysis aids in understanding the sensitivity of software project outcomes to various metrics and methodologies.

10.7.4 Risk Assessment

The study contributes to risk assessment in systems engineering by offering a framework to estimate unobserved work durations and to impute contribution rates for commits with long CTDs. This approach, involving novel imputation techniques enables a more accurate estimation of development effort, thereby aiding in the assessment and mitigation of project risks associated with resource allocation and deadline adherence.

10.7.5 Rate/Anomaly Detection

The study advances the capability for rate and anomaly detection by introducing methods to evaluate the statistical significance of estimated hours of work based on imputed contribution rates. This allows for the identification of outliers and anomalies in software development patterns, facilitating the early detection of issues that could impact project timelines and quality.

10.7.6 Improvement in Project Management Practices

The methodologies developed in this study have potential implications for project management practices within systems engineering. By enabling a more accurate estimation of software development contributions, project managers can optimize resource allocation, enhance

workflow efficiency, and make more informed decisions regarding team composition and project timelines. The ability to quantify contributions and contribution rates in a meaningful way allows for the data-driven management of software projects, aligning with the principles of systems engineering that advocate for evidence-based decision-making.

10.7.7 Facilitation of Interdisciplinary Collaboration

This research contributes to bridging the gap between systems engineering and software engineering disciplines. By focusing on the quantification of software development contributions from a systems perspective, the study facilitates interdisciplinary collaboration, promoting a more integrated approach to designing and managing complex software systems. The findings and methodologies introduced herein provide a common ground for systems engineers and software developers to collaborate more effectively, ensuring that software components are developed in harmony with broader system objectives.

10.8 Opportunities for Future Research

The study presents several opportunities for future research in the area of software development performance management. These opportunities span various aspects of software engineering metrics, methods, and implications for practical applications in the field of systems engineering and beyond.

10.8.1 Alternative Methods for Measuring Developer Contribution

The study suggests the exploration of alternative metrics for assessing developer contributions, specifically beyond Cyclomatic Complexity and Levenshtein Distance measures

(e.g., LLM based contribution analysis). This includes the development and validation of new metrics that could offer a more accurate understanding of developer efforts and contributions.

10.8.2 Quantitative Analysis of Contribution Measurement Quality

A comprehensive quantitative analysis comparing different contribution measurement methods, such as Cyclomatic Complexity and Levenshtein Distance, is identified as a research need. This analysis would aim to identify the quality differences between these metrics in capturing meaningful contributions.

10.8.3 Language-Agnostic Tools for Software Performance Analysis

The study points out the limitation of Cyclomatic Complexity being language-dependent and suggests the development of language-agnostic tools that could provide broader applicability in analyzing software performance.

10.8.4 Investigating the Impact of Credit for Reduced Complexity

There exists an opportunity to conduct an in-depth investigation into instances where Cyclomatic Complexity is reduced due to thoughtful programming improvements. Specifically, research could examine how often these instances occur and their impact on complexity-based contribution values, especially in cases where current methods do not credit these reductions.

10.8.5 Dynamic Calibration of Model Complexity Rates (mCR)

Future research could explore the dynamic calibration of mCR values in relation to the complexity delta of each commit. This involves refining the Contribution Rate Imputation Method (CRIM) by experimenting with a spectrum of mCR values according to contribution size in order to achieve more accurate estimations of developer effort.

10.8.6 Exploration of GBT Regression for Contribution Measurement

The study quotes the exploration of Gradient Boosted Tree (GBT) regression as a method for quantifying developer performance. Although GBT showed promise, its practical application faced challenges, highlighting an area for future research to improve or refine this approach.

10.8.7 Evaluation of Imputed Contribution Rates

There is a need for rigorous evaluation of imputed contribution rates, especially in terms of estimating the time spent by developers on commits. This involves validating the accuracy and reliability of methods used to estimate developer hours and exploring alternative approaches for sanity checking estimated hours.

10.8.8 Refinement of Outlier Detection Algorithms

The Time-Delta Method presents a novel approach for estimating software development contribution rates. A critical area for future research lies in the enhancement of outlier detection algorithms. By developing more sophisticated methods to identify and analyze anomalies within CTD windows and complexity metrics, researchers can uncover deeper insights into unusual developer behaviors or project anomalies that could signify potential areas of inefficiency or risk.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN THE WRITING PROCESS

During the preparation of this work the author used the following AI powered tools:

- Amazon Bedrock for summarization.
- ChatGPT for summarization.
- Scite.ai for literature search.

After using these services, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

WORKS CITED

[1] *Apache Spark*. (2023). Apache Software Foundation. [Online]. Available: https://spark.apache.org/

[2] *Amazon EMR Serverless*. (2024). Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/emr/serverless/

[3] *PySpark: Python API for Apache Spark*. (2023). Apache Software Foundation. [Online]. Available: https://spark.apache.org/docs/latest/api/python/index.html

[4] *Git*. (2024). Software Freedom Conservancy, Inc. [Online]. Available: https://git-scm.com

[5] *PyGit2: A Python Git Library*. (2023). [Online]. Available: https://www.pygit2.org/

[6] *libgit2*. (2024). libgit2. [Online]. Available: https://libgit2.org/

[7] *Jupyter Notebooks*. (2024). Project Jupyter. [Online]. Available: https://jupyter.org/

[8] *Amazon Simple Storage Service (S3)*. (2024). Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/s3/

[9] *AWS Athena*. (2024). Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/athena/

[10] *Amazon Redshift*. (2024). Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/redshift/

[11] *Apache Druid*. (2024). Apache Software Foundation. [Online]. Available: https://druid.apache.org/

[12] *PostgreSQL*. (2024). PostgreSQL Global Development Group. [Online]. Available: https://www.postgresql.org/

[13] *Tableau*. (2024). Salesforce Inc. [Online]. Available: https://www.tableau.com/

[14] *Apache Superset*. (2024). Apache Software Foundation. [Online]. Available: https://superset.apache.org/

[15] *Matplotlib*. (2024). Matplotlib Development Team. [Online]. Available: https://matplotlib.org/

[16] C. Kolassa, D. Riehle, and M. A. Salim, "The Empirical Commit Frequency Distribution of Open Source Projects," 2013, doi: 10.1145/2491055.2491073.

[17] C. Gote, I. Scholtes, and F. Schweitzer, "Git2net - Mining Time-Stamped Co-Editing Networks From Large Git Repositories," 2019, doi: 10.1109/msr.2019.00070.

[18] "Histogram Interpretation: Skewed (Non-Normal) Right." NST. https://www.itl.nist.gov/div898/handbook/eda/section3/histogr6.htm# (accessed February, 13, 2024.

[19] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics. Doklady,* vol. 10, pp. 707-710, 1965.

[20] P. E. Black, "Dictionary of Algorithms and Data Structures," 2019. [Online]. Available: https://xlinux.nist.gov/dads/HTML/Levenshtein.html

[21] T. Mende, R. Koschke, and F. Beckwermert, "An evaluation of code similarity identification for the grow-and-prune model," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 21, no. 2, pp. 143-169, 2009, doi: https://doi.org/10.1002/smr.402.

[22] *Claude v 2.1*. (2024). Anthropic. [Online]. Available: https://www.anthropic.com/claude // Replace with the actual URL if different
[23] *Amazon Bedrock*. (2024). Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/bedrock
[24] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering,* vol. SE-2, no. 4, pp. 308-320, 1976, doi: 10.1109/TSE.1976.233837.
[25] D. Landman, A. Serebrenik, E. Bouwers, and J. Vinju, "Empirical Analysis of the Relationship Between CC and SLOC in a Large Corpus of Java Methods and C Functions," *Journal of Software Evolution and Process,* 2015, doi: 10.1002/smr.1760.
[26] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi, "A Massive Analysis of Ethereum Smart Contracts Empirical Study and Code Metrics," *Ieee Access,* 2019, doi: 10.1109/access.2019.2921936.
[27] C. Ebert, J. Cain, G. Antoniol, S. Counsell, and P. Laplante, "Cyclomatic Complexity," *IEEE software,* vol. 33, no. 6, pp. 27-29, 2016, doi: 10.1109/MS.2016.147.
[28] *Lizard - A Code Complexity Analyzer*. (2023). [Online]. Available: https://github.com/terryyin/lizard
[29] "40 WPM Typing Test English." YouTube.com. https://www.youtube.com/watch?v=HzkcR5nYu_E (accessed May, 23, 2023.
[30] "Barbara Blackburn, the World's Fastest Typist." https://en.wikipedia.org/wiki/Barbara_Blackburn_(typist) (accessed May, 22, 2023.
[31] "Typing 200 WPM is Hard." https://www.youtube.com/watch?v=c0Q_23215lI (accessed May 22, 2023.
[32] F. Wen, C. Nagy, M. Lanza, and G. Bavota, "Quick remedy commits and their impact on mining software repositories," (in eng), *Empir Softw Eng,* vol. 27, no. 1, p. 14, 2022, doi: 10.1007/s10664-021-10051-z.
[33] J. Eyolfson, L. Tan, and P. Lam, "Correlations between bugginess and time-based commit characteristics," *Empirical software engineering : an international journal,* vol. 19, no. 4, pp. 1009-1039, 2014, doi: 10.1007/s10664-013-9245-0.
[34] U. Techera, M. Hallowell, N. Stambaugh, and R. Littlejohn, "Causes and Consequences of Occupational Fatigue: Meta-Analysis and Systems Model," (in eng), *Journal of occupational and environmental medicine,* vol. 58, no. 10, pp. 961-973, 2016/10// 2016, doi: 10.1097/jom.0000000000000837.
[35] T. F. Meijman, "Mental fatigue and the efficiency of information processing in relation to work times," *International Journal of Industrial Ergonomics,* vol. 20, no. 1, pp. 31-38, 1997/07/01/ 1997, doi: https://doi.org/10.1016/S0169-8141(96)00029-7.
[36] D. SciPy, "scipy.stats.f_oneway," 2023 2023. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html.
[37] *scipy.stats.kruskal*. (2024). [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html
[38] S. National Institute of and Technology, "How can we compare several populations with unknown distributions (the Kruskal-Wallis test)?," 2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/prc/section4/prc41.htm.
[39] D. SciPy, "scipy.stats.mannwhitneyu," 2023 2023, doi: 10.5281/zenodo.593367.
[40] Y. Zhang and A. Haghani, "A Gradient Boosting Method to Improve Travel Time Prediction," *Transportation Research Part C Emerging Technologies,* 2015, doi: 10.1016/j.trc.2015.02.019.

[41] Y. Li, X. Huang, Y. Li, F. Chen, and Q. Li, "Machine Learning Based Algorithms for Wind Pressure Prediction of High-Rise Buildings," *Advances in Structural Engineering,* 2022, doi: 10.1177/13694332221092671.
[42] X. Zhang, C. Dai, W. Li, and Y. Chen, "Prediction of Compressive Strength of Recycled Aggregate Concrete Using Machine Learning and Bayesian Optimization Methods," *Frontiers in Earth Science,* 2023, doi: 10.3389/feart.2023.1112105.
[43] J. Wang, P. Li, R. Ran, Y. Che, and Y. Zhou, "A Short-Term Photovoltaic Power Prediction Model Based on the Gradient Boost Decision Tree," *Applied Sciences,* 2018, doi: 10.3390/app8050689.
[44] S. Sun, J. Zhang, J. Bi, and Y. Wang, "A Machine Learning Method for Predicting Driving Range of Battery Electric Vehicles," *Journal of Advanced Transportation,* 2019, doi: 10.1155/2019/4109148.
[45] A. Bosu, J. C. Carver, C. Bird, J. D. Orbeck, and C. Chockley, "Process Aspects and Social Dynamics of Contemporary Code Review: Insights From Open Source Development and Industrial Practice at Microsoft," *Ieee Transactions on Software Engineering,* 2017, doi: 10.1109/tse.2016.2576451.
[46] S. National Institute of, Technology, and L. Information Technology. "Engineering Statistics Handbook - 2.3.1.3. Simple Linear Regression." https://www.itl.nist.gov/div898/handbook/prc/section2/prc213.htm (accessed.
[47] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika,* vol. 52, pp. 591-611, 1965, doi: 10.2307/2333709.
[48] *Scipy.stats.shapiro*. (2023). [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html
[49] R. B. D'Agostino, "An omnibus test of normality for moderate and large sample size," *Biometrika,* vol. 58, pp. 341-348, 1971 1971. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8546332/pdf/VJGB-25-21060.pdf.
[50] R. D'Agostino and E. S. Pearson, "Tests for departure from normality," *Biometrika,* vol. 60, pp. 613-622, 1973.
[51] *Scipy.stats.normaltest*. (2023). [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.normaltest.html

APPENDIX A: SUPPLEMENTAL INFORMATION

A.1 CTD Buckets

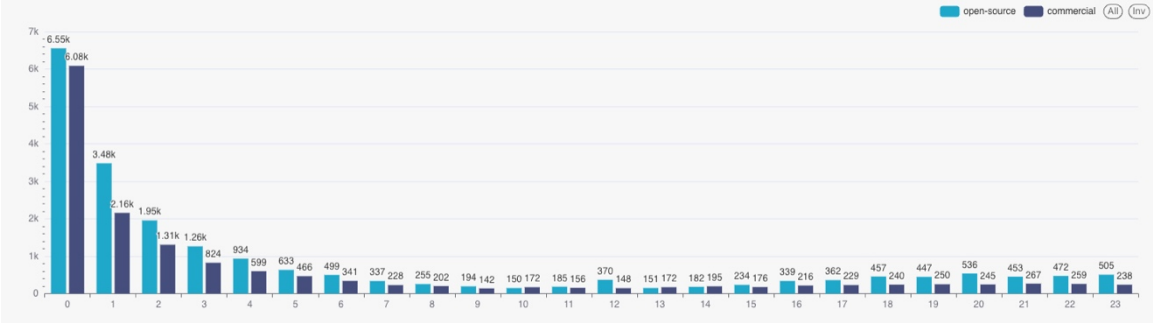

Figure 43: CTD < 24 Hours Bucket Distribution

A.2 Estimated Hours

Table 20: Outlier Details > 300 Estimated Hours

commit_sha	repo_code	complexity_delta	imputed_hours_mean	Description
a031f4ef83edc132d5f49382bfef491161de2476	open-source-repo-3	1002	1964.71	An abnormal addition of a 2000+ line case statement in a test. Each branch of the case statement added complexity at an abnormal rate, which translated to an inaccurate number of estimated hours.
4e831f4cee140b004fae44d528d47c284d8ac9eb	open-source-repo-2	338.88	826.54	This brings in an exact copy of the pg_bsd_indent repo. 61 changed files with 5,737 additions and 0 deletions. An unnatural wholesale import of code from another repo.
66b776b0250dd980d8f6aac264b5e3443ec465dc	open-source-repo-3	272.14	533.61	Likely due to a very small change that introduced a loop. This loop had a high Cyclomatic Complexity that caused the estimated hours to be reported unnaturally high.

8b23b7b04234424791e26b8d2d26f61ef1311a9f	open-source-repo-3	272.14	533.61	79 changed files with 84,042 additions and 1 deletion. Likely due to an unnatural wholesale import of logic.
0f7cdaaf57634138e058054d7bcd3beaae73d2db	open-source-repo-4	169.71	413.93	Merge pull request ESQL-655 from elastic/main. 96 changed files with 1,140 additions and 468 deletions. A large merge that is unlikely to have taken the number of estimated hours but had a complexity reading that caused the estimated hours to be calculated at this level.
d830543a49a315172e596c1971537028f43d809b	open-source-repo-4	145.17	354.07	Merge remote-tracking branch 'origin/main' into lucene_snapshot. 91 changed files with 4,407 additions and 3,668 deletions. A large merge that is unlikely to have taken the amount of estimated hours but had a complexity reading that caused the estimated hours to be calculated at this level.
b03a5a7406d137fa289c468afeb91179e408fd8d	open-source-repo-4	140.62	342.98	Merge pull request ESQL-1240 from elastic/main. 92 changed files with 1,005 additions and 542 deletions. A large merge that is unlikely to have taken the number of estimated hours but had a complexity reading that caused the estimated hours to be calculated at this level.
a7c186ff022d9efc30cf3976e2b1380712d1e2e9	open-source-repo-4	131.32	320.29	Merge pull request ESQL-637 from elastic/main. 98 changed files with 1,637 additions and 815 deletions. A large merge that is unlikely to have taken the number of estimated hours but had a complexity reading that caused the estimated hours to be calculated at this level.

69914bf61e40994d561f168b81bc2375208c138e	open-source-repo-4	131.32	320.29	[ML] Persist data counts and data feed timing stats asynchronously. 14 changed files with 275 additions and 91 deletions. A commit that does not appear to be unnaturally large. It's possible that this commit was actually contributed at a higher rate than the model contribution rate and was interpreted to have a high amount of estimated hours.
a38a018ef89c1cedcb1978f9f0e7a532f19ecc3d	open-source-repo-4	129.96	316.98	Merge pull request ESQL-1095 from elastic/main. 99 changed files with 1,760 additions and 1,626 deletions. A large merge that is unlikely to have taken the amount of estimated hours but had a complexity reading that caused the estimated hours to be calculated at this level.

A.3 Sanity Checking Imputed Hours

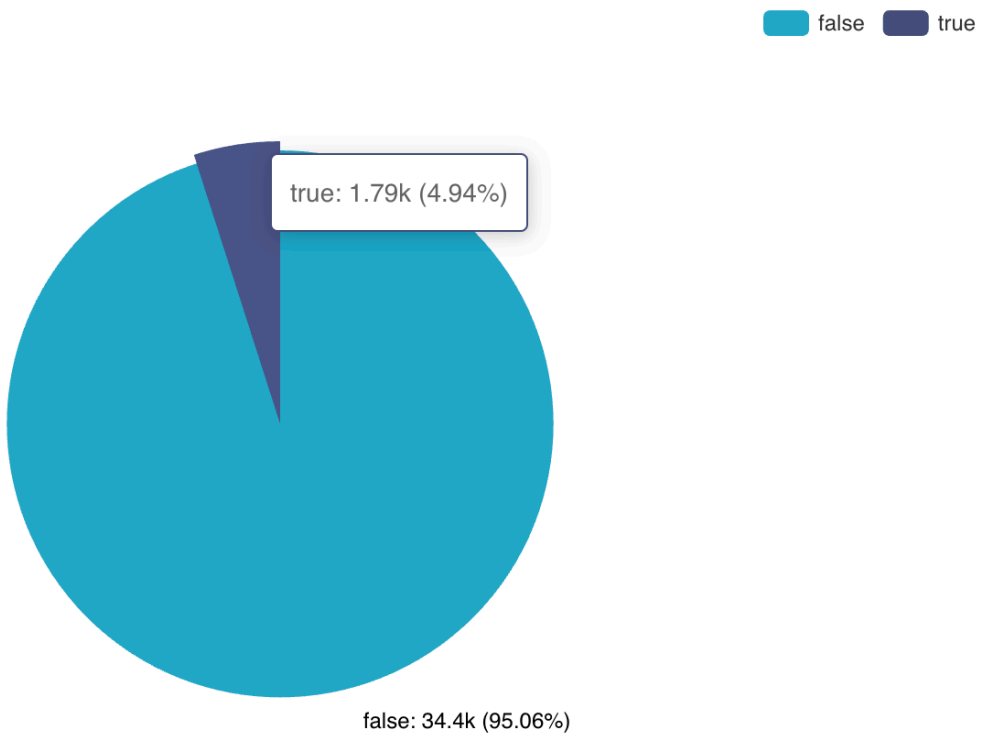

Figure 44: Experiment 3 - Unnatural vs. Natural Complexity Based Hour Commit Estimates