COEVOLUTION INDEX: A METRIC FOR TRACKING EVOLUTIONARY COUPLING

A Thesis

by

Hüseyin Yapıcı

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
May 2023

COEVOLUTION INDEX: A METRIC FOR TRACKING EVOLUTIONARY COUPLING

Approved by:

Prof. Dr. Hasan Sözer, Advisor
Department of Computer Science
Ozyegin University

Dr. Kübra Kalkan Çakmakçı
Department of Computer Science
Ozyegin University

Prof. Dr. Oya Kalıpsız
Department of Computer Engineering
Yıldız Technical University

Date Approved: 23 May 2023

For my mother Özlem and my father İlyas, who made this possible by introducing me to the computer at a young age, and have been always a source of constant support, encouragement and motivation.

ABSTRACT

This thesis proposes a new metric, namely the coevolution index (CEI), for measuring the relative evolutionary coupling of modules of a software system. CEI is inspired by the h-index, which is a popular metric used for measuring the productivity and citation impact of scholars and scientists. CEI of a module is equal to n, which is the number of times it is modified together with at least n other modules of the system. We develop a script that can calculate CEI for source files in a code repository. We analyze the repository of 7 software systems. Source files that are subject to a high number of changes to address issues tend to have high CEI scores. CEI also reflects a relative footprint in maintenance efforts by definition. Hence, it can help in tracking technical debt interest and focusing the refactoring efforts for improving maintainability and reusability.

ÖZETÇE

Bu tez, yazılım sisteminin modüllerinin göreceli evrimsel bağlılığını ölçmek için yeni bir metrik olarak birlikte evrim indeksi (CEI) metriğini önermektedir. CEI tanımlanırken, bilim adamlarının üretkenliğini ve atıflarının etkisini ölçmek için kullanılan popüler h-indeks metriğinden esinlenilmiştir. Bir modülün CEI değerinin n olması için sistemde en az n diğer modülle birlikte n kez değiştirilmiş olması gerekmektedir. Bu tez çalışması kapsamında, kod deposundaki kaynak dosyaları için otomatik CEI değeri hesaplayabilen bir araç geliştirilmiştir. Bu araç 7 farklı yazılım sisteminin kod depoları üzerinde uygulanmıştır. Sorunları çözmek için yüksek sayıda değişikliklere konu olan kaynak dosyalarının aynı zamanda yüksek CEI değerlerine sahip olma eğiliminde oldukları gözlemlenmiştir. CEI ayrıca tanım gereği yazılım bakım çalışmaları için harcanan eforun ayak izini yansıtmaktadır. Dolayısıyla CEI, teknik borç faizini takip etmek için kullanılabilmekle birlikte, yazılım bakım yapılabilirliğini ve yeniden kullanılabilirliğini iyileştirmek için uygulanan yeniden yapılandırma çalışmaları kapsamında, belirli modüllere odaklanma konusunda yardımcı olabilir.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor for his invaluable guidance, support, and mentorship. His insights have been instrumental in shaping the direction of my research.

I am indebted to my thesis jury members Dr. Kübra Kalkan Çakmakçı and Prof. Dr. Oya Kalıpsız for their valuable feedback and discussions. Their suggestions have provided significant contributions in writing my thesis.

I am grateful for the Apache Software Foundation's contributions to the open-source culture, its softwares, and the strong community that supports its development. Thanks to their codebase and JIRA records, we had the opportunity to expand our research and observe our results.

I would like to express my deepest acknowledgement to my precious parents, Özlem and İlyas, for their unwavering and infinite support and belief in me throughout my life. This journey has been made possible thanks to you.

Finally, I would like to thank to my unique and beautiful country, Türkiye, who raised and equipped me with valuable knowledge and skills.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acronyms

CEI Coevolution Index. i, iv, v, viii, ix, x, 1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 15, 17, 18, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43

EC Evolutionary Coupling. i, 1, 3, 4, 10, 11, 41, 42, 43

PR Pull Request. i, ix, x, 6, 13, 14, 15, 16, 17, 19, 22, 25, 28, 31, 32, 33, 34, 36, 37, 38, 39, 41

TD Technical Dept. i, 2, 3, 4, 5, 10, 42, 43

CHAPTER I

INTRODUCTION

Evolutionary Coupling (EC), which is also named as *change coupling* [1] and *logical coupling* [2], represents a relationship among software artefacts that are changed together [3]. This relationship directly impacts maintenance efforts although it is not always detectable via explicit dependencies in source code [2, 4, 5]. Therefore, the detection of EC can be used for discovering unseen, yet important dependencies [6].

EC has been mainly used for identifying architectural modularity problems [7], predicting defects [8], and recovering software architecture documentation [9, 10]. However, there has been no measurement approach proposed for EC other than counting the number of co-changes. A simple count of co-changes can be misleading [10]. Developers might commit a set of unrelated changes. So, we cannot assume the existence of EC between two modules if they are changed together only once. It is also hard to set a generic threshold for the number of co-changes. This thesis proposes a new metric, namely the CoEvolution Index (CEI), to provide a relative measure for tracking EC.

CEI is inspired by the *h-index* [11], which is a popular metric used for measuring the productivity and citation impact of scholars and scientists. The h-index is measured as the number of papers, h that have received at least h citations. It eliminates bias that can be imposed by a high number of publications with a lack of citations or a single publication with a high number of citations. It was shown that the h-index correlates with obvious success indicators such as prestigious awards, fellowships, and positions at top universities [12]. CEI aims at capturing the relative EC of a software module, m, by using a metric that is similar to the h-index as follows.

CEI(m) = n iff the module m is modified n times together with at least n other modules of the system.

CEI measures the cumulative footprint of a module in maintenance efforts. It eliminates bias that can be imposed by a high number of minor modifications unrelated with other modules or a single commit (pull/merge request) that involve the modification of many (possibly unrelated) modules at once. High CEI may be used as an indicator for deficiencies regarding modularity, maintainability and reusability. CEI can also provide valuable information for tracking Technical Debt (TD) [13, 14] and prioritizing refactoring efforts for managing TD.

We develop a script that can analyze commits in a repository and automatically calculate CEI for source files in that repository. We also conduct an empirical study with 7 software systems. We focus on source files that have high CEI scores. These files are subject to a relatively high number of changes applied to address issues. Hence, CEI can help in quantifying TD interest and obstacles regarding the maintainability and reusability of the pertinent files.

The remainder of this thesis is organized as follows. We summarize the related studies in Chapter 2. We explain the experimental setup in Chapter 3. Chapter 4 presents the results of the research and provides a detailed analysis and interpretation of the findings and a discussion of the implications of the results. Finally, in Chapter 5, we conclude the thesis.

CHAPTER II

RELATED WORK

There is no standard, commonly accepted metric for measuring EC [3]. To the best of our knowledge, there is no metric or measurement approach for it other than counting the number of co-changes. However, the relevance of the total count is unknown [10]. Should we assume the existence of EC between two modules if they are changed together once at any point in time? If not, how many times these modules should be changed together so that they are deemed to be coupled? CEI quantifies EC, which impedes the maintainability and reusability of modules of a system. It captures not only how frequently a module is changing, but also the difficulty of changes due to its coupling with other modules.

CEI can provide valuable information for tracking and managing TD [13, 14]. TD is a metaphor for describing design or implementation constructs that degrade the maintainability of a software system although they might be introduced intentionally as convenient and practical solutions in the short term [15]. TD can be accumulated incrementally and become unmanageable eventually [13], especially when it is left invisible and unresolved [16]. Therefore, TD should be managed to be kept under control [17]. Managing TD involves its tracking [17], which requires effective tools for its assessment [18]. An analysis of existing TD measurement tools [19] shows that the majority of these tools rely on static code analysis only. TD has been mainly evaluated with measures related to software code quality metrics such as modularity [20, 18]. However, empirical results show that TD is not always detectable by measuring code quality only [14]. Another observation regarding existing TD measurement tools [19] is that they mostly focus on the measurement of *principal* (i.e., cost of eliminating TD) rather than *interest*

(i.e., extra maintenance effort caused by TD). We propose CEI as a relative measure that can be used as a TD interest index. It can be used for assessing the consequences of TD.

The DV8 tool suite [21] and the Titan tool [22] analyzes the version history and issue reports in addition to source code. Although it does not provide a TD interest index explicitly [19], it facilitates the collection of data that can be used as a proxy measure [23]. Titan is implemented based on the *Design Rule Space (DRSpace)* concept [7]. A DRSpace is a set of architecturally connected modules that is related to a concern implemented by the analyzed system. A module can be architecturally connected with various other modules in different ways, each of which is captured by a DRSpace. Some of these connections can be subject to flaws and as such, they can lead to bug propagation. These flawed structures, namely *Architecture Roots (ArchRoots)*, are considered as a type of TD. Software modules that are involved in *ArchRoots* are associated with a high maintenance effort [24] as a penalty. This penalty can be measured [23] in terms of the extra effort spent on fixing defects, which is attributed to the existence of TD, i.e., *ArchRoots*.

Titan employs EC for identifying one of the DRSpaces, namely the *change space*, which is used for detecting architectural anomalies. EC, which is also named as *change coupling* [1] and *logical coupling* [2], represents a relationship (connection) among software artefacts that are frequently changed together. This relationship might be implicit in the source code [2, 4, 5]. Therefore, detection of EC can be used for discovering unseen, yet important dependencies [6]. However, it was not exploited by other TD measurement tools [19]. EC has been mainly used for identifying architectural anomalies and modularity issues [25, 2, 7] or pinpointing the root causes of defects [8]. Titan also employs EC for detecting architectural anomalies. However, EC is simply measured by counting the number of co-changes for each pair of modules. One has to define threshold levels above which this number would be deemed as an anomaly. We propose CEI a relative

measure and the rate of change in CEI can be used as a TD interest index, which is related to the increasing maintenance effort due to the existence of TD [14]. We present our empirical study in the following chapter to evaluate CEI.

CHAPTER III

EXPERIMENTAL SETUP

In this chapter, we describe our experimental setup, including the properties of our subject systems, dataset, employed platforms and tools. We aim at answering the following research question.

To what extent CEI of a module and/or the rate of its increase correlate with the issues reported for that module?

We would like to know *i)* if certain modules of a software system stand out among others due to significantly high CEI or an increasing trend in their CEI during software evolution, and *ii)* if such modules are also associated with a relatively high number of issues reported for the system. CEI can be analyzed at various levels of granularity. We treated every source file as a module and measured CEI for each file separately. Furthermore, to validate our findings and test the CEI, we conducted an additional study as well to get the count of changed files within the merged and closed pull requests (PRs). We compared whether there was any correlation between these values. Our subject systems are Apache Software Foundation's open-source software systems that implemented in Java, where each source file represents a class. We analyzed the repositories of 7 software systems. We explain these systems and the collected dataset in the following.

3.1 Subject Systems and the Dataset

Table 1 lists information regarding the subjects systems. All of these systems have public GitHub[1] repositories and they use JIRA[2] as the issue tracking system. However, it is important to note that *Iceberg* and *Drill* projects do not have JIRA

[1] `https://github.com/`
[2] `https://www.atlassian.com/software/jira`

issues available. All the subject systems listed in Table 1 mostly employ Java as the programming language. Some of them have been previously used in other empirical studies [7] as well.

Table 1: An overview of the subject systems.

System	1st Year of Release	# of Lines of Code	# of Source Files	# of Contributors
Hadoop	2006	3.9M	11.6K	491
HBase	2008	998K	4.8K	402
TinkerPop	2010	681K	3K	180
CXF	2008	931K	7.3K	193
Tika	2007	217K	1.7K	119
Iceberg	2017	421K	2.6K	355
Drill	2015	886K	4.6K	183

The first column of Table 1 lists the names of the systems, and the second column lists the initial year of release for each. The total lines of code, and the number of source files for each project are provided in the third and fourth columns, respectively. These values are obtained by using the *CLOC* [3] tool at the time of writing this thesis. The last column lists the number of developers who contribute to each of these projects. This information is obtained from the GitHub projects' info as contributors.

Hadoop is a data processing framework for distributed storage and processing of big data offering reliability and scalability. The *Hadoop* software library utilizes simple programming models to process large datasets across computer clusters as a framework. *HBase* is a non-relational distributed database designed to handle large-scale data storage and processing. Due to its nature as a non-relational database, It is capable of storing unstructured and semi-structured data as well. *TinkerPop* is a framework for graph computing that is open-source and provides developers wide range of capabilities and technologies, along with an extensive ecosystem of third-party graph libraries and systems. *CXF* is a Java-based web service framework built on two projects, namely *Celtix* and *XFire*. It facilitates

[3] `https://cloc.sourceforge.net/`

the development of services that can support a variety of protocols such as SOAP, XML/HTTP, and RESTful HTTP using frontend programming APIs like JAX-WS and JAX-RS. Thanks to it, developers can build services that are capable of seamless integration across different platforms and technologies. *Tika* is a content analysis toolkit that can detect and extract text and metadata from thousand of various file types including popular ones like PPT, XLS, and PDF. *Iceberg* is an open table format that provides high-performance capabilities for large-scale analytic tables. It enables a system to concurrently access and process the same tables securely with the ability for using multiple engines like Spark, Trino, Flink, Presto, Hive, and Impala. Thanks to *Iceberg*, it is possible to use SQL tables for big data with high reliability and simplicity. *Drill* is a schema-free SQL query engine that allows to use of a diverse range of NoSQL databases and file systems including MongoDB, Google Cloud Storage, Amazon S3, Azure Blob Storage, *HBase* and more.

Table 2 lists the number of resolved tasks for these systems as collected from JIRA. In the case of *Iceberg* and *Drill* projects, no JIRA projects have been created as an issue-tracking system. There are no available and trackable tasks for these projects in JIRA. Consequently, this information could not be included for these projects in Table 2.

Table 2: The number of completed tasks of various types.

System	**Total**	**Bug**	**Improvement**	**Feature**	**Other**
Hadoop	9.9K	5K	2.5K	476	1.8K
HBase	18.5K	8.5K	3.8K	465	5.5K
TinkerPop	866	472	393	0	1
CXF	6.6K	4K	1.4K	252	1K
Tika	2.4K	992	845	148	439
Iceberg	N/A	N/A	N/A	N/A	N/A
Drill	N/A	N/A	N/A	N/A	N/A

To obtain the data in Table 2, the analysis was performed based on task types that were resolved as "Fixed" and moved to the "Done" status before the year 2023 for all the projects. We also queried tasks based on the issue types they are

associated with. These queries are listed in Listing 3.1 and 3.2.

Listing 3.1: The JIRA query used for searching all tasks, where X refers to the project name.

```
statusCategory = Done and project = X and
resolution = fixed and resolutiondate < ‘‘2023−01−01”
```

Listing 3.2: The JIRA query used for searching tasks with issue type, where X and Y respectively refer to the project name and the issue type like *Bug*, *New Feature*, and *Improvement*.

```
statusCategory = Done and project = X and issuetype = Y
resolution = fixed and resolutiondate < ‘‘2023−01−01”
```

It was not always possible to access this information since JIRA records were not kept for *Iceberg* and *Drill* projects; instead, the “Issues” feature on GitHub is used. When the GitHub issues are examined, we observed that 39 different labels were used for *Iceberg* and 36 different labels were used for *Drill*. However, we noticed that labels were not assigned to issues properly. For instance, only a few issues among more than 10,000 issues were labelled as “Bug”. Due to incomplete and incorrect labeling, using this information would be misleading. For this reason, it was not used and ignored in our empirical study.

The second column of Table 2 lists the total number of resolved tasks for each system over its lifetime. The third, fourth and fifth columns provide the number of tasks for various categories, namely *bugs*, *improvements* and *new features*, respectively. The last column lists the number of tasks for other categories such as *subtask* etc. We observe that the majority of tasks are categorized as bugs and improvements. We hypothesize that a module with a high CEI or quickly increasing CEI would be associated with these tasks, i.e., they will be subject to more improvements and they will lead to bug propagation. Such modules are expected to have a higher level of technical complexity and undergo more frequent changes and improvements. These modules, which are constantly modified, may be more

prone to errors and problems. Furthermore, these constant changes increase TD.

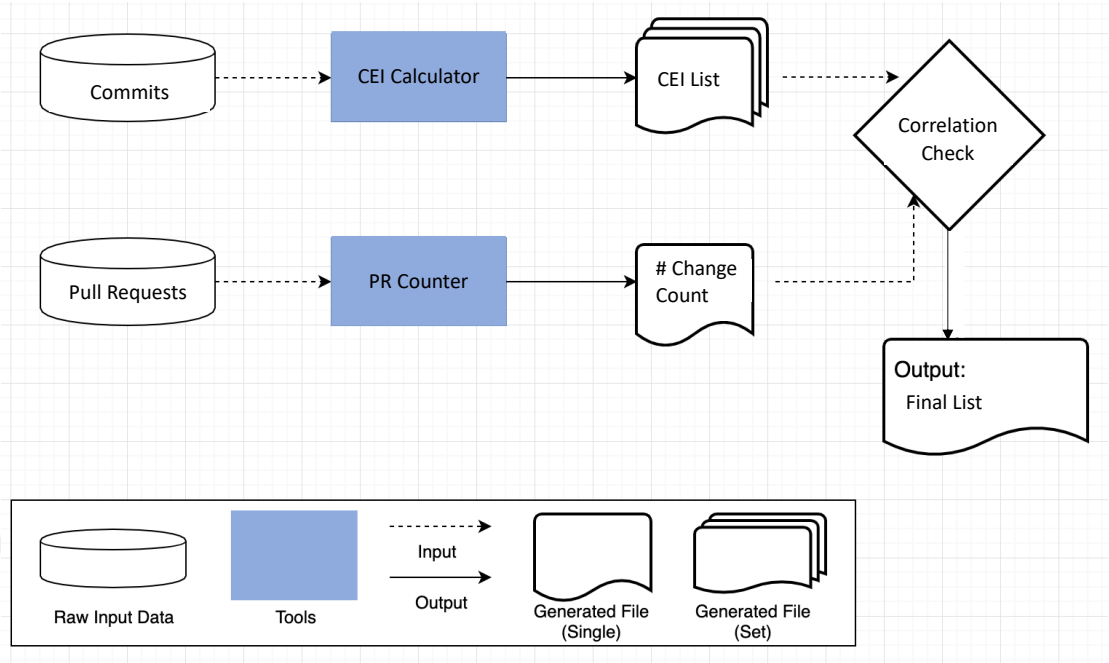

Figure 1: The overall process followed in our empirical study.

Leveraging the data obtained from GitHub, we utilized two separate datasets in our research as seen in Figure 1. One of these datasets is specific to each project and consisted of CEI values calculated for each source file modified in the commits. We developed a Python script to obtain and analyze the commit history. Also this script automatically calculates CEI for each file between the given start and end times. It uses the GitHub API to retrieve commits in JSON format. It employs multiple tokens and pagination to overcome the request limits imposed by the API. Then, the script extracts the list of changed files in each commit to identify the level of EC among them. The type of file to be listed is filtered by giving the extensions to the script as input. Thanks to this filtering, CEI calculations are made only for the given types of files. Then, it starts processing the files that are changed in each commit. It keeps track of file pairs that change together. For file pairs that change together, it increases the co-change count by 1 and writes it back to the memory. It stores also the filenames (with path) with the number of co-changes of these file pairs. A two-dimensional matrix is used for this purpose. After processing the file pairs within a commit, it repeats the same process for

a new commit in a new iteration. The script stops processing commits when it reaches the specified end date. This way, the information about how many times files have changed together within the given date range is determined. Then, the rows of the matrix are processed one by one. For each file being considered in a row, the CEI value is calculated. The script extracts the list of changed files in each commit to identify the level of EC among them. It calculates CEI for each module m as follows. If the corresponding file has been modified n times together with at least n other modules in the system, then the CEI value for module m will be n. Finally, the calculated CEI value is added to the last column of the matrix. After the CEI calculation is completed for all files, the matrix that is kept in the memory is saved in .csv format with the pre-determined file name and path. After this process, the script terminates its execution.

The script is parameterized to take the Github repository path, API key and the list of relevant file extensions as parameters as shown in Listing 3.3.

Listing 3.3: A snippet from the script, where the parameters are set for the analysis of commits and calculation of the number of source file co-changes.

```
repo = 'githubusername/githubprojectname'
api_key = 'your_api_key_here'
extension = ['''file_extension1''', ''.file_extension2''']
```

Note that multiple file extensions can be specified in an array at same time for other uses. The following terminal command is used for running the script:

```
python3 main.py --start 2020-01-01 --end 2022-12-31
```

When executing this command, one needs to provide the start date after "–start" and the end date after "–end". Both date types support the format "yy-mm-dd" that corresponds to numeric values for year-month-day. In our empirical study, we divided the time into quarterly periods and ran the script based on the last 3 years cumulatively. We first ran the script for the first 3 months, then for the first 6 months, and so on. In this way, we increased the scope of the analysis

period incrementally. Finally, we ran it for the whole 36 months period. As a result, we obtained 12 different .csv files. We obtained CEI values for all files that were committed in the last 3 years across the 7 software systems we worked on. This dataset allowed us to analyze the cumulative changes within the 3 years period.

A snippet from one of the generated .csv files is shown in Figure 2. The contents of the file are not easily readable. Moreover, in cases where the file contains hundreds or even thousands of entries (because of the number of changed file names), it becomes even more complex and challenging to comprehend the content.

	A
1	,hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/conf/Configuration.java,hadoop-common-project/hadoop-commo
2	PmemMappableBlockLoader.java,hadoop-hdfs-project/hadoop-hdfs/src/main/java/org/apache/hadoop/hdfs/server/datanode/fsdataset/impl/Pm
3	mmand/TensorFlowLaunchCommand.java,hadoop-submarine/hadoop-submarine-yarnservice-runtime/src/main/java/org/apache/hadoop/yarn/su
4	che/hadoop/yarn/api/protocolrecords/impl/pb/GetContainersResponsePBImpl.java,hadoop-yarn-project/hadoop-yarn/hadoop-yarn-common/src/
5	manager/webapp/RMAppsBlock.java,hadoop-yarn-project/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-resourcemanager/src/main/java,
6	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/conf/Configuration.java,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
7	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/AbstractFileSystem.java,0,0,0,2,0,0,0,2,2,2,0,2,2,2,0,0,2,2,0,0,0,
8	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/BatchListingOperations.java,0,0,0,0,0,2,0,0,0,2,0,0,0,0,0,2,0,0,0,
9	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/ChecksumFileSystem.java,0,2,0,0,0,0,0,2,2,2,0,2,2,2,0,0,2,2,0,0,
10	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/CommonConfigurationKeys.java,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

Figure 2: A part of a screenshot from the generated raw .csv file with calculated file change and CEI values.

The automatically generated .csv file is post-processed by another Python script that we developed. This script converts the file to .xlsx format. A snippet of the resulting file is shown in Figure 3. The file contains the same information in the form of a matrix, where rows and columns correspond to source files in the system.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Unnamed: 0	c/main/jav	/main/java	ain/java/o	main/java/	n/java/org	in/java/org	/src/main/	ain/java/o	src/main/j	/src/main/	n/src/main	c/main/java
2	hadoop-co	0	0	0	0	0	0	0	0	0	0	0	0
3	hadoop-co	0	0	0	2	0	0	0	2	2	2	0	2
4	hadoop-co	0	0	0	0	0	2	0	0	0	2	0	0
5	hadoop-co	0	2	0	0	0	0	0	2	2	2	0	2
6	hadoop-co	0	0	0	0	0	0	0	0	0	0	0	0
7	hadoop-co	0	0	2	0	0	0	0	0	0	2	0	0
8	hadoop-co	0	0	0	0	0	0	0	0	0	0	0	0
9	hadoop-co	0	2	0	2	0	0	0	0	2	2	0	2
10	hadoop-co	0	2	0	2	0	0	0	2	0	2	0	2

Figure 3: A part of a screenshot from the generated raw .xlsx file with calculated file change and CEI values.

Each cell of the matrix partially depicted in Figure 3 shows the number of

times the corresponding pair of files change together in a commit. The matrix is not a square matrix because it includes one additional column at the end. The last column lists the calculated CEI values for each source file. Figure 4 shows a snippet from an edited version of the file contents depicted in Figure 3, where only the first and the last columns are kept and the remaining columns are removed.

	A	B
1	FileName	CEI
2	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/conf/Configuration.java	1
3	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/AbstractFileSystem.java	2
4	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/BatchListingOperations.java	2
5	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/ChecksumFileSystem.java	2
6	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/CommonConfigurationKeys.java	2
7	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/CommonPathCapabilities.java	2
8	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/CreateFlag.java	2
9	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/DelegateToFileSystem.java	2
10	hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/fs/FileContext.java	2

Figure 4: A part of a screenshot from the edited .xlsx file with calculated file change and CEI values.

The second dataset is a single file where we tracked the number of changes for files mentioned in PRs that are merged and closed. We call these changes issue-related changes. This file is obtained by following the activities shown in Figure 1. We performed another ranking based on the number of issue-related changes. This number is also calculated for each file with a Python script that we developed. This script, similar to the previous one, utilizes the GitHub API. It processes the merge and closed PRs within the given start and end dates specified as parameters. In our empirical study, we set the start and end dates to cover the last three years, starting from the beginning of 2020 until the end of 2022, just like we did for running the previous script to be consistent. The script also facilitates data filtering based on file extensions provided as parameters. As a result of the script's execution, for each PR, it keeps the file names that changed in the memory and incrementally updates their change counts. It continues this process iteratively. After processing all the PRs within the specified date range, it saves the dataset in .csv format to the pre-defined file name and path, and then it terminates. The script we developed uses the following parameters as inputs: the GitHub repository path, GitHub API key. Additionally, the start and end

dates that are formatted as "yy-mm-dd" and the file extensions, can be provided as parameters in the script as listed in the code snippet in Listing 3.4.

Listing 3.4: A snippet from the script, where the parameters are set for the analysis of PRs and calculation of the number of times each source file is involved in a PR.

```
repo = 'githubusername/githubprojectname'
api_key = 'your_api_key_here'
getCommits(repo, api_key,'2020-01-01','2022-12-31',
['''.file_extension1''', '''.file_extension2''''])
```

The following terminal command is used for running the script:

```
python3 PRReaderMergeCount.py
```

After running this script, we converted the resulting .csv file to the more readable .xlsx format, similar to what we did with the previous .csv files. This conversion made the dataset ready for use as input. Note that the set of files that take place in the two rankings are not necessarily the same.

We used a MacBook Pro 2019 computer with a 2,6 GHz 6-Core Intel Core i7 Processor and 16 GB 2667 MHz DDR4 RAM memory to run the scripts for each subject system. Our scripts and the collected dataset are available online[4].

Table 3: An overview of the subject systems.

System	**Total # of Commits**	**Total # of Merged PRs**	**# of Merged PRs in the Last 3 Years**
Hadoop	26.3K	2.7K	2.1K
HBase	19.5K	3.7K	3K
TinkerPop	19.5K	2K	443
CXF	17K	659	366
Tika	6.7K	615	477
Iceberg	4K	4.5K	3K
Drill	4.4K	2.6K	584

We obtained the total number of commits, merged and closed PRs for the subject systems we analyzed from GitHub. In Table 3, the first column displays

[4]https://figshare.com/articles/dataset/CoevolutionIndex/23501787

the subject systems' names. The second column represents the total commit count, while the third column shows the count of merged and closed PRs. Since our empirical study focuses on the last 3 years, we also include the count of merged and closed PRs within that time frame in the fourth column. We observe significant differences between the number of commits and the number of PRs.

Table 4: An overview of the subject systems.

System	**# of Files in Commits**	**# of Files in PRs**
Hadoop	5,540	3,313
HBase	3,765	3,308
TinkerPop	850	535
CXF	2,504	841
Tika	2,806	728
Iceberg	4,936	3,435
Drill	2,736	1,648

As a result of the study carried out by following the process depicted in Figure 1, we obtained the source files modified in commits and PRs. We observed that the files in the merged and closed PRs belonging to the last 3 years are significantly different from the files changed in the commits for each subject system. This important observation is better visible in Table 4. We can see significant differences even between the number of files for most of the systems. In our empirical study, we aim at evaluating the correlation between the CEI values of source files and the number of times these files are involved in a merged and closed PR. In the final part of Figure 1, we use the previously obtained two different datasets to perform a correlation check. For this purpose, we employ an evaluation metric that allows us to assess the correlation between the datasets. In the following, we explain the metric we used for this evaluation.

3.2 Evaluation Metric

Correlation measures the strength of association between two variables [26]. We evaluated the correlation between the two different rankings of source files: *i)* the ranking based on CEI, and *ii)* the ranking based on the number of times each file is

changed in association with a merged and closed PR. We need a correlation metric to quantify this evaluation. The metric has to be applicable when the evaluated data set does not follow a normal distribution. In particular, a non-parametric test must be used if the population underlying the sample is not normally distributed, which is the case with our data set. We used Spearman's rank correlation coefficient (Spearman's ρ) [26], which is one of the commonly applied non-parametric statistical tests.

Spearman's ρ measures the relationship between two variables based on their ranking. In order to determine the rank correlation, the first step involves ranking all the observations in variable X and separately ranking all the observations in variable Y. In cases where tied values occur, they are assigned the average of the ranks they would have received if they were not tied. The rank correlation can be obtained by applying the formula listed in Equation 1.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{1}$$

Hereby, d_i represents the pairwise distances between the ranks of the variables x_i and y_i. n refers to the total number of samples in the dataset. Spearman's ρ value can vary between $+1$ and -1, indicating the range of possible values. These values correspond to various types of relations as listed in the following.

- A value of $+1$ indicates a perfect positive rank correlation with statistically high significance as the ranks of one variable increase, the ranks of the other variable also increase consistently.

- A value of -1 represents a perfect negative association between the ranks, implying that as the ranks of one variable increase, the ranks of the other variable decrease in a consistent manner.

- A value of 0 indicates no rank correlation, suggesting that there is no consistent relationship between the ranks of the two variables. The more the value is closer to 0, the less is the statistical significance.

We worked on a project-by-project basis for the top 10 files with the highest final CEI values among the files in our projects. For each variable x_i, we used the last CEI value specific to that file. Additionally, we obtain the count of changes for each file through merged and closed PRs at GitHub in each project and use these values as y_i. We calculated the Spearman coefficient by the Spearman's rho formula using an online website [5]. We discuss the obtained results in the following chapter.

[5] `https://www.statskingdom.com/correlation-calculator.html`

CHAPTER IV

RESULTS AND DISCUSSION

In this chapter, we first provide information regarding the overall distribution of the number of issue-related changes for files. Then, we focus on the top 10 files of each subject system, when they are ranked based on their CEI at the end of the 3-years period. We analyze the trend of increase in CEI for these files based on calculations performed at the end of every 3-months period. Hence, CEI is measured 4 times within a year and 12 times in total within the 3-years period. We also calculate the ranking of the top 10 files, when they are ranked based on the number of issue-related changes. We discuss the correlation between the two rankings. Finally, we provide an overall evaluation and conclude the chapter with a discussion of threats to validity.

4.1 Distribution of Issue-related Changes

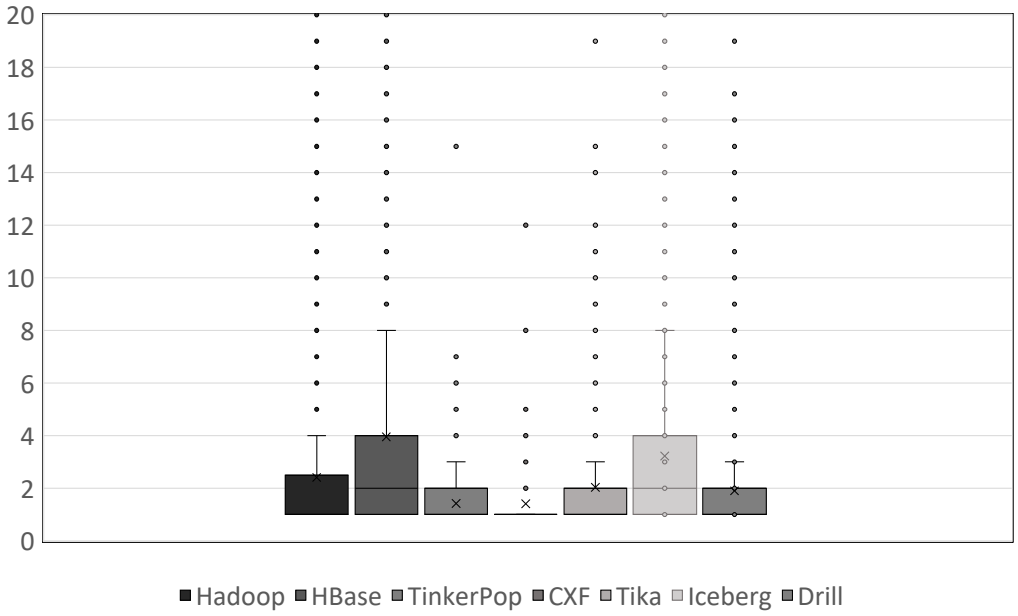

Figure 5: The distribution of the number of issue-related changes applied on files for all the subject systems.

We draw box plots for each project based on the second dataset, which includes the count of issue-related changes from merged/closed pull requests. Figure 5 depicts a box-plot regarding the number of issue-related changes for the files of 7 systems. The y-axis represents the number of such changes per file, and it is bounded at the value of 20 to make the chart more readable since the maximum values significantly vary among the systems. In fact, the maximum numbers of changes for a file are calculated as 72, 201, 19, 15, 12, 60 and 29 for *Hadoop*, *HBase*, *TinkerPop*, *CXF*, *Tika*, *Iceberg* and *Drill* respectively.

4.2 Hadoop Project

We can observe a high variance in the number of changes and the existence of a high number of outliers. There exists a high variance among the outlier files as well. For instance, Figure 6 depicts a Pareto chart for those files from the *Hadoop* project that are modified 5 or more times in association with a merged and closed PR.

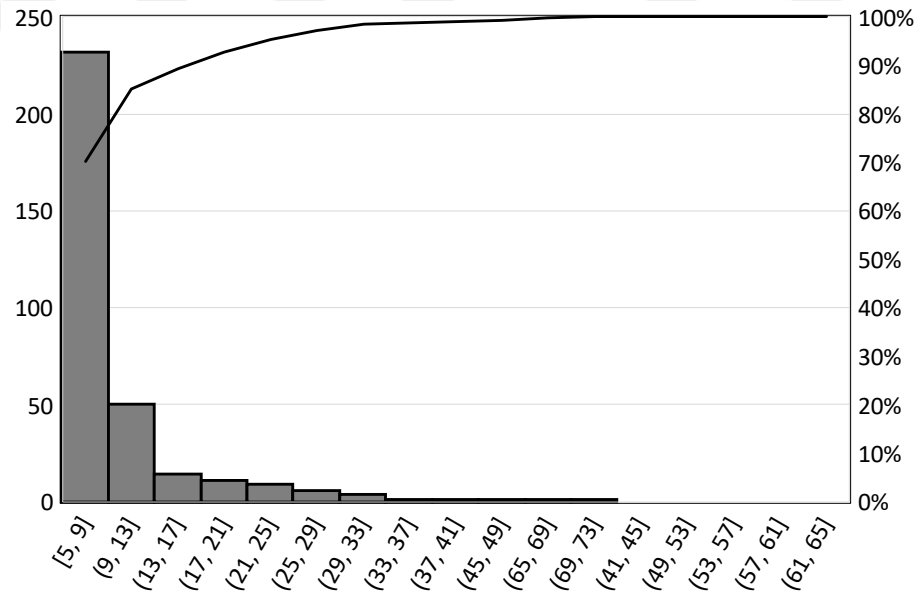

Figure 6: The distribution of the number of issue-related changes for the files of the *Hadoop* project that changed 5 or more times to address an issue.

More than 200 files were changed between 5 and 9 times to address an issue. On the other hand, the number of files that were changed between 9 and 13 times is 50. Note that these are only the outlier files among 3313 of them that are partially depicted in Figure 5.

We calculated CEI cumulatively for all the files of all the subject systems at the end of every 3 months starting from the beginning of 2020. Then, we ranked these files based on their final CEI at the end of 36 months. We exclusively analyzed the top 10 files from each project based on this ranking.

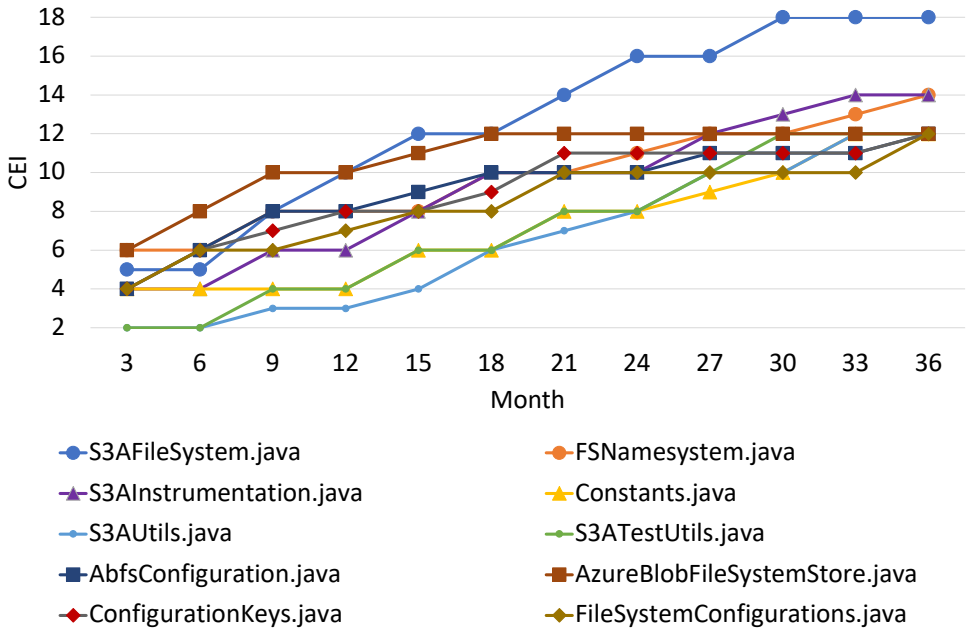

Figure 7: The increasing CEI trend for 10 files with the highest CEI from the *Hadoop* project throughout the 36 months period.

Figure 7 depicts CEI of top 10 files of *Hadoop* over 36 months. We observe a linearly increasing CEI trend in general. This trend holds for all the 10 files until the 18^{th} month, after which CEI remains constant for a few files. The maximum CEI is calculated as 18 for one of the files (*S3AFileSystem.java*). Two files reached 14 and the remaining 7 files reached 12 as the final CEI at the end of 3 years. These files are listed in Table 5.

Table 5: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *Hadoop* project.

File Name	**Last CEI**	**Change Count**	**Change Rank**
S3AFileSystem.java	18	72	1
FSNamesystem.java	14	66	2
S3AInstrumentation.java	14	34	18
AzureBlobFileSystemStore.java	12	38	4
FileSystemConfigurations.java	12	32	7
Constants.java	12	31	9
ConfigurationKeys.java	12	27	10
AbfsConfiguration.java	12	25	16
S3AUtils.java	12	25	18
S3ATestUtils.java	12	18	31

The second column of Table 5 lists the final CEI. The third column lists the number of issue-related changes. The last column (*change rank*) shows the ranking of these files based on the number of these changes.

We see that the files that are ranked the first (*S3AFileSystem.java*) and the second (*FSNamesystem.java*) are also the top two files when they are ranked according to CEI. The ranking of other files is not aligned directly with their CEI; however, these are the files that stand out even among the outliers, which are changed at least 18 times to close an issue (See Figure 6). The lowest ranking is 31 among 3313 files. That corresponds to less than 1% of the overall files. Hence, high CEI points at files that are associated with a high number of issue-related changes. We also calculated Spearman's rank correlation coefficient to assess the relationship between these two values, and test our CEI performance. Spearman's ρ for *Hadoop* project is equal 0.74388. This value also indicate statistically significant.

4.3 HBase Project

We observe a similar trend in Figure 8 for the *HBase* project. The chart depicted in this figure also takes outlier files from those spotted in Figure 5. It shows the number of files that are changed 9 or more times in association with a merged and closed PR. We get the value of 9 mentioned here from the box plot that was generated as a result of our calculations.

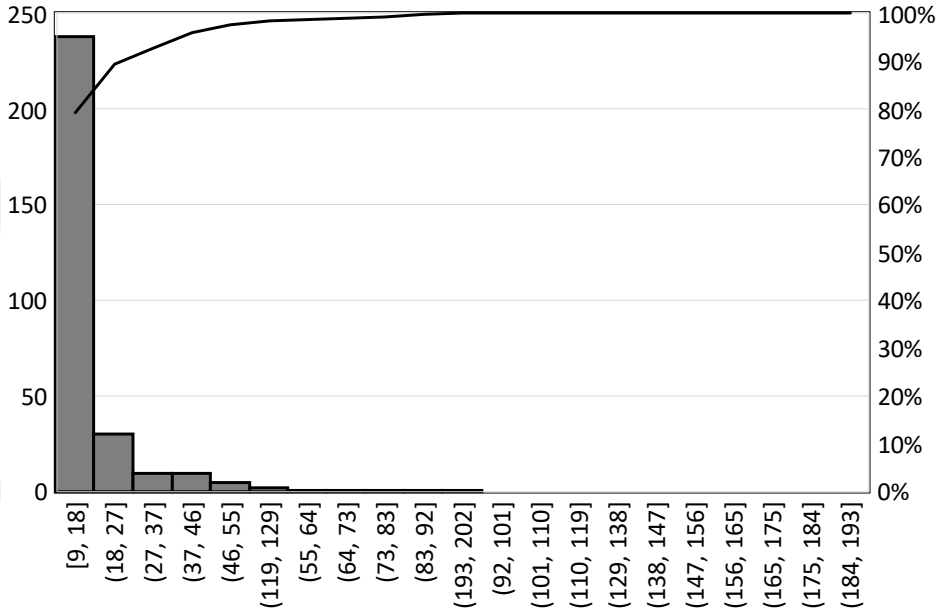

Figure 8: The distribution of the number of issue-related changes for the files of the *HBase* project that changed 9 or more times in association with a merged and closed PR.

We can again observe a significant imbalance among the number of files in terms of the number of their issue-related changes. More than 200 files were changed between 9 and 18 times, whereas the number of those files that are changed 18 to 27 times is less than 50.

Figure 9 shows the CEI values of the top 10 files in *HBase* over a span of 36 months. 7 of the 10 files analyzed for the *HBase* project have exactly the same sequence of CEI values (depicted with triange-shaped tick marks), starting at 0, quickly increasing up to 14 on the 21^{st} month, and remaining the same for the last 15 months.

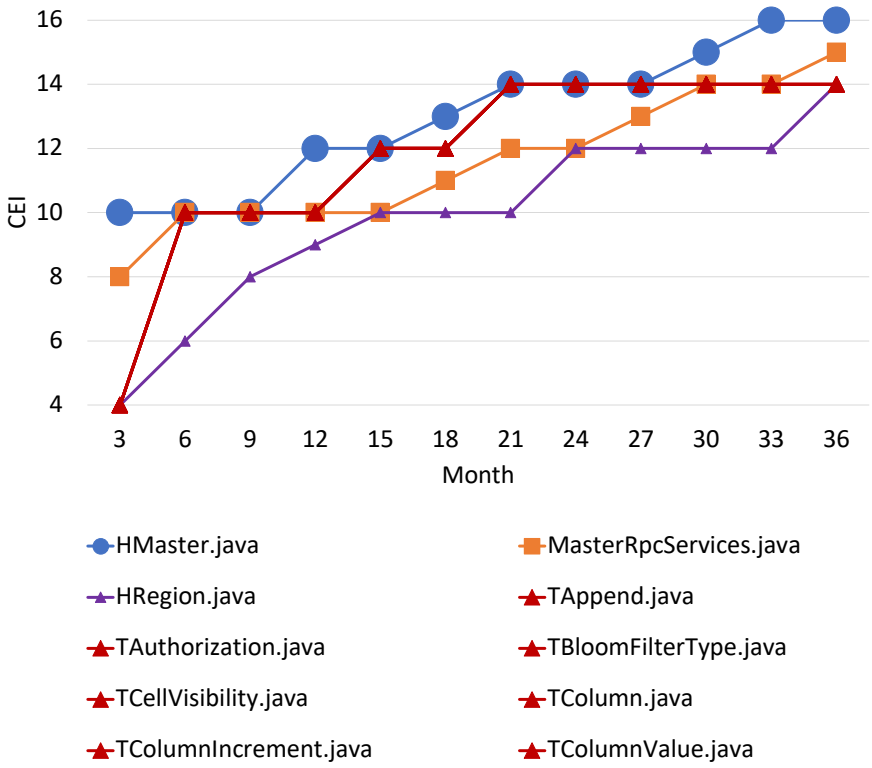

Figure 9: The increasing CEI trend for 10 files with the highest CEI from the *HBase* project throughout the 36 months period.

The file with the highest CEI (*HMaster.java*) has a regularly increasing trend of CEI starting at 10, and increasing up to 16 at the end of 3 years. The second highest CEI is calculated as 15 for a file (*MasterRpcServices.java*) with a similar trend regarding the increase of CEI. The third highest CEI is calculated as 14 for a file (*HRegion.java*). As a difference from the other two, CEI of this file increases more rapidly, starting from 0 at the end of 3 months.

However, we do not observe any impact of this variance (i.e., in the trend of increase of CEI) on the ranking of files based on the number of issue-related changes. Table 6 lists number of these changes and the corresponding rankings of the 10 files.

Table 6: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *HBase* project.

File Name	**Last CEI**	**Change Count**	**Change Rank**
HMaster.java	16	201	1
MasterRpcServices.java	15	76	5
HRegion.java	14	128	2
TAppend.java	14	14	117
TAuthorization.java	14	14	118
TCellVisibility.java	14	13	132
TBloomFilterType.java	14	12	175
TColumn.java	14	11	204
TColumnIncrement.java	14	11	205
TColumnValue.java	14	11	206

We see that the file that is ranked first (*HMaster.java*) is also the top file based on its CEI ranking. The next two files (*MasterRpcServices.java* and *HRegion.java*) are also among the top in both rankings. The ranking of the other files are in the order of hundreds; however, these files are all among the outliers, which are changed at least 11 times to close an issue (See Figure 8). The lowest ranking is 206 (approximately top 6%) among 3308 files.

We also calculated Spearman's ρ, which is 0.63246. This value indicates a moderate positive relationship between the two data set being compared in the case of *HBase*.

4.4 TinkerPop Project

The distribution of the number of issue-related changes for the files in the *TinkerPop* project is presented in Figure 10. The highest number of changes for a file in this project is 19, which is observed in only one file. Conversely, 47 files were modified 4-7 times.

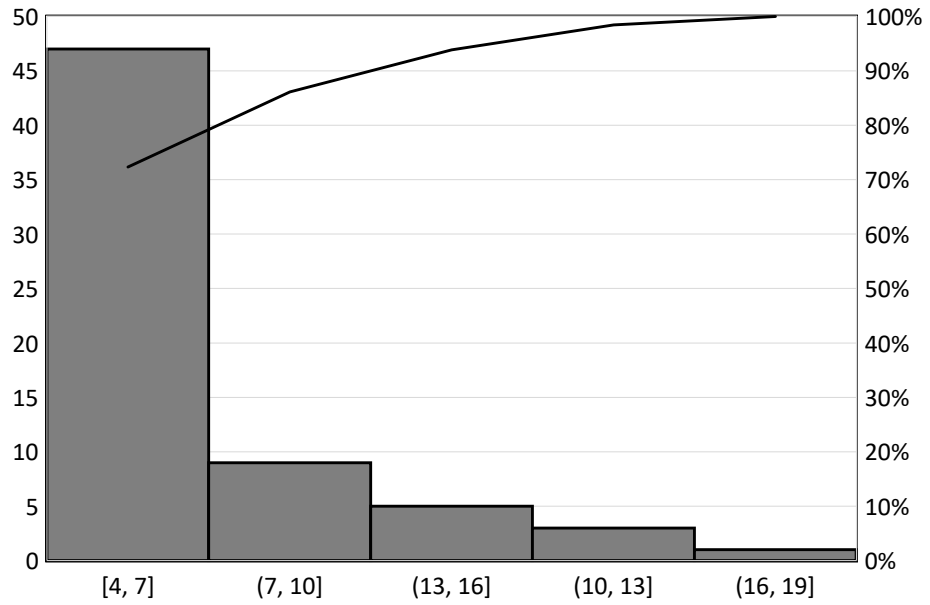

Figure 10: The distribution of the number of issue-related changes for the files of the *TinkerPop* project that changed 4 or more times in association with a merged and closed PR.

Files with high CEI values in this project are those that have undergone a significant number of changes related to issues.

Figure 11 below depicts CEI of top 10 files of *TinkerPop* over the 36 months period. Out of the files that we have reviewed, all except for one had an initial CEI value of 0. After 36 months, 3 of these files had reached a CEI value of 16.

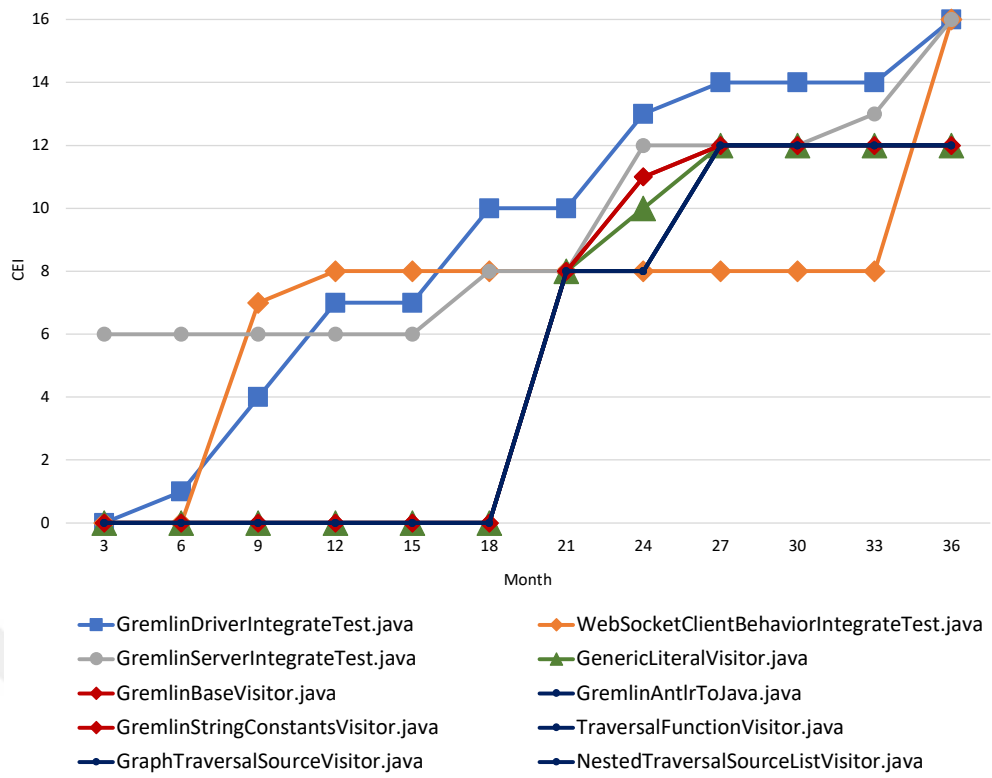

Figure 11: The increasing CEI trend for 10 files with the highest CEI from the *TinkerPop* project throughout the 36 months period.

When examining the file *GremlinDriverIntegrateTest.java*, it is observed that the CEI value reaches 16 like the other two files in the 36th month. However, unlike the other files, CEI value of this file has shown a steady increase from the beginning (with CEI = 0) instead of starting at a similar level. *WebSocketClient-BehaviorIntegrateTest.java* remains the same from months 12 to 33 with a CEI value of 8. However, by the end of the 36th month, the CEI value has dramatically doubled and reached 16. It is evident that significant development has been made in these 3 months that significantly affected this file. The other 7 files, on the other hand, have gone through different processes and converged at CEI is 12. Due to changes in the code base, the corresponding test codes have also been updated.

It is worth mentioning that a significant number of files labeled with the naming convention "visitor" exhibit a high CEI value, suggesting strong interdependencies among these files for the specific feature under consideration.

Table 7: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *TinkerPop* project.

File Name	**Last CEI**	**Change Count**	**Change Rank**
GremlinDriverIntegrateTest.java	16	19	1
WebSocketClientBehaviorIntegrateTest.java	16	15	2
GremlinServerIntegrateTest.java	16	10	10
GenericLiteralVisitor.java	12	10	12
GremlinBaseVisitor.java	12	8	17
GremlinAntlrToJava.java	12	2	179
GremlinStringConstantsVisitor.java	12	2	180
TraversalFunctionVisitor.java	12	2	182
GraphTraversalSourceVisitor.java	12	1	350
NestedTraversalSourceListVisitor.java	12	1	354

The file names and the number of changes made in the corresponding pull requests have been clearly documented in the accompanying Table 7. So, high CEI values designate files that are associated with a high number of issue-related changes for this project as well. Based on its CEI ranking, we observe that the file ranked first (*GremlinDriverIntegrateTest.java*) is also the top file. The file ranked second in the change rank also corresponds to the second highest CEI value, which is *WebSocketClientBehaviorIntegrateTest.java*. Upon analyzing the top 10 files, the lowest ranking is 354 out of 535 files, approximately in the top 66

Spearman's rank correlation coefficient is calculated for this project, and Spearman's ρ is 0.7739. This indicates that there is a strong positive relationship between the variables being compared. Therefore, this value indicates a stronger relationship compared to the values observed for *Hadoop* and *HBase*.

4.5 CXF Project

Figure 12 shows the distribution of the number of issue-related changes for the files of the *CXF* project. The maximum number of changes for any file is 15 for this project. In fact, there is only a single file, which was changed 15 times. On the other hand, there are 26 files that were changed 4, 5 or 6 times.

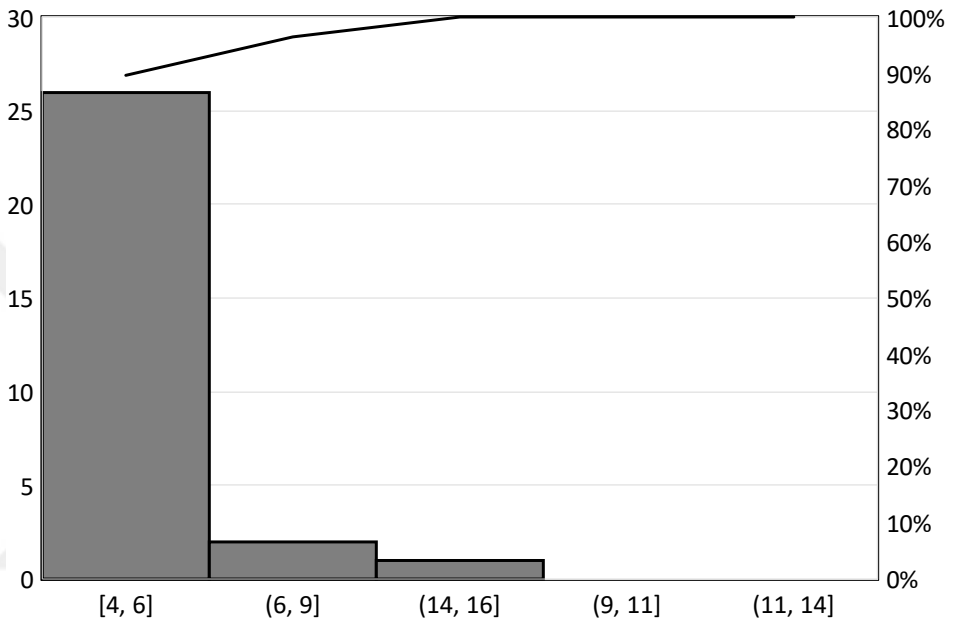

Figure 12: The distribution of the number of issue-related changes for the files of the *CXF* project that changed 4 or more times in association with a merged and closed PR.

Figure 13 below depicts CEI of top 10 files of *CXF* over the 36 months period. 6 of the 10 files analyzed for the *CXF* project has exactly the same sequence of CEI values (depicted with small, circular tick marks), starting at 0, suddenly increasing up to 6 between the 9^{th} and the 12^{th} months, and remaining the same thereafter.

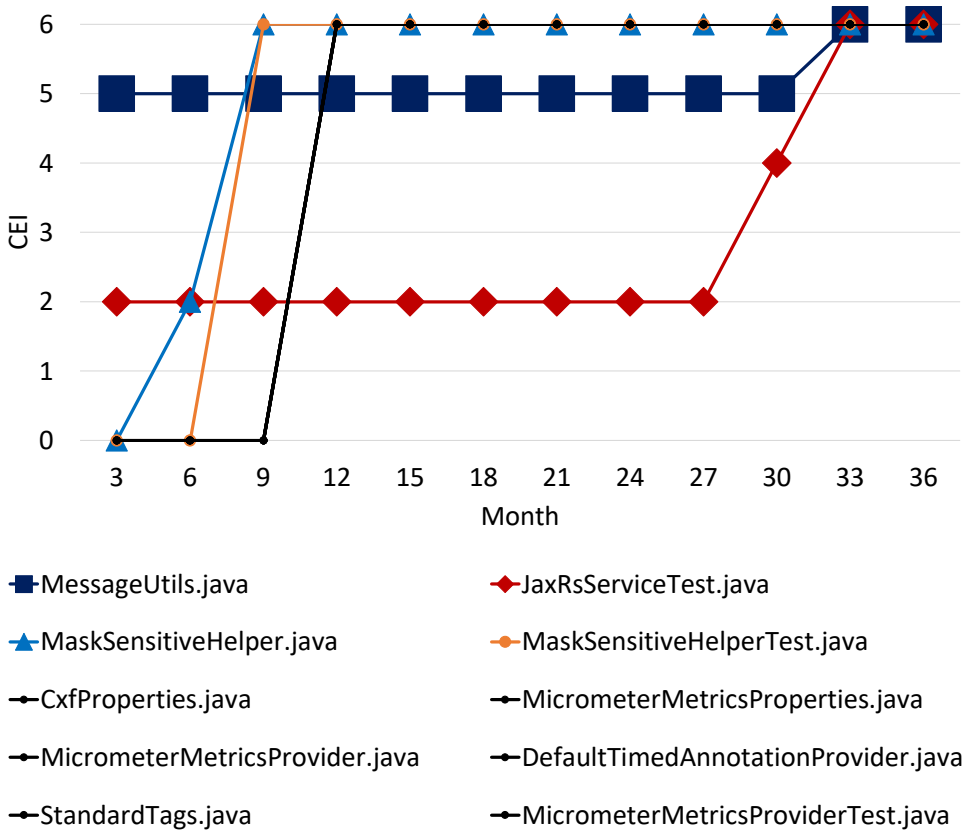

Figure 13: The increasing CEI trend for 10 files with the highest CEI from the *CXF* project throughout the 36 months period.

The final CEI value for all the top 10 files is 6 for this project. However, some files like *MaskSensitiveHelper.java* and *MaskSensitiveHelperTest.java* have a sharply increasing CEI from 0 to 6, whereas some other files already have a nonzero CEI within the first 3 months. For instance, one of the files, *MessageUtils.java*, have a CEI of 5 throughout the first 30 months. It increases to 6 at the end of the 33^{rd} month.

Table 8 lists 10 files with the number of times they are changed to close an issue and their ranking based on the number of these changes. They all have 6 as the last CEI value.

Table 8: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *CXF* project.

File Name	**Last CEI**	**Change Count**	**Change Rank**
JaxRsServiceTest.java	6	5	8
StandardTags.java	6	5	11
MessageUtils.java	6	5	12
CxfProperties.java	6	4	16
MaskSensitiveHelper.java	6	4	19
MaskSensitiveHelperTest.java	6	4	20
MicrometerMetricsProperties.java	6	3	35
MicrometerMetricsProvider.java	6	3	36
DefaultTimedAnnotationProvider.java	6	3	37
MicrometerMetricsProviderTest.java	6	3	39

We observe that the trend in the increase of CEI (i.e., whether it stays high all the time or whether it suddenly increases from low to high values) does not have any correlation with the number of issue-related changes. On the other hand, high CEI designates files that are associated with a relatively higher number of issue-related changes. The correlation between high CEI and high number of such changes is weaker for the *CXF* project. However, we should not that the variance in CEI is also low. 6 of out of the top 10 files are outliers, which are changed at least 4 times to close an issue (See Figure 12). Yet, high CEI captures top 5% of the files in terms of their issue-related change ranks. The lowest ranking is 39 among 806 files.

We calculate the Spearman's ρ between datasets for *CXF* also. Since both datasets have tied ranks (all values are the same), the calculation of Rs will result in a denominator of zero, leading to an undefined value or NaN (Not a Number). When the Spearman's rank correlation coefficient is equal to NaN, it indicates that there is either no variation or a perfect tie in the ranks of the data. This can occur when all the values in one of the dataset are the same, leading to undefined or meaningless correlation. This occurs because the formula for rs involves dividing by the standard deviation, which becomes zero in this case.

4.6 Tika Project

The number of changes is even smaller for the *Tika* project as can be observed in Figure 5, above and Figure 14 in the following. There is only a single file that was changed 12 times. The vast majority of the files (more than 100 of them) were changed 2 or 3 times to close an issue.

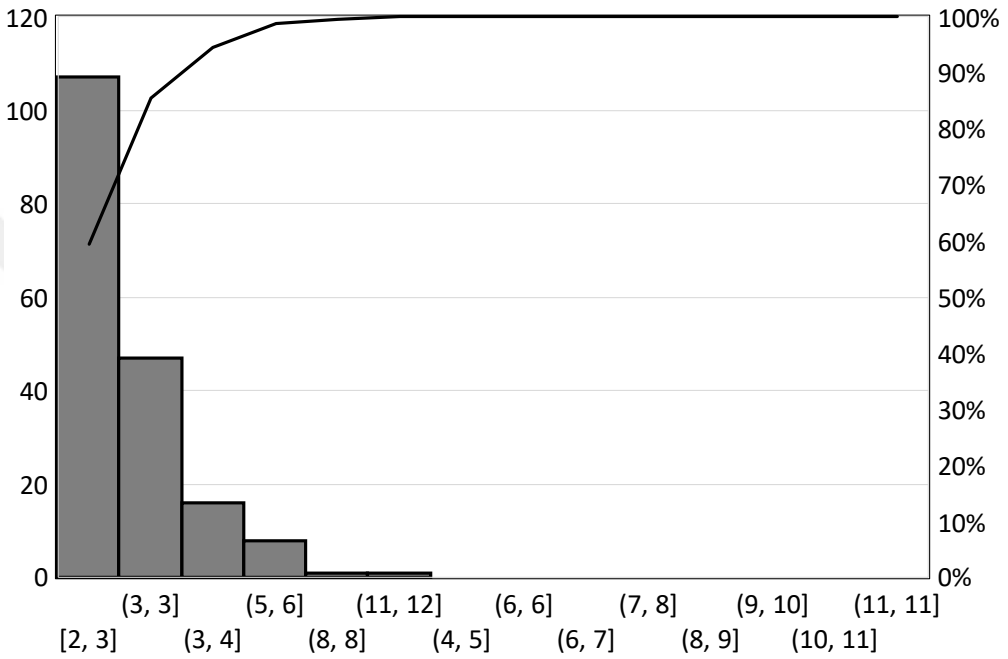

Figure 14: The distribution of the number of issue-related changes for the files of the *Tika* project that changed 2 or more times in association with a merged and closed PR.

Figure 15 depicts CEI of top 10 files of *Tika* over the 36 months period. All the files for the *Tika* project show very similar or the same trends regarding CEI. They all start with a 0 CEI value, quickly increasing up to 12 or 13 until the 18^{th} month, and remaining the same thereafter.

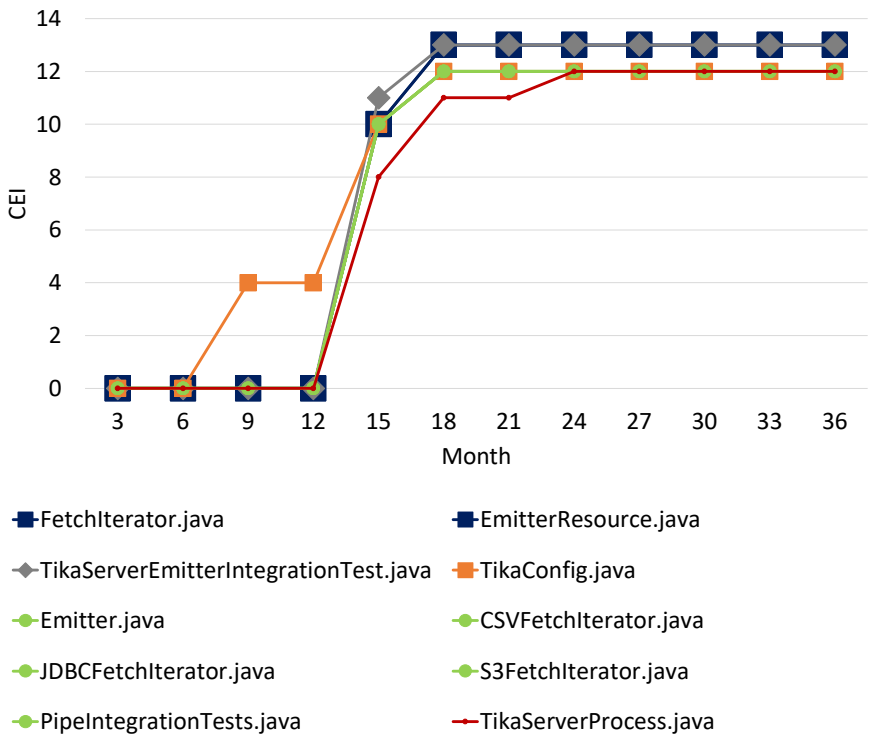

Figure 15: The increasing CEI trend for 10 files with the highest CEI from the *Tika* project throughout the 36 months period.

Table 9 lists these 10 files with their final CEI together with the number of times they are changed to close an issue and their ranking based on the number of these changes. We do not observe a correlation among the file rankings for the *Tika* project except a few files, namely *FetchIterator.java*, *TikaConfig.java* and *Emitter.java*, which are ranked as 57, 2 and 55, respectively. Note that there are 728 files involved in merged and closed PRs in total for this project. The lowest ranking is 420 and 422 for the last two files listed in Table 9. These files are changed only once to close an issue. Moreover, two of the files with the heighest CEI, *EmitterResource.java* and *TikaServerEmitterIntegrationTest.java* are not associated with any PR at all. Therefore, change count and the corresponding ranking cannot be provided for these files.

Table 9: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *Tika* project.

File Name	**Last CEI**	**Change Count**	**Change Rank**
FetchIterator.java	13	3	57
EmitterResource.java	13	-	-
TikaServerEmitterIntegrationTest.java	13	-	-
TikaConfig.java	12	8	2
Emitter.java	12	3	55
CSVFetchIterator.java	12	2	134
JDBCFetchIterator.java	12	2	135
TikaServerProcess.java	12	2	148
S3FetchIterator.java	12	1	420
PipeIntegrationTests.java	12	1	422

We analyzed these files and the repository of this project more closely to find out the reasons for these unexpected results. We found out that the project was subject to a major refactoring, during which many files were removed and new files were introduced. For instance, *EmitterResource.java* was removed from the project in May, 2021. The file named *TikaServerEmitterIntegrationTest.java* was renamed as *TikaServerPipesIntegrationTest.java* in the same month. Therefore, CEI remained to be the same for these and many other files during the second half of the 3-year period. The number of issue-related changes is also low for the *Tika* project in general. Files were changed mostly 2 or 3 times to merge and close a PR. The highest change count is 8 for the files listed in Table 9 (*TikaConfig.java*). The highest overall change count is 12 for one of the files, namely *TikaCLI.java* for the whole project (See Figure 5). The CEI of this file at the end of the 3-years period is 8. Therefore, it is not listed in Table 9 among the top 10 files. However, its CEI ranking is still among the top 1.6%, ranking 45 among 2806 files.

As mentioned, Tika was different from the other projects. In order to compare it with other projects in our study, we calculated Spearman's rank correlation coefficient in a similar manner. Spearman's ρ for *Tika* project is equal -0.3882. Results of the Spearman's correlation indicated that there is a non-significant very

small negative relationship between the datasets. The calculated Spearman's correlation coefficient, which measures the strength and direction of the relationship between the variables, is very close to zero and is negative in value. The term "non-significant" implies that the observed correlation is not statistically significant, meaning that it is likely due to random chance rather than a true underlying relationship between the variables. In other words, there is not enough evidence to conclude that the observed correlation is different from zero. However, since the correlation is close to zero and not statistically significant, the practical or meaningful significance of this relationship may be limited.

4.7 Iceberg Project

Figure 16 displays the distribution of the number of issue-related changes for the files in the *Iceberg* project. The maximum number of changes for a single file in this project is 60, observed only once. In contrast, 159 files were altered between 8 to 11 times.

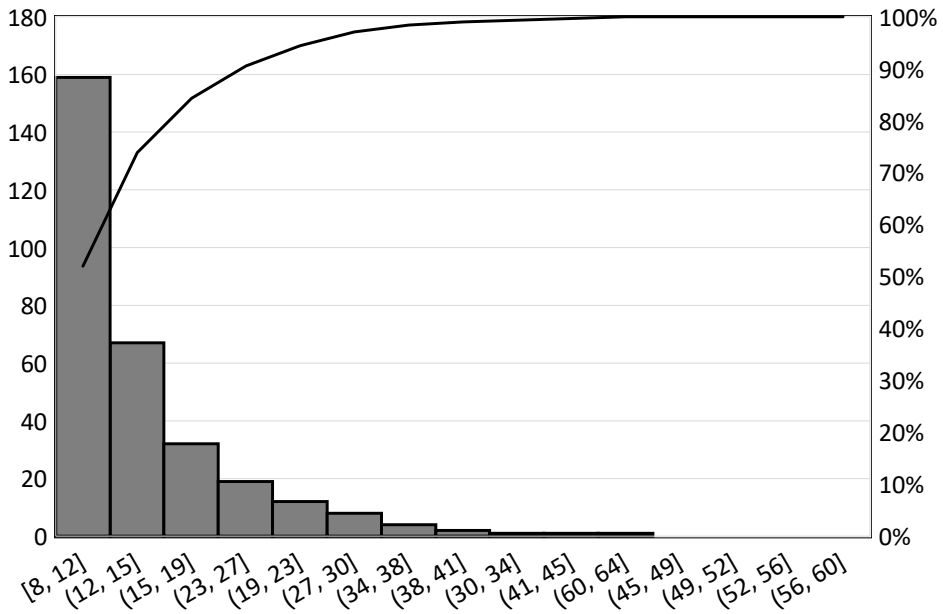

Figure 16: The distribution of the number of issue-related changes for the files of the *Iceberg* project that changed 8 or more times in association with a merged and closed PR.

Files with high CEI values in this project are those that have undergone a significant number of changes related to issues. Figure 17 below depicts CEI of top 10 files of *Iceberg* over the 36 months period.

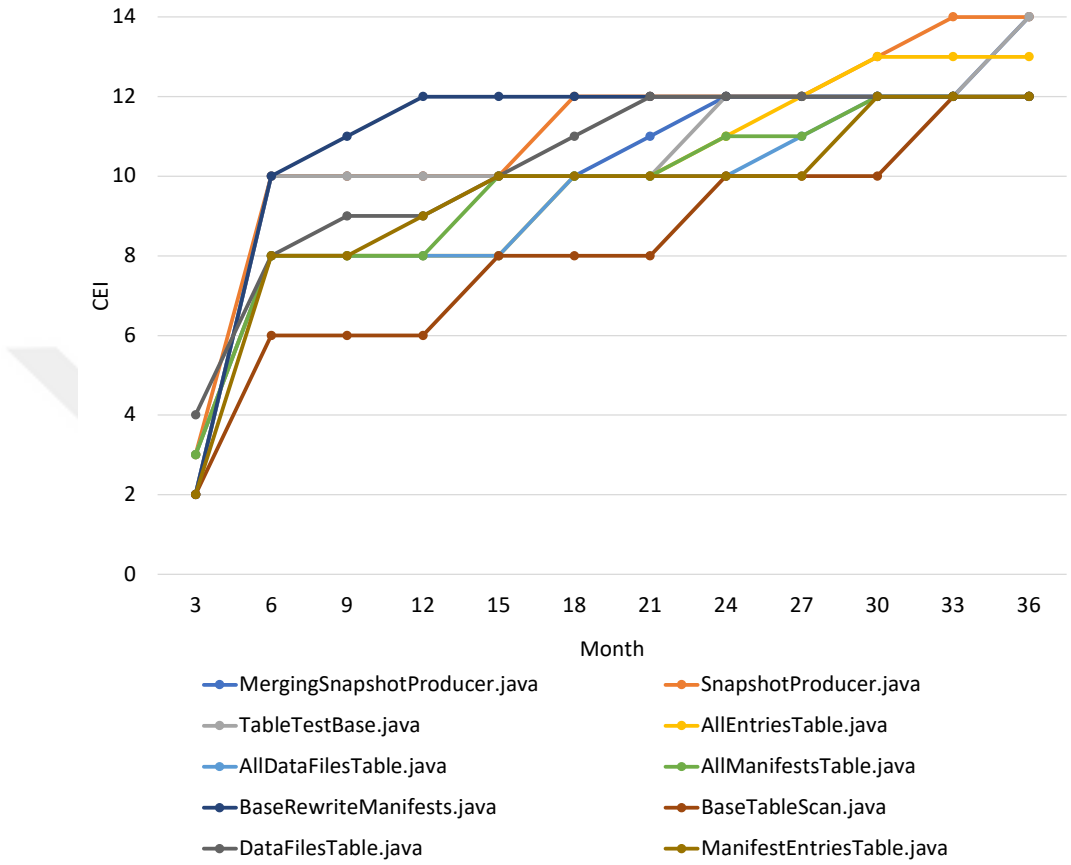

Figure 17: The increasing CEI trend for 10 files with the highest CEI from the *Iceberg* project throughout the 36 months period.

When evaluating the results of the project, a significant increase in the CEI value was observed from the 3rd to the 6th month. This increase was later saturated and the project continued with small increases until the 36th month on Figure 17.

Table 10: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *Iceberg* project.

File Name	**Last CEI**	**Change Count**	**Change Rank**
MergingSnapshotProducer.java	14	36	5
SnapshotProducer.java	14	35	6
TableTestBase.java	14	33	9
AllEntriesTable.java	13	23	31
AllDataFilesTable.java	12	22	38
AllManifestsTable.java	12	30	10
BaseRewriteManifests.java	12	20	46
BaseTableScan.java	12	34	7
DataFilesTable.java	12	26	19
ManifestEntriesTable.java	12	23	30

Table 10 lists top 10 files with the highest CEI values. The second column lists the number of times these files are changed as part of PRs on GitHub. The last column lists the ranking of files based on this count.

When considering the change rank, 4 out of the top 10 files also appear in the top 10 based on their CEI values. 46 among 3435 files corresponds to approximately the top 1.34% in terms of ranking.

The Spearman's ρ value for the Iceberg project is 0.62957. This value indicates a moderate positive correlation between the variables being compared. This suggests that there is a tendency for the variables to change in a similar direction, although the strength of the relationship is not extremely strong.

4.8 Drill Project

Figure 18 presents the distribution of issue-related changes for files in the *Iceberg* project. The highest number of changes observed for a single file in this project is 29, occurring only once. On the other hand, a significant number of files, 199, experienced alterations between 3 and 5 times.

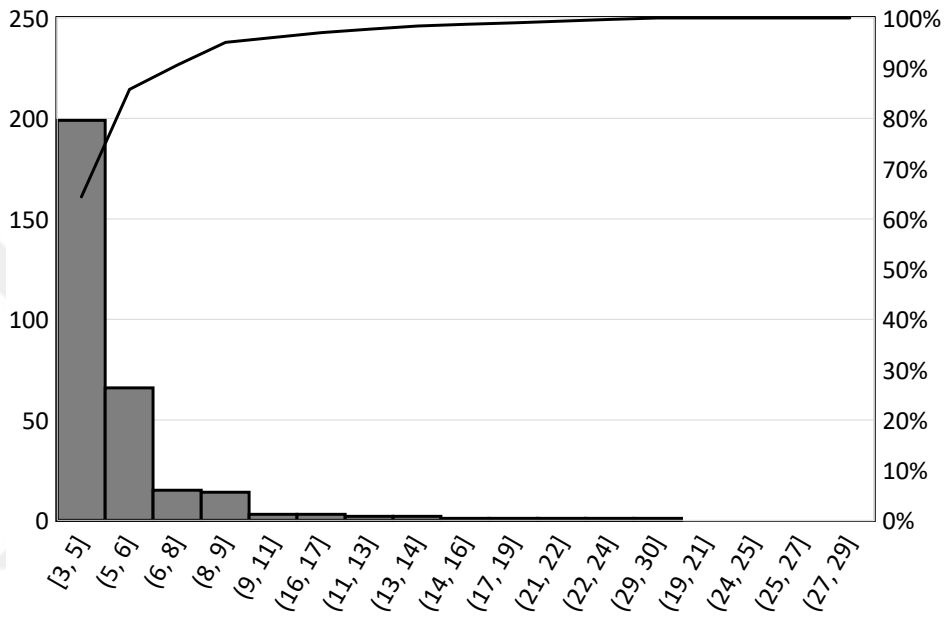

Figure 18: The distribution of the number of issue-related changes for the files of the *Drill* project that changed 8 or more times in association with a merged and closed PR.

Files with high CEI values in this project are those that have undergone a significant number of changes related to issues.

Figure 19 below depicts CEI of top 10 files of *Drill* over the 36 months period.

When evaluating the results of the project, a significant increase in the CEI value was observed from the 3rd to the 6th month. This increase was later saturated and the project continued with small increases until the 36th month on Figure 19.

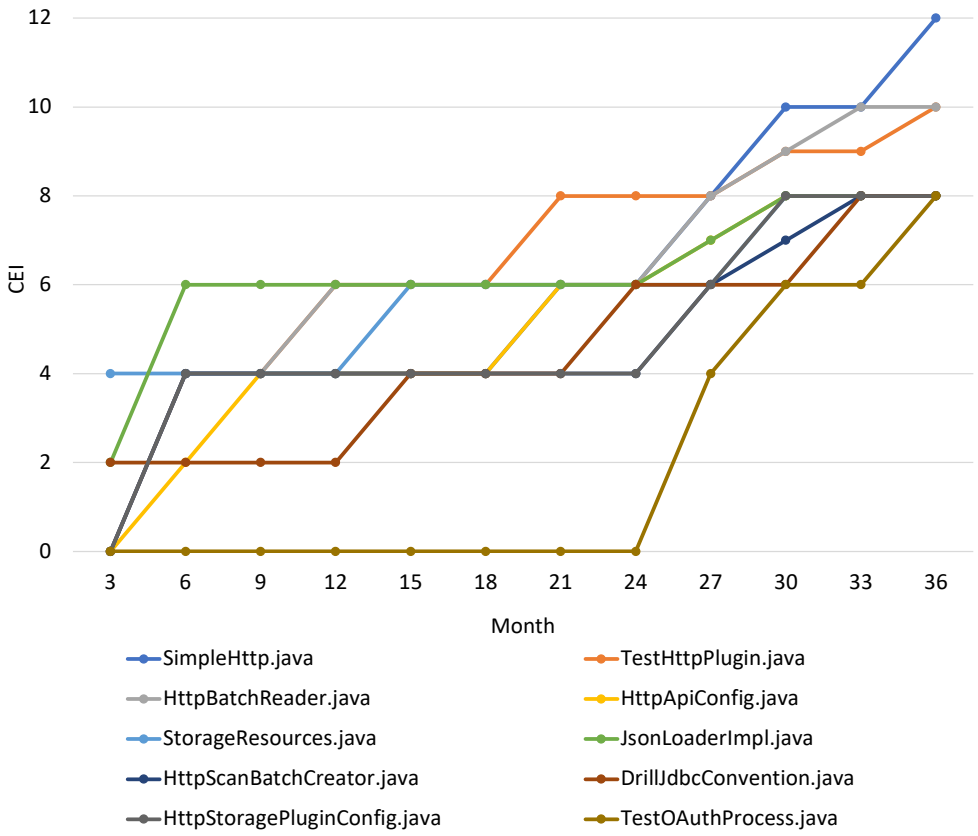

Figure 19: The increasing CEI trend for 10 files with the highest CEI from the *Drill* project throughout the 36 months period.

The highest CEI value, observed in the *SimpleHttp.java* file, has increased from 4 to 12 over a period of 3 years. When examining the *TestOAuthProcess.java* file, it is observed that the CEI value is 0 until the 24th month. In the 27th month, this value dramatically increases to 4, and in the 36th month, it further increases to 8.

Table 11 lists top 10 source files that are ordered based on their final CEI values. The second column lists the number of times these files are changed as part of PRs on GitHub. The last column lists the ranking of files based on this count.

Table 11: The final CEI, the number of issue-related changes and the corresponding ranking for 10 files with the highest CEI from the *Drill* project.

File Name	Last CEI	Change Count	Change Rank
SimpleHttp.java	12	29	1
TestHttpPlugin.java	10	23	2
HttpBatchReader.java	10	21	3
HttpApiConfig.java	8	17	5
StorageResources.java	8	15	8
JsonLoaderImpl.java	8	14	9
HttpScanBatchCreator.java	8	12	11
DrillJdbcConvention.java	8	12	12
HttpStoragePluginConfig.java	8	11	13
TestOAuthProcess.java	8	9	16

The total number of files involved in PRs for the *Drill* project is 1648. When looking at Table 11 based on the CEI values, TestOAuthProcess.java is the file ranked lowest with a change rank of 16 among the top 10 files. 16 among 1648 files corresponds to approximately the top 0.97% in terms of ranking.

Spearman's ρ for the *Drill* project is calculated as 0.8115. This value is significantly higher than that of all other software systems. *Drill* project indicates a strong positive correlation between the variables being compared. This suggests that there is a consistent and significant relationship between the datasets used in the analysis.

4.9 Discussion

We observed that the number of issue-related changes follows a Pareto distribution among source files. A small number of files account for the majority of the issue-related changes, whereas a large number of files are not modified to address any issue.

We also observed that high CEI values are correlated with a high number of issue-related changes. There were exceptional cases to this observation for the *Tika* project. However, we found out that this project was subject to major refactoring activities within the 3-years period we focus on, and the set of files included

in the dataset was drastically changed. Another exceptional observation was made for the *CXF* project. All the 10 files for this project (See Figure 13) have the same final CEI value (i.e., 6) and they were all modified 3 to 5 times in total. It is not feasible to derive meaningful interpretations for projects in which the number of changes, CEI values, and the variance of these values are minimal. However, relatively high CEI values are correlated with a high number of issue-related changes for large-scale projects that have a long track record of pull requests.

Table 12: Spearman's p values for all the subject systems.

System	**Spearman's p value**
Hadoop	0.74388
HBase	0.63246
TinkerPop	0.77394
CXF	N/A
Tika	-0.3882
Iceberg	0.62957
Drill	0.8115

We calculated Spearman's rank correlation coefficient, as shown in Table 12 (Spearman's ρ) [26] to assess the relationship between CEI and the number of issue related changes for *Hadoop*, *HBase*, *TinkerPop*, *Iceberg* and *Drill* projects. Spearman's ρ values turned out to be 0.74388, 0.63246, 0.77394, 0.62957 and 0.8115 for these five projects, respectively. These values indicate statistically significant associations. There are two exceptions to this observation. The first exception is the *Tika* project, where we see a negative correlation and the correlation does not turn out to be significant anyhow. However, this is due to the fact that the project was subject to a major refactoring, where many files were removed and new files were introduced. Moreover, CEI values do not vary significantly (either 12 or 13) among the source files in this project. The second exception is the the *CXF* project, for which the Spearman's ρ could not be calculated. Evaluation of correlation for this project does not make sense since CEI values are the same (equal to 6) for all the source files in this project.

The trends of increase in CEI were different among files for some projects although their final CEI values were the same or very close to each other at the end of the 3-years period. However, we could not observe any distinguishing correlation between the variance in these trends and the number of issue-related changes. Hence, we can summarize our observations as follows.

CEI of a module correlates with the number of issue reports associated with that module. The rate of increase in CEI does not necessarily have a significant impact on this correlation.

We focused on a 3-years period in our study. This time interval can be extended. However, one should be taking major refactoring activities into account, while calculating CEI. We observed one such case for the *Tika* project. The chance of coinciding with such cases would increase when the size of the time interval increases.

4.10 Threats to Validity

Our study is subject to an external validity threat since it is based on 7 systems, which are mostly implemented with Java. The empirical study can be extended by increasing the number and the variety of subject systems. There also exist construct and conclusion validity threats due to the employed dataset. CEI is inevitably correlated with the number of changes since it measures EC. We aim at measuring the correlation of CEI with issue-related changes in particular. The number of changes that are calculated based on committed files is not necessarily the same as the number of changes made for merging and closing a PR (See Table 4). We relied on the tags and labels on commit messages and JIRA tasks to create our dataset. If JIRA records were missing or inaccurate such as unlabeled/mislabeled, our results would be affected. The way that developers commit their changes also impacts the calculation of CEI. A lack of regular and frequent commits can lead to several unrelated files committed together, which would then

lead to false positives for EC. In our empirical study, we focused on a time period of 3 years. This time interval can be extended. Besides, we divided 3 years into 3-month intervals. This can also be changed to be meaningful on a project basis. However, in doing so, significant refactoring activities should be taken into account when calculating the CEI. We observed such a case with the *Tika* project. After the refactoring conducted to reduce TD, there may be files that cannot be found, files with changed names, or files that have been moved.

CHAPTER V

CONCLUSION

A new metric, CoEvolution Index (CEI), is proposed for capturing the relative evolutionary coupling (EC) of software modules by using a metric that is similar to the h-index. CEI of a module is equal to n, which is the number of times it is modified together with at least n other modules. It eliminates bias that can be imposed by many local modifications or a single commit that can involve a large number of unrelated modules. We developed a script that can calculate CEI for source files in a code repository. We investigated to what extent CEI of a source file correlates with the issues reported for that file. We analyzed the repositories of 7 software systems. We observed that a very small portion of files is involved in a large portion of changes that are associated with pull requests. We also conclude that CEI can be used for pinpointing these files. Thus, CEI can aid in the measurement of TD interest and the quantification of the obstacles regarding the maintainability and reusability of the relevant files.

The number and the variety of subject systems can be increased in our empirical study as future work. The size of the time interval considered in the study can also be extended. A high CEI or an increasing trend in CEI can be correlated with issues that are revealed at a later point in time. Analysis of issue-related changes in a moving time window can better reveal this impact.

REFERENCES

[1] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *Proceedings of the 16th Working Conference on Reverse Engineering*, pp. 135–144, 2009.

[2] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings of the International Conference on Software Maintenance*, pp. 190–198, 1998.

[3] S. Kirbas, T. Hall, and A. Sen, "Evolutionary coupling measurement: Making sense of the current chaos," *Science of Computer Programming*, vol. 135, pp. 4 – 19, 2017.

[4] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 411–420, 2011.

[5] R. Schwanke, L. Xiao, and Y. Cai, "Measuring architecture quality by structure plus history analysis," in *Proceedings of the International Conference on Software Engineering*, pp. 891–900, 2013.

[6] F. Fontana, R. Roveda, and M. Zanoni, "Technical debt indexes provided by tools: A preliminary discussion," in *Proceedings of the 8th IEEE International Workshop on Managing Technical Debt*, pp. 28–31, 2016.

[7] Y. Cai, L. Xiao, R. Kazman, R. Mo, and Q. Feng, "Design rule spaces: A new model for representing and analyzing software architecture," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 657–682, 2019.

[8] S. Kirbas, B. Caglayan, T. Hall, S. Counsell, D. Bowes, A. Sen, and A. Bener, "The relationship between evolutionary coupling and defects in large industrial software," *Journal of Software: Evolution and Process*, vol. 29, no. 4, 2017.

[9] Q. Alsarhan, B. S. Ahmed, M. Bures, and K. Zamli, "Software module clustering: An in-depth literature analysis," *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 1905–1928, 2022.

[10] A. Saydemir, M. Simitcioglu, and H. Sozer, "On the use of evolutionary coupling for software architecture recovery," in *Proceedings of the 15th Turkish National Software Engineering Symposium*, pp. 1–6, 2021.

[11] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National Academy of Sciences*, vol. 102, no. 46, pp. 16569–16572, 2005.

[12] L. Bornmann and H. Daniel, "What do we know about the h index?," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 9, pp. 1381–1385, 2007.

[13] W. Cunningham, "The WyCash portfolio management system," *SIGPLAN OOPS Messenger*, vol. 4, no. 2, p. 29–30, 1992.

[14] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016.

[15] E. Allman, "Managing technical debt: Shortcuts that save money and time today can cost you down the road," *Queue*, vol. 10, no. 3, p. 10–17, 2012.

[16] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

[17] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE Software*, vol. 29, no. 6, pp. 22–27, 2012.

[18] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In search of a metric for managing architectural technical debt," in *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pp. 91–100, 2012.

[19] P. Avgeriou et al., "An overview and comparison of technical debt measurement tools," *IEEE Software*, vol. 38, no. 3, pp. 61–71, 2021.

[20] Z. Li, P. Liang, P. Avgeriou, N. Guelfi, and A. Ampatzoglou, "An empirical investigation of modularity metrics for indicating architectural technical debt," in *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, p. 119–128, 2014.

[21] Y. Cai and R. Kazman, "DV8: automated architecture analysis tool suites," in *Proceedings of the IEEE/ACM International Conference on Technical Debt*, pp. 53–54, 2019.

[22] L. Xiao, Y. Cai, and R. Kazman, "Titan: A toolset that connects software architecture with quality analysis," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, p. 763–766, 2014.

[23] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A case study in locating the architectural roots of technical debt," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering*, vol. 2, pp. 179–188, 2015.

[24] W. Snipes, S. Karlekar, and R. Mo, "A case study of the effects of architecture debt on software evolution effort," in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 400–403, 2018.

[25] T. Ball, J. Kim, A. Porter, and H. Siy, "If your version control system could talk," in *Proceedings of the ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, vol. 11, 1997.

[26] C. Spearman, "The proof and measurement of association between two things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

VITA

Hüseyin YAPICI received his Bachelor of Science degree in Computer Engineering from Dokuz Eylül University in 2019 as Honor Student. Since 2010, he has worked at technology startups and companies. He is currently working as an iOS Team Lead & Senior Software Engineer for iOS mobile application development in the IoT department of one of the leading R&D companies in Türkiye. He mentors and supports tech startups. His research focuses on software engineering, software architecture design, IoT, and mobile technologies. Besides his professional career, he works as a volunteer in non-governmental organizations in various positions to simplify human life with technology, spread knowledge, and give back to society. Moreover, IEEE Computer Society Upsilon Pi Epsilon '20 was awarded to Hüseyin Yapıcı for his academic achievement, his contributions to giving back to society through computer science and engineering at locally and global wide. He has been also honored with the prestigious IEEE Computer Society Richard E. Merwin Student Scholarship in 2022.