Journal of Software Engineering Research and Development, 2024, 12:17, doi: 10.5753/jserd.2024.4118

Studying the Impact of CI/CD Adoption on Atoms of Confusion Distribution and Prevalence in Open-Source Projects

Diego N. Feijó| Federal University of Ceará | *diegofeijo@alu.ufc.br*]
Carlos D. A. de Almeida| Federal University of Ceará | *diego.andrade@ufc.br*]
Lincoln S. Rocha| Federal University of Ceará | *lincoln@dc.ufc.br*]

Abstract

Atoms of Confusion (AoC) are indivisible code patterns that may confuse developers when trying to understand them, and that have less confusing equivalent patterns. Previous works suggest it is a good practice to avoid them. While there are studies on AoC relating them to bugs, there is not much about their relationship with the practices of Continuous Integration and Continuous Delivery (CI/CD). Since CI/CD is generally praised as a group of good practices, related to better code being released reliably and faster to clients, there is a possibility that the presence of CI/CD would also impact the presence of AoC, possibly making them less prevalent since they can be problematic to development processes. To clarify this relationship, we analyzed 10 open-source long-lived Java libraries and 10 open-source Java projects for Android, to see if there was any difference in the AoC rate, diffusion, and density before and after the implementation of CI/CD when comparing each project, the average total. We also analyzed the atoms separately, checking for the most and least prevalent. Our results show the metrics have considerably changed for all projects when checked separately, and less so on average, but we could not find a statistically relevant relationship between most of these changes and CI/CD. We found a significant relation when checking the growth rate on one of the metrics. We also found that the most prevalent atom is the `Logic as Control Flow`, and the least is the `Arithmetic as Logic`.

Keywords: *Software Engineering, Atoms of Confusion, Continuous Integration, Continuous Delivery, Mining Repositories, Open-Source Software*

1 Introduction

Software Engineering is trying to find solutions to reach both fast software production and high-quality software product delivery. This is also true for open-source projects widely used by end users and companies around the world (Øyvind Hauge et al., 2010; Lenarduzzi et al., 2020), and they keep growing in scale. As such, developers should learn and adopt practices and methods that would help them in this regard.

Continuous Integration (Fowler and Foemmel, 2006) and Continuous Delivery (Humble and Farley, 2010) are a set of practices in software engineering, which consists of pipelines that automatize the building, testing, and delivery processes of software, making it a faster and more reliable way to produce valuable software artifacts in short cycles (Chen, 2015).

Since quickness is relevant, it is undesirable to lose more time than necessary on activities such as understanding source code, which is an essential software developing task, but also very time-consuming, having developers spend 50% of their time on it (Minelli et al., 2015; Xia et al., 2018). One factor that can increase the time spent on such activities is confusion. In summary, confusion is the lack of certainty a developer has about the execution of a piece of code. To avoid consuming even more time on this kind of task, it is important for projects to minimize the proportion of confusing code they contain.

Considering this, an Atom of Confusion (AoC) was first defined as the smallest piece of code that may confuse developers (Gopstein et al., 2017). An example of AoC is the `Pre-Increment/Decrement`, an atom candidate for both C and Java languages, which happens when an increment (or decrement) of a variable happens before it is assigned in the same line of code (e.g., $a = ++\ b$). There is also the `Post-Increment/Decrement`, where the increment/decrement occurs after the assignment (e.g., $a = b{+}{+}$).

Since this initial definition, previous works have already studied the prevalence of AoC (Gopstein et al., 2018; Mendes et al., 2022; Tahsin et al., 2023), and even their impact during development (Gopstein et al., 2018; Bogachenkova et al., 2022; Pinheiro et al., 2023). Results vary, but the presence of atoms is generally considered a problem. Loosely related, there also have been studies aiming to understand the presence of CI/CD and the possible impact it has on variables regarding software development (Almeida et al., 2022; Fairbanks et al., 2023; Liu et al., 2023).

By definition, AoC are a source of code misunderstanding, impacting software development and maintenance tasks. Meanwhile, CI/CD is a well-known practice in Software Engineering that seeks to increase software quality and speed up software delivery. However, there is a gap in research regarding the relationship between AoC and CI/CD that may be relevant. Since AoC can confuse, they could also cause problems related to slowness in software comprehension, maintenance, evolution, and even the introduction of bugs since the code becomes more difficult to understand. These are problems that CI/CD adoption generally aims to solve or mitigate by using code reviews and code quality checkers, including static code analysis. Therefore, it is reasonable to suppose a connection between the adoption of CI/CD practices in a software project and the presence of AoC in such a project. More specifically, the CI/CD practices may reduce the presence of AoC or, at least, slow down their growth.

Therefore, to fill this gap, we conducted an empirical study to gather evidence on whether the practice of CI/CD has any impact on the distribution and prevalence of atoms of confusion in open-source projects. To achieve this, we establish a set of metrics to serve as a proxy for measuring the distribution and prevalence of AoC. Both metrics were calculated from data extracted from 20 open-source Java projects (10 long-lived libraries and 10 Android projects) using static code analysis tools developed on top of Spoon (Pawlak et al., 2016). This study extends our previous study (Feijó et al., 2023) that deeper the investigation and includes new analyses to support our findings and draw our conclusions.

The first set of metrics is used to measure the distribution of AoC and it is composed of AoC Rate (`ACR`), which we define as the number of AoC per line of code; AoC Diffusion (`ACDIF`), which we define as the percentage of classes that contain AoC; and AoC Density (`ACDEN`), which we define as the number of AoC per class that contains atoms. The second set is intended to measure the prevalence of AoC, it is: AoC Frequency Percentage (`FP`), which we define as the number of releases that contain a type of atom divided by the total amount of releases; and AoC Relative Percentage (`RP`), which we define as the number of atoms of a certain type divided by the total number of atoms.

All the 20 projects (and their releases) used as subjects in our empirical investigation were selected by well-defined criteria to guarantee their relevance and fulfill this paper's goal. The data analysis was made on the projects separately and as a group on average. We used the Wilcoxon Signed-Rank Test to check the statistical significance when comparing the grouped data before and after CI/CD, and a data distribution analysis to better visualize the data within the same periods, both on average and for each project, using plots.

Our findings reveal a lack of statistically relevant connection between CI/CD adoption and the AoC distribution metrics in the studied projects, except for the geometric mean of the `ACDIF`, which got lower after CI/CD. Regarding the AoC prevalence metrics, considering both before and after CI/CD, the results showed us that the most prevalent atom is the `Logic as Control Flow`, both in `FP` and `RP`, while the least prevalent is the `Arithmetic as Logic`, if not counting atoms that are not present. We also found that the `FP` of all types of atoms increased when comparing before and after CI/CD.

This paper is divided into the following. In Section 2 we explain the fundamentals and theory behind Atoms of Confusion and CI/CD that serve as the base for this study. Section 3 describes the methodology we used to get our data and the projects from which we extracted it, showing the important metrics and the process we utilized to get our results. In Section 4, we show the results we got from our analyses and discuss them while properly answering our research questions, followed by their implications for future works and developers. In Section 5, we discuss the possible threats to the validity of our study. In Section 6, we discuss the related work and, in Section 7, we present the paper's final remarks, followed by Section 8 where we reference the repository where our data and code is available.

2 Background

2.1 Atoms of Confusion

Confusion is defined by Gopstein et al. (2017) as when the developer's interpretation of a piece of code differs from the machine's interpretation. In other words, a confusing code executes in a way developers do not expect. Following that, Gopstein et al. (2017) also defined the concept of Atoms of Confusion for the first time as the smallest piece of code capable of confusing developers. Castor (2018) complemented the definition of Atoms of Confusion as easily identifiable and indivisible patterns of code that are likely to confuse developers, and that have an equivalent block of code that is less likely to confuse.

While the original study and list of Atoms of Confusion was based on the C programming language (Gopstein et al., 2017), different programming languages have different Atoms of Confusion candidates, and there are already studies that proposed their lists of AoC candidates for specific programming languages (Castor, 2018; Langhout and Aniche, 2021; Torres et al., 2023; Costa et al., 2023).

By definition, AoC can be transformed into other patterns of code that function in the same way and which are less likely to be confusing, making the use of AoC unnecessary. However, they are still prevalent in software projects from various contexts (Gopstein et al., 2018; Mendes et al., 2022), even finding themselves overlapping with recommendations from popular code style guides (Gopstein et al., 2017).

AoC tend to grow in numbers when the project grows in size, even disproportionally so when compared to the increase in lines of code (Mendes et al., 2022). They were also shown to be connected to the presence of bugs and frequently appeared in bug fixes (Gopstein et al., 2018), pointing to the possibility of them being dangerous and a problem that projects may want to get rid of. However, their impact is lacking, or at least unclear, in the context of code reviews, seemingly not causing confusion comments, nor being removed on pull requests (Bogachenkova et al., 2022).

Table 1 shows all the types of atom candidates from the Java programming language we analyzed throughout this paper, with a code example of the atom, and of the less confusing equivalent.

2.2 Continuous Practices in a Nutshell

Continuous Practices are a series of software engineering practices whose main purpose is to get changes made on code verified and into production, or into the hands of users and customers, safely and quickly in a sustainable way (Humble, 2017). These practices are often divided into Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment (CD), and are often summarized as CI/CD or just CD.

CI/CD is closely related to the presence of pipelines that automate recurring and repetitive tasks related to building, deploying, and testing software. There are CI/CD services such as Jenkins, CircleCI, Travis, and GitHub Actions that can be used to better coordinate the process involving continuous practices and quality checks on the application

Table 1. List of Atoms of Confusion identifiable by the BOHR tool. Adapted from Mendes et al. (2022)

Atom of Confusion Name	Acronym	Snippet with AoC	Snippet without AoC
Infix Operator Precedence	IOP	`int a = 2 + 4 * 2;`	`int a = 2 + (4 * 2);`
Post-Increment/Decrement	Post-Inc/Dec	`a = b ++;`	`a = b ;` `b += 1 ;`
Pre-Increment/Decrement	Pre-Inc/Dec	`a = ++b ;`	`b += 1 ;` `a = b ;`
Conditional Operator	CO	`b = a == 3 ? 2 : 1;`	`if ( a == 3){ b = 2;}` `else {b = 1;}`
Arithmetic as Logic	AaL	`(a - 3) * (b - 4) != 0`	`a != 3 && b != 4`
Logic as Control Flow	LaCF	`a == ++a > 0 || ++b > 0`	`if (!( a + 1 > 0 )) {b += 1;}` `a += 1`
Change of Literal Encoding	CoLE	`a = 013;`	`a = Integer.parseInt("13", 8);`
Omitted Curly Braces	OCB	`if (a) f1 ( ); f2 ( );`	`if (a){ f1( ); } f2( ) ;`
Type Conversion	TC	`a = (int) 1.99f;`	`a = (int) Math.floor(1.99f);`

Repurposed Variables	RV	`int a [] = new int[5];`
		`a[4] = 3;`
		`while (a[4] > 0) {`
		`  a[3 - v1[4]] = a[4];`
		`  a[4] = v1[4] - 1;}`
		`System.out.println(a[1]);`
			`int a [] = new int[5];`
			`int b = 5 ;`
			`while (b > 0) {`
			`  a[3 - a[4]] = a[4];`
			`  b = b - 1;}`
			`System.out.println(a[1]);`

source code and other artifacts, before being used in production (Shahin et al., 2017).

According to Chen (2015), the implementation of CI/CD offers several benefits such as reduced deployment risk and increased release frequency. These benefits lowered costs and made getting user feedback much easier and faster. Old release practices required too much effort and time to use, and often caused troubles. CI/CD eliminated these issues. These benefits are corroborated by Itkonen et al. (2016) while adding that both customers and developers could perceive them.

In an effort to properly separate and define the continuous practices and DevOps, Stahl et al. (2017) proposed a few ways to understand the practices, as a set of separate definitions. This was done because of the great ambiguity these terms have both in the industry and in the literature. Continuous Integration is the frequent integration of developers' works, usually daily at least. This is a practice dependent on the developers' behavior since they are the ones who actively integrate their work. Continuous Delivery is treating each change made to the project as a potential release candidate, in other words, it needs to be properly tested and verified by a continuous delivery pipeline. This is a development process, not connected to a developer's actions since the pipeline is automatized. In fact, it is possible for a Continuous Delivery pipeline to be present, while some developers do not act with Continuous Integration in mind, that is, they may not integrate their work quickly as per the CI practice. Continuous Deployment is constantly and rapidly placing release candidates, previously evaluated during Continuous Delivery, in a production environment, usually for customer use.

According to these definitions, our focus is on the Continuous Delivery aspect of CI/CD, or just CD. Liu et al. (2023) often uses the complete acronym. Still, since their study's method of identifying CI/CD was the presence of CI/CD services and pipelines, it implied their focus is the same as ours in this aspect.

3 Study Methodology

3.1 Research Goal and Question

The main goal of this paper is to know the impacts that CI/CD may have on the distribution and prevalence of AoC. To guide our study, we asked the following research questions:

RQ1. Is there a statistically significant difference between the distribution of Atoms of Confusion before and after CI/CD adoption in open-source Java projects?

CI/CD is often related to faster releases and automation by pipelines (see Section 2.2). A CI/CD pipeline often has checkers for code quality for example. Since AoCs are undesirable pieces of code, there is a possibility that CI/CD and its practices have an impact on their distribution. Our objective with this question is to know if there is such an impact. To see if there was an impact, we first needed metrics that we could use as proxies for the distribution of AoC, to compare before and after the implementation of CI/CD. The three metrics we chose are the AoC rate, diffusion, and density (see Section 3.3).

RQ2. Is there a difference in the prevalence of different types of Atoms of Confusion before and after CI/CD adoption in open-source Java projects?

While our intentions with this question are not different from the first one, our objective this time is to check if there is an impact caused by the implementation of CI/CD for each type of AoC. The different types of AoC have various degrees of prevalence, with some being very rare and others very common. In fact, some types of AoC are even recommended to be used in code style guides as mentioned before. These differences may imply that the quality checks of CI/CD focus on some atoms but not all atoms. To answer this question, we used metrics as proxies for the prevalence of each type of AoC in terms of frequency and relative percentages (see more details in Section 3.4).

With these goals in mind, we first selected our projects following a set of criteria (see Section 3.2), for a total of 20 projects. We then extracted the information from these projects with two static analysis tools, BOHR (Mendes et al., 2022)[1] and JMetrix[2], to acquire the data to be used during the analysis to get the results we are searching for. Finally, we discuss the different results and their implications.

3.2 Selection of Projects

We select 20 open-source projects to serve as subjects of our empirical investigation, divided into two groups: long-lived Java libraries (in the first 10 rows of Table 2) and Java-based Android projects (in the last 10 rows of Table 2). All of them are freely available on GitHub, and were chosen based on the following criteria, inspired by the ones used in (Fairbanks et al., 2023): (i) the projects must have adopted CI/CD at some point in their development; (ii) the projects must have been active as recently as 2022; (iii) they must have at least 25 stars and at least 2 contributors, to avoid personal and school projects; and (iv) the projects must have at least 8 release version tags, otherwise the project's history with CI/CD and AoC would be too short, threatening the statistical analysis.

For the Java libraries, we chose some of the projects from Almeida et al. (2022). These repositories were curated from the SmartShark dataset[3], and filtered by checking the presence and proper use of CD. They are all from Apache. We filtered 8 projects from the 25 total. Some projects that passed most but not all the filters in the work of Almeida et al. (2022) were manually checked for the presence of CI/CD and were considered candidates for our study if they passed the criteria we established. We chose 2 projects using this method.

For the Android projects, we manually filtered the projects identified by Liu et al. (2023) that adopted a CI/CD service. Their dataset contained a large number of Android repositories from three different sites. We limited ourselves to repositories on GitHub, totaling around $4,000$. We then used Python, and the ghAPI[4], a Python library, to communicate with GitHub's API and create the main filtering, following our selection criteria. After filtering the repositories, we were left with 240 eligible projects to select from. With these 240 eligible projects, we then randomly selected a project from it and had to manually check for imperfections in our filtering. For example, we took out projects that had CI/CD implemented, but the implementation happened too early or too late in their development process, as such, they had little to no history with CI/CD, or little to no history without, which would generate bad comparisons or no comparisons at all. As such, we took out projects with three releases or less before and after the implementation of CI/CD. A large number of projects were filtered because of this. We also took out projects that had 10% or more of other programming languages, since our tools only worked with Java code. There were also cases of projects that did not appear to have any CI/CD when manually checked. When a selected project wasn't discarded for the above reasons, it was added to our project list, and this process was repeated until we got 10 Android projects.

Table 2. GitHub's Information of Studied Projects.

Project	#Stars	#Tags	#Commits
commons-lang	$2,503$	96	$7,270$
commons-dbcp	312	66	$2,827$
struts	$1,216$	143	$6,658$
commons-codec	409	44	$2,423$
commons-bcel	216	35	$2,508$
commons-compress	282	76	$4,129$
commons-configuration	179	77	$3,857$
commons-net	211	75	$2,918$
freemarker	874	44	$2,343$
commons-vfs	197	55	$3,656$
infinity-for-reddit	$3,461$	118	$2,033$
gestureviews	$2,325$	16	432
discreet-launcher	167	61	639
xupdate	$2,134$	32	246
colorpickerview	$1,417$	19	278
opentracks	678	141	$5,311$
presencepublisher	70	50	206
asteroidossync	91	28	$1,088$
unexpected-keyboard	619	28	616
shitter	197	108	$1,678$

3.3 AoC Distribution Metrics

As seen earlier in Section 3, the metrics we chose are the AoC rate, diffusion, and density. We intended to use these metrics because they are less dependent on the size of the project, as there is a tendency for the number of AoC to grow as the projects grow in size (Mendes et al., 2022). So, if we only compared the raw number of AoC, this tendency would overwhelm any possible impact CI/CD could have.

For the AoC rate, since it is the number of AoC per line of code, it can represent the distribution of AoC independently from the project size. The diffusion is a percentage that shows how spread out the atoms are in the project classes, with a 100% percentage indicating that all classes have at least one atom. The density shows how many atoms are contained within a single atom-containing class, on average, and this can also show us how spread out the atoms are, but differently from the diffusion: if the density does not increase

[1] https://github.com/wendellmfm/bohr
[2] https://github.com/lincolnrocha/JMetriX
[3] https://smartshark.github.io/
[4] https://ghapi.fast.ai/

much, or even decreases, while the atoms are increasing, that may signify that the atoms "infected" more classes.

3.3.1 Rate

To compare the AoC rate (ACR) between the periods with and without CI/CD, we first determined how that would be measured. For that, we extract two metrics from each considered release: the number of atoms of confusion (NAC) and the number of lines of code (LOC). Next, we calculate the ratio between NAC and LOC to compute the AoC rate (ACR = NAC ÷ LOC). We compute the ACR metric for all considered releases in each project and group it into two periods, AoC rate before and after CI/CD adoption. Finally, we compute the statistical mean and median of the ACR metric for each period and project, and use them as proxies to statistically compare the rate of AoC before and after CI/CD adoption across the studied projects.

3.3.2 Diffusion

For the AoC diffusion (ACDIF), we also extracted the two metrics necessary for its calculation, the number of classes in total (NCT) and the number of classes with atoms (NCA). Next, we calculated the ratio between NCA and NCT to get the ACDIF proper (ACDIF = NCA ÷ NCT). We then repeated what was made with the ACR, computing the metric for the releases of each project, and then separating between before and after CI/CD adoption to get the mean and median.

3.3.3 Density

The process is the same for the AoC density (ACDEN). The necessary metrics for its calculation were already used for the past two metrics, the NAC and the NCA, and the density is the ratio between NAC and NCA (ACDEN = NAC ÷ NCA). The rest of the process is the same as the other two metrics.

3.4 AoC Prevalence Metrics

The frequency percentage (FP) and the relative percentage (RP) were used for the analysis of specific atom types. These are also resistant to the impact of project growth since they are percentages. The frequency percentage checks the prevalence by representing how frequently an atom type appears considering all projects and their release versions, while the relative percentage represents how frequent the atoms are relative to the total of atoms.

For the FP, we first get the total number of releases spanning all projects and then, from this number, we count the ones which contain the type of atom. The FP is the ratio between the releases that contain the type of atom and the total number of releases. If the FP is 100%, that means that type of AoC is present in every analyzed release.

The RP is different. While it is still a percentage, it is relative to the total number of atoms. To calculate it, we divide the quantity of the specific atom type by NAC for each release. But since we want a general metric, we then calculate the mean value considering all the releases. This value will represent the prevalence of an atom type when considering all other types. A value of 100% would mean that all atoms are of the same type, which is unlikely to happen.

3.5 Determining Before and After CI/CD

To compare the AoC Metrics before and after CI/CD, we had to know when CI/CD started for each project. For this, we manually checked all projects and their releases while using the criteria from Liu et al. (2023), i.e., we searched for the configuration file of a CI/CD service (see Section 2.2). These files are usually .yml or .yaml files where the configuration of a CI/CD service is stored (e.g., the `travis.yml` for the TravisCI[5]). The release when the file first appeared was considered the release where CI/CD started. If multiple CI/CDs were used in a single project, we considered the first one used as the start of CI/CD. All releases from when CI/CD started to the most recent we downloaded we consider "After CI/CD", while all releases older than the release when CI/CD started we consider "Before CI/CD".

3.6 Data Mining

The mining process started with the project selection since we had to filter more than 4000 projects by mining their repositories and checking their information to see if they were valid candidates for our study according to our criteria previously mentioned in Section 3.2.

After choosing projects, we downloaded several release versions from each, trying to balance the number before and after the implementation of CI/CD. These releases had their code statically analyzed with Java programs made with Spoon: BOHR and JMetriX. Spoon is an open-source Java library that analyzes and transforms Java source code. BOHR, created by Mendes et al. (2022), is a tool made to identify AoC and extract data related to them from the source code of Java projects. The AoC it can identify are based on the AoC list for the Java programming language suggested by Langhout and Aniche (2021), and are shown in Table 1, as well as the equivalent less-confusing pattern for each. JMetriX is a tool made to extract general metrics and information about Java source code. It was used to extract the number of lines of code (LoC) metric from the projects.

3.7 Data Analysis

After the mining process, we got the number of LoC, the total number of AoC, the number of classes in total and with atoms, and the number of AoC of each type, for each release of each project we selected. Next, we computed the AoC distribution metrics (ACR, ACDIF, and ACDEN) and prevalence metrics (FP and RP). Since the number of AoC is considerably lower than the number of LoC, we use the 10^{-3} scale to represent the ACR metric.

After computing all metrics for each project, we calculated means and medians when applicable and divided between periods "Before CI/CD" and "After CI/CD" since we wanted to compare the possible impact of CI/CD on the distribution and prevalence of atoms considering the release history. We

[5]https://www.travis-ci.com/

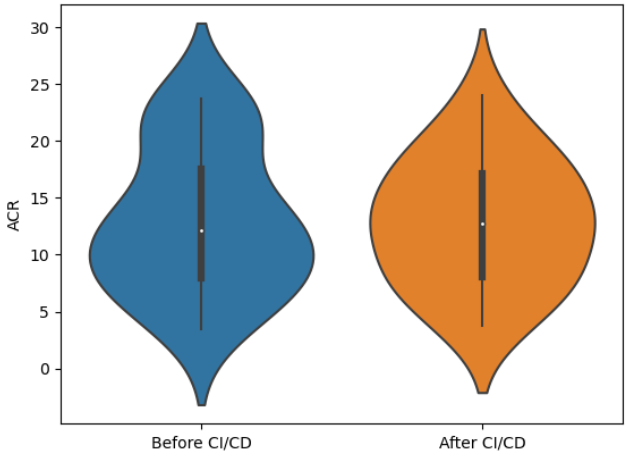

Figure 1. The violin plot of ACR before and after CI/CD adoption.

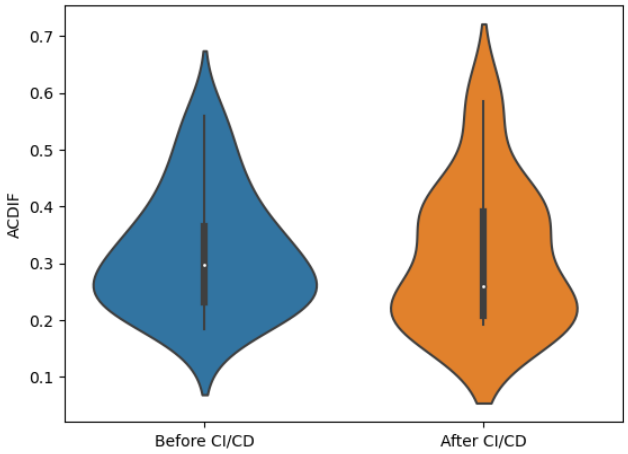

Figure 2. The violin plot of ACDIF before and after CI/CD adoption.

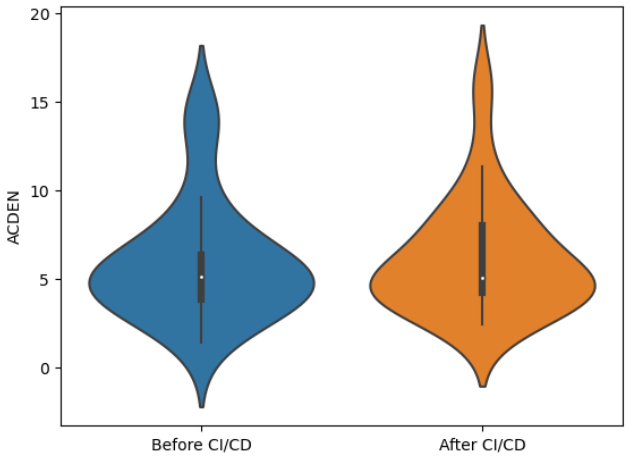

Figure 3. The violin plot of ACDEN before and after CI/CD adoption.

also made this separation when visualizing the data for each project.

Now, for the ACR, ACDIF and ACDEN, we compared the general data now divided as "Before CI/CD" and "After CI/CD" using the Wilcoxon Signed-Rank Test, a non-parametric test that does not assume normality in the data. The Wilcoxon test checks the hypothesis of having a statistically significant relation between the two sets.

For this test, the result is a p-value which must be 0.05 or lower for the null hypothesis, that there is no relation between data, to be rejected and the relation to statistically relevant. Using this test on each project's individual metrics could generate unreliable results because of a lack of data. As such, we only used this test with the means and medians

The following experiment for the distribution metrics focused on the projects' release history and the metrics' increase (or decrease) between each release, as percentages. These differences between releases were then used to calculate their geometric mean, again divided between "Before CI/CD" and "After CI/CD". In this situation, the geometric mean represents a tendency to increase or decrease and, as such, it is interesting to observe if there is a difference in tendency between the two periods.

The comparison of the FP and RP metrics was done by observation since our objective with these is to check for any differences between the prevalences, that is if certain types of atoms started appearing more or less frequently, by how much, and which ones are the most and least frequent. The Wilcoxon Signed-Rank Test can also be used in theory, but it is not ideal since it lacks the necessary data to generate robust results. Thus, we decided not to use it.

Finally, we also analyzed three projects more closely: Struts, GestureViews, and ColorPickerView. These three were selected because some or all of their distribution metrics increased after CI/CD, which we consider counter-intuitive. We also wanted to pick at least one Android project and one long-lived library. For this analysis, we checked the data we already had on each of these projects, and also their repositories on GitHub, to possibly identify a pattern possibly related to the metrics' increase.

4 Study Results and Discussion

4.1 Research Question 1 Answer

To answer our first research question, we started by computing the mean, median, and geometric mean of the differences between releases, for the ACR, ACDIF and ACDEN metrics, for each studied project and summarized them in Tables 3, 4, 5, 6, 7 and 8. We also created violin plot visualizations for the mean results of each metric as shown in Figures 1, 2, and 3. Figures 4, 5, and 6 are the same, except the plots are specific for each project, instead of a mean that represents all of them. showing the history of each metric were also created as shown in Figures 7, 8, and 9. Next, we employ the Wilcoxon Signed-Rank Test to compare the data before and after CI/CD adoption. Thus, one can verify whether there are statistically significant differences between them.

The Wilcoxon Signed-Rank Test is a non-parametric test to compare paired data samples. First, for the ACR's mean comparison, we defined the null and alternative hypotheses as follow: $H_0^{\overline{x}} : \overline{x}(\texttt{ACR}_\texttt{b}) = \overline{x}(\texttt{ACR}_\texttt{a})$ (it means that there is no statistical difference between ACR before and after CI/CD adoption), $H_1^{\overline{x}} : \overline{x}(\texttt{ACR}_\texttt{b}) > \overline{x}(\texttt{ACR}_\texttt{a})$ (it means that ACR before is significantly higher than ACR after CD adoption), and $H_2^{\overline{x}} : \overline{x}(\texttt{ACR}_\texttt{b}) < \overline{x}(\texttt{ACR}_\texttt{a})$ (it means that ACR before is significantly lower than ACR after CD adoption). Next, for the ACR's median comparison, we defined the null and alternative hypotheses as follows: $H_0^{\widetilde{x}} : \widetilde{x}(\texttt{ACR}_\texttt{b}) = \widetilde{x}(\texttt{ACR}_\texttt{a})$,

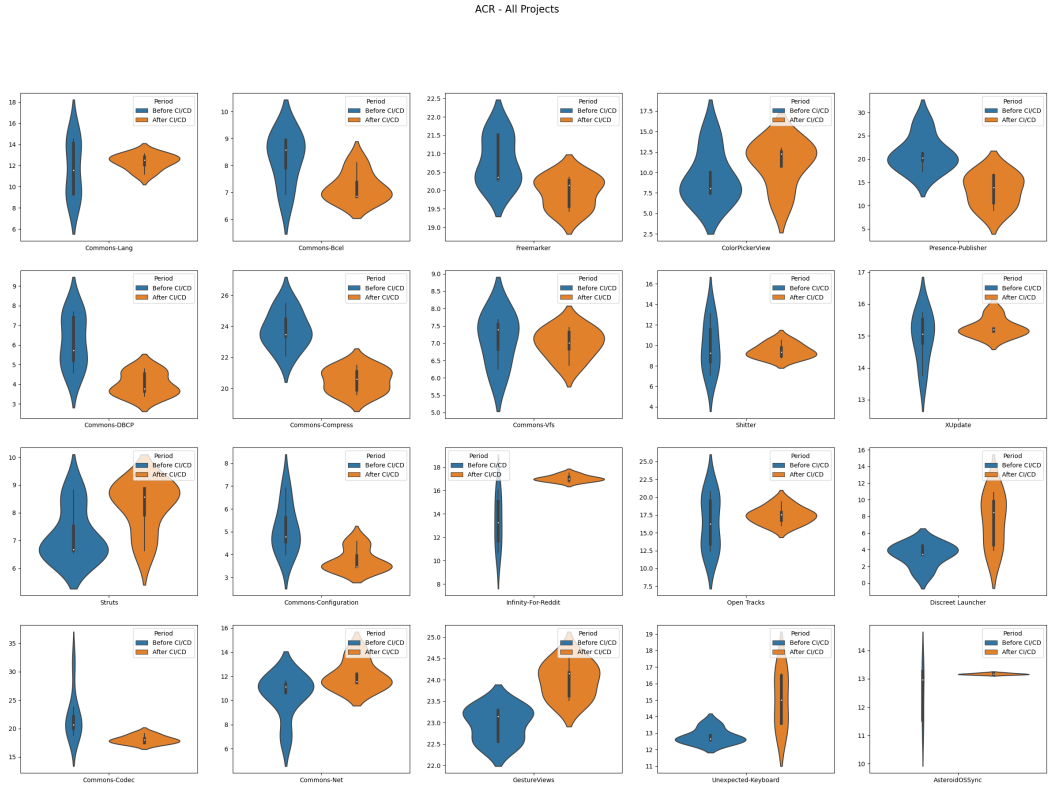

Figure 4. The violin plot of ACR before and after CI/CD adoption for each individual project

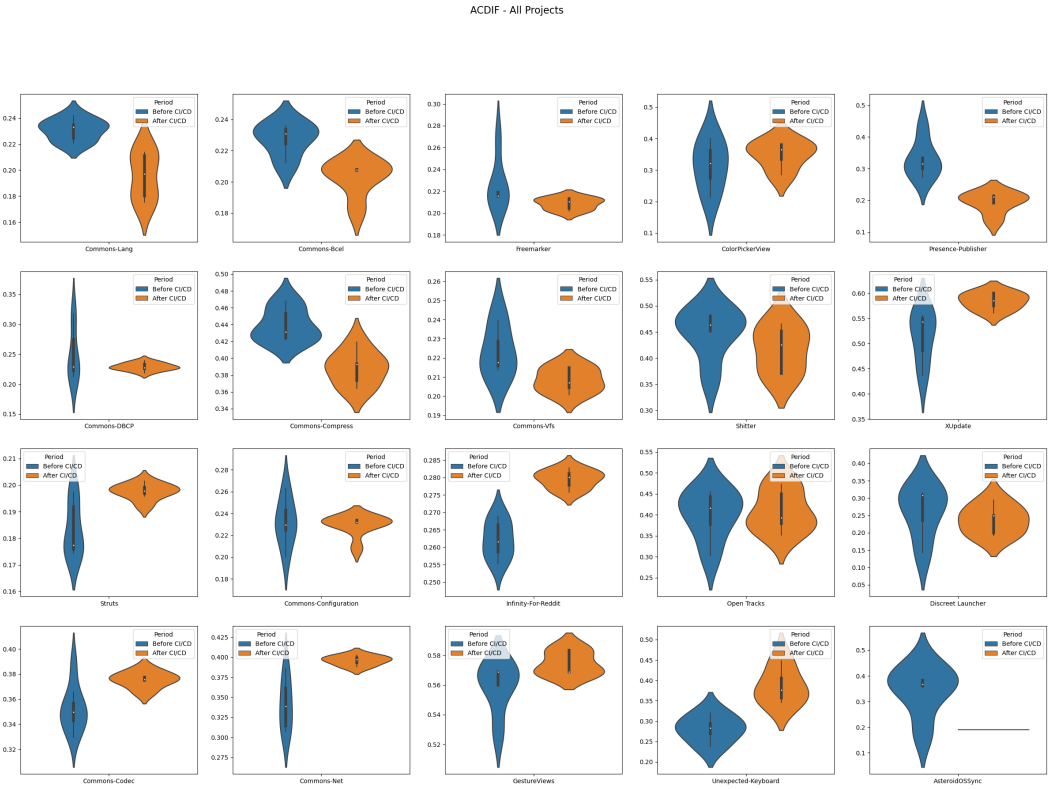

Figure 5. The violin plot of ACDIF before and after CD adoption for each individual project

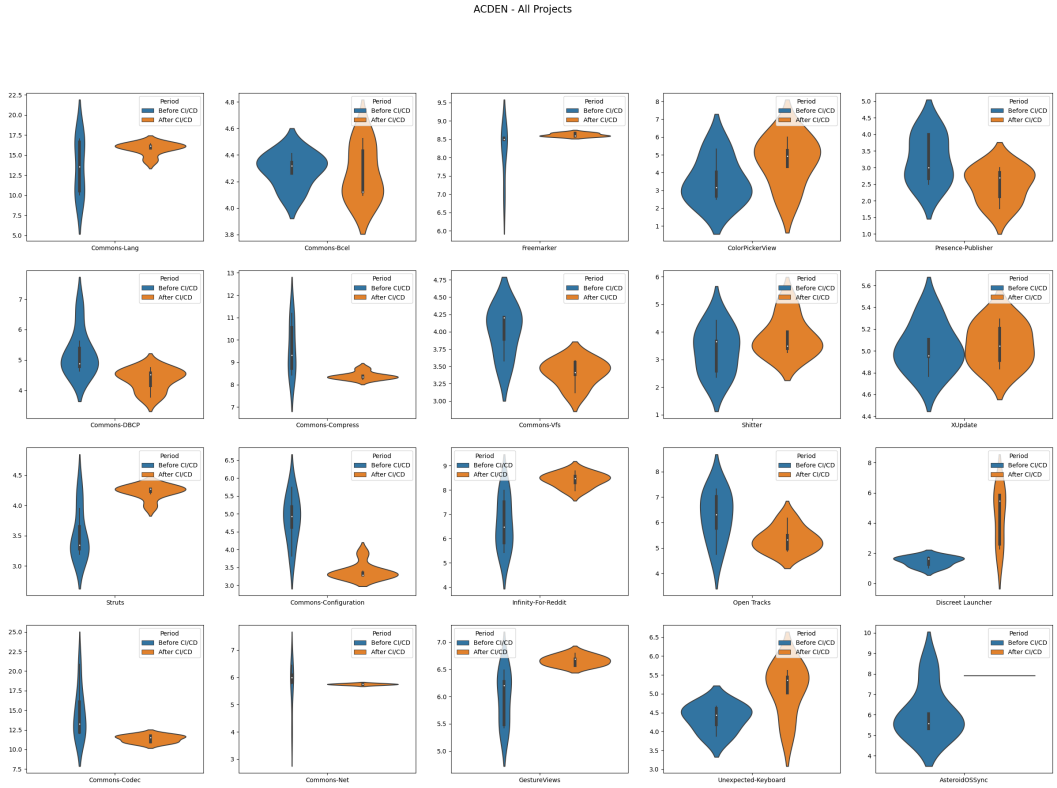

Figure 6. The violin plot of ACDEN before and after CD adoption for each individual project

Table 3. Summary of projects and AoC rate statistics. $\texttt{ACR}_\texttt{b}$ (ACR before CI/CD adoption), and $\texttt{ACR}_\texttt{a}$ (ACR after CI/CD adoption). The $\overline{x}$ and $\widetilde{x}$ stand for statistical mean and median respectively.

Project	$\overline{x}(\texttt{ACR}_\texttt{b})$	$\overline{x}(\texttt{ACR}_\texttt{a})$	$\widetilde{x}(\texttt{ACR}_\texttt{b})$	$\widetilde{x}(\texttt{ACR}_\texttt{a})$
commons-lang	11.73	12.36	11.54	12.52
commons-dbcp	6.17	4.00	5.72	3.77
struts	7.19	8.23	6.68	8.57
commons-codec	22.12	18.07	20.69	18.02
commons-bcel	8.26	7.20	8.58	6.85
commons-compress	23.77	20.53	23.47	20.61
commons-configuration	5.14	3.76	4.79	3.47
commons-net	10.47	12.09	11.14	11.56
freemarker	20.79	19.96	20.36	20.15
commons-vfs	7.11	7.00	7.40	7.00
infinity-for-reddit	13.34	17.04	13.27	16.99
discreet-launcher	3.42	7.53	3.47	8.49
opentracks	16.44	17.48	16.25	17.53
xupdate	14.96	15.26	15.06	15.20
presencepublisher	21.17	13.32	20.25	13.86
asteroidossync	12.47	13.17	12.96	13.17
unexpected-keyboard	12.82	15.04	12.64	15.01
shitter	9.88	9.46	9.23	9.32
colorpickerview	9.39	11.09	8.09	12.24
gestureviews	22.97	23.99	23.15	24.15

Note: $\overline{x}(\texttt{ACR}_\texttt{a})$, $\overline{x}(\texttt{ACR}_\texttt{b})$, $\widetilde{x}(\texttt{ACR}_\texttt{a})$, $\widetilde{x}(\texttt{ACR}_\texttt{b})$, $\hat{x}(\texttt{ACR}_\texttt{b})$, and $\hat{x}(\texttt{ACR}_\texttt{a})$ values are given in 10^{-3} scale.

Table 4. Summary of projects and AoC rate growth statistics. $\texttt{ACR}_\texttt{b}$ (ACR before CI/CD adoption), and $\texttt{ACR}_\texttt{a}$ (ACR after CI/CD adoption). The $\hat{x}$ stands for geometric mean.

Project	$\hat{x}(\texttt{ACR}_\texttt{b})$	$\hat{x}(\texttt{ACR}_\texttt{a})$
commons-lang	7.76	−3.29
commons-dbcp	−7.11	−4.38
struts	5.12	−4.02
commons-codec	−7.15	−0.55
commons-bcel	−8.19	1.33
commons-compress	−1.13	−1.97
commons-configuration	2.77	0.04
commons-net	10.22	5.09
freemarker	−1.36	−0.79
commons-vfs	−8.06	0.36
infinity-for-reddit	8.56	1.15
discreet-launcher	28.35	26.39
opentracks	7.37	−2.91
xupdate	0.50	0.18
presencepublisher	−3.81	−12.43
asteroidossync	−0.01	−0.12
unexpected-keyboard	−0.08	1.81
shitter	16.66	−5.70
colorpickerview	−18.75	13.27
gestureviews	0.81	1.04

Note: $\hat{x}(\texttt{ACR}_\texttt{b})$, and $\hat{x}(\texttt{ACR}_\texttt{a})$ values are all percentages. A positive value means growth, while a negative means decline.

Table 5. Summary of projects and AoC diffusion statistics. $\texttt{ACDIF}_\texttt{b}$ (ACDIF before CI/CD adoption), and $\texttt{ACDIF}_\texttt{a}$ (ACDIF after CI/CD adoption). The $\overline{x}$ and $\widetilde{x}$ stand for statistical mean and median, respectively.

Project	$\overline{x}(\texttt{ACDIF}_\texttt{b})$	$\overline{x}(\texttt{ACDIF}_\texttt{a})$	$\widetilde{x}(\texttt{ACDIF}_\texttt{b})$	$\widetilde{x}(\texttt{ACDIF}_\texttt{a})$
commons-lang	23.10	19.56	23.31	19.70
commons-dbcp	25.03	22.90	22.88	22.76
struts	18.35	19.76	17.74	19.78
commons-codec	35.24	37.55	34.96	37.58
commons-bcel	22.74	20.31	23.09	20.76
commons-compress	43.92	38.82	43.14	39.29
commons-configuration	23.21	22.94	22.94	23.19
commons-net	34.09	39.60	33.85	39.68
freemarker	22.67	20.86	21.59	21.05
commons-vfs	22.37	20.85	21.77	20.70
infinity-for-reddit	26.22	27.95	26.17	28.01
discreet-launcher	26.19	23.73	30.77	25.00
opentracks	40.12	40.88	41.61	39.30
xupdate	51.43	58.43	54.29	58.54
presencepublisher	33.01	19.42	31.58	21.14
asteroidossync	34.43	19.09	36.66	19.09
unexpected-keyboard	28.10	38.67	28.29	37.62
shitter	44.88	41.64	46.34	42.52
colorpickerview	31.46	34.94	32.21	36.55
gestureviews	55.91	57.45	56.86	56.86

Note: $\overline{x}(\texttt{ACDIF}_\texttt{a})$, $\overline{x}(\texttt{ACDIF}_\texttt{b})$, $\widetilde{x}(\texttt{ACDIF}_\texttt{a})$, and $\widetilde{x}(\texttt{ACDIF}_\texttt{b})$ values are all percentages.

Table 6. Summary of projects and AoC diffusion growth statistics. $\texttt{ACR}_\texttt{b}$ (ACDIF before CI/CD adoption), and $\texttt{ACR}_\texttt{a}$ (ACDIF after CI/CD adoption). The $\hat{x}$ stands for geometric mean.

Project	$\hat{x}(\texttt{ACDIF}_\texttt{b})$	$\hat{x}(\texttt{ACDIF}_\texttt{a})$
commons-lang	0.11	−3.07
commons-dbcp	0.32	0.33
struts	2.09	0.03
commons-codec	−1.73	1.77
commons-bcel	−2.46	−2.95
commons-compress	−1.35	−3.01
commons-configuration	2.60	−1.24
commons-net	4.69	1.03
freemarker	−5.35	−1.33
commons-vfs	0.92	−1.25
infinity-for-reddit	1.05	0.75
discreet-launcher	13.29	4.56
opentracks	1.20	−3.22
xupdate	5.89	1.76
presencepublisher	−4.89	−4.52
asteroidossync	−19.38	0.98
unexpected-keyboard	10.36	3.06
shitter	0.65	−1.71
colorpickerview	23.13	−1.21
gestureviews	1.80	0.00

Note: $\hat{x}(\texttt{ACDIF}_\texttt{b})$, and $\hat{x}(\texttt{ACDIF}_\texttt{a})$ values are all percentages. A positive value means growth, while a negative means decline.

$H_1^{\widetilde{x}} : \widetilde{x}(\texttt{ACR}_\texttt{b}) > \widetilde{x}(\texttt{ACR}_\texttt{a})$, and $H_2^{\widetilde{x}} : \widetilde{x}(\texttt{ACR}_\texttt{b}) < \widetilde{x}(\texttt{ACR}_\texttt{a})$. And finally, for the ACR's geometric mean comparison, we defined the null and alternative hypotheses as follows: $H_0^{\hat{x}} : \hat{x}(\texttt{ACR}_\texttt{b}) = \hat{x}(\texttt{ACR}_\texttt{a})$, $H_1^{\hat{x}} : \hat{x}(\texttt{ACR}_\texttt{b}) > \hat{x}(\texttt{ACR}_\texttt{a})$, and $H_2^{\hat{x}} : \hat{x}(\texttt{ACR}_\texttt{b}) < \hat{x}(\texttt{ACR}_\texttt{a})$. The hypotheses H_0, H_1, and H_2 are equivalent for $\overline{x}$, $\widetilde{x}$, and $\hat{x}$. This data for the ACR is present in Table 9. As seen on Tables 10 and 11, we defined the same hypotheses for the ACDIF and ACDEN, following the same nomenclature.

Tables 9, 10 and 11 summarize the statistical test results for all three metrics. After applying the statistical test to compare the metrics before and after CI/CD for the arithmetic mean and the median, we were able to identify that in all the cases of $H_0^{\overline{x}}$ and $H_0^{\widetilde{x}}$ the null hypotheses could not be rejected (i.e., none of the statistical test results have a significance level of $\alpha < 0.05$). Thus, the alternative hypotheses $H_1^{\overline{x}}$, $H_1^{\widetilde{x}}$, $H_2^{\overline{x}}$, and $H_2^{\widetilde{x}}$ are all rejected. For the geometric mean, however, $H_0^{\hat{x}}$ was rejected for ACDIF, and $H_1^{\hat{x}}$ could not be rejected, which means there was a significant decrease in the growth tendency of ACDIF after the implementation of CI/CD. Therefore, the statistical results indicate that the adoption of CI/CD has no significant impact on the AoC rate or density for the analyzed group of projects, but there was a significant impact on the growth of the AoC diffusion.

Table 7. Summary of projects and AoC density statistics. $\texttt{ACDEN}_\texttt{b}$ (ACDEN before CI/CD adoption), and $\texttt{ACDEN}_\texttt{a}$ (ACDEN after CI/CD adoption). The $\overline{x}$ and $\widetilde{x}$ stand for statistical mean and median, respectively.

Project	$\overline{x}(\texttt{ACDEN}_\texttt{b})$	$\overline{x}(\texttt{ACDEN}_\texttt{a})$	$\widetilde{x}(\texttt{ACDEN}_\texttt{b})$	$\widetilde{x}(\texttt{ACDEN}_\texttt{a})$
commons-lang	13.56	15.83	13.54	16.12
commons-dbcp	5.23	4.36	4.88	4.52
struts	3.52	4.23	3.34	4.27
commons-codec	14.55	11.38	13.29	11.50
commons-bcel	4.29	4.26	4.32	4.12
commons-compress	9.62	8.39	9.32	8.35
commons-configuration	4.87	3.39	4.93	3.29
commons-net	5.71	5.74	5.98	5.74
freemarker	8.20	8.62	8.50	8.59
commons-vfs	4.00	3.41	4.20	3.41
infinity-for-reddit	6.63	8.42	6.48	8.47
discreet-launcher	1.46	4.42	1.63	5.47
opentracks	6.25	5.34	6.30	5.33
xupdate	5.03	5.06	4.95	5.04
presencepublisher	3.23	2.48	3.00	2.69
asteroidossync	6.10	7.90	5.57	7.90
unexpected-keyboard	4.36	5.11	4.44	5.36
shitter	3.34	3.83	3.66	3.50
colorpickerview	3.54	3.65	3.16	4.94
gestureviews	5.98	6.66	6.21	6.69

4.2 Research Question 2 Answer

To answer our second research question, we calculated the metrics FP and RP for each type of atom studied. The metrics are shown in Table 12. We then analyzed the data to identify objects of interest, such as the most and least prevalent types

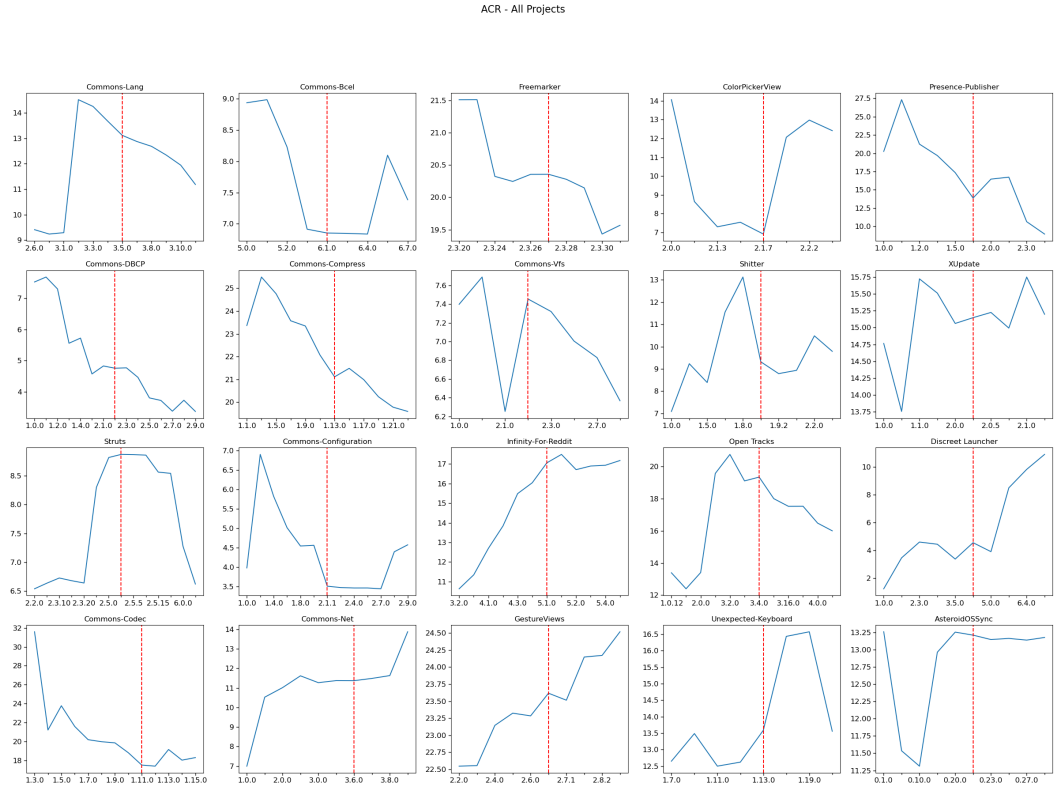

Figure 7. The plot of ACR's history for each individual project

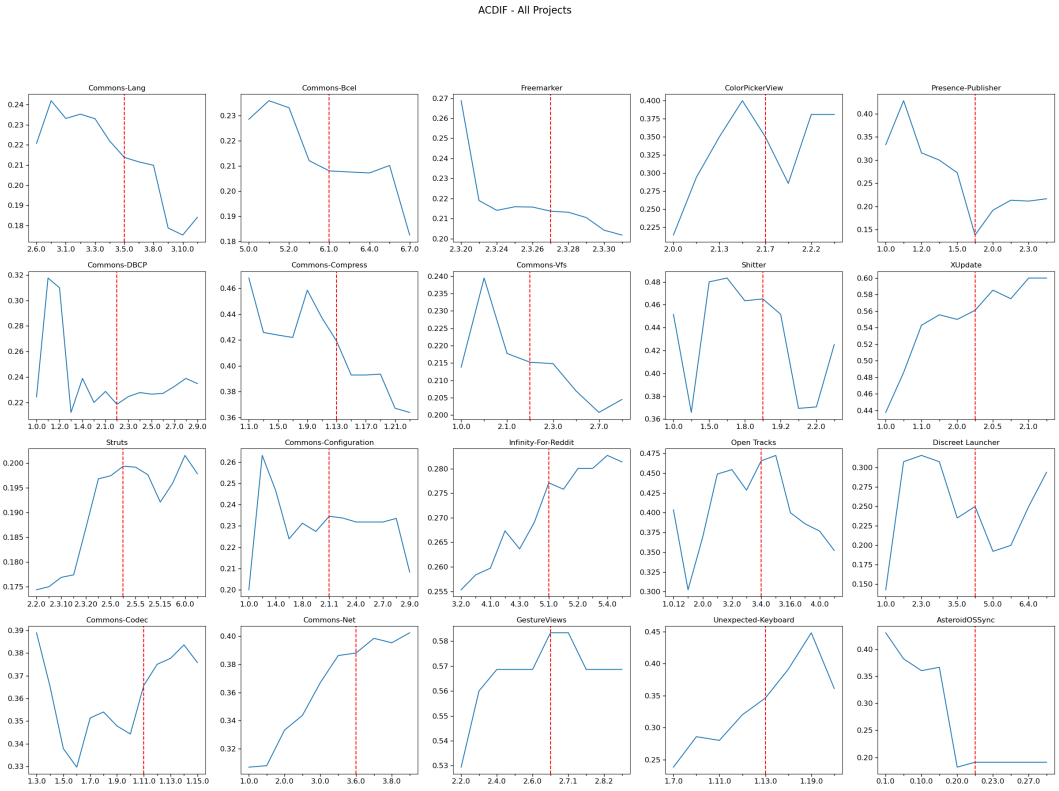

Figure 8. The plot of ACDIF's history for each individual project

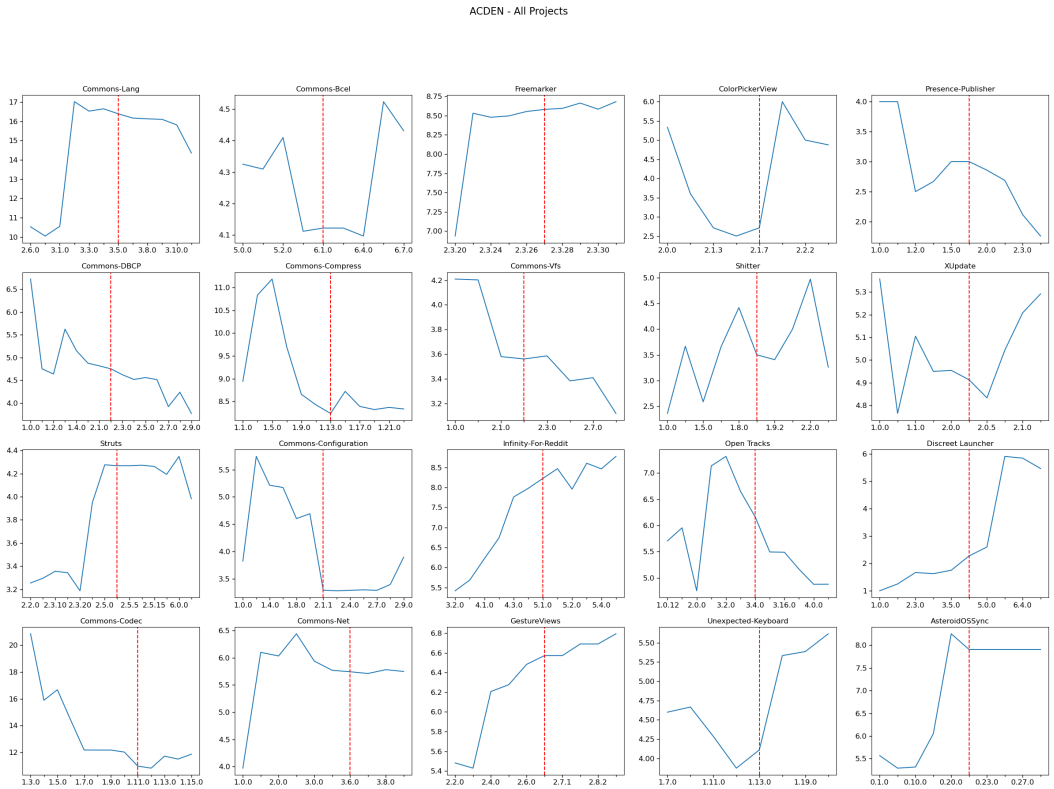

Figure 9. The plot of ACDEN's history for each individual project

Table 8. Summary of projects and AoC density growth statistics. $\texttt{ACR}_\texttt{b}$ (ACDEN before CI/CD adoption), and $\texttt{ACR}_\texttt{a}$ (ACDEN after CI/CD adoption). The $\hat{x}$ stands for geometric mean.

Project	$\hat{x}(\texttt{ACDEN}_\texttt{b})$	$\hat{x}(\texttt{ACDEN}_\texttt{a})$
commons-lang	9.60	−2.44
commons-dbcp	−5.42	−3.01
struts	4.65	−1.01
commons-codec	−7.57	−0.27
commons-bcel	−1.67	1.51
commons-compress	−1.18	−0.18
commons-configuration	4.17	−2.62
commons-net	7.75	−0.09
freemarker	5.38	0.30
commons-vfs	−7.78	−2.72
infinity-for-reddit	8.06	1.60
discreet-launcher	15.02	25.58
opentracks	3.11	−5.03
xupdate	−1.93	1.33
presencepublisher	−6.94	−10.13
asteroidossync	10.33	−0.85
unexpected-keyboard	−5.56	9.72
shitter	17.03	−5.92
colorpickerview	−22.32	18.17
gestureviews	4.28	0.94

Note: $\hat{x}(\texttt{ACDEN}_\texttt{b})$, and $\hat{x}(\texttt{ACDEN}_\texttt{a})$ values are all percentages. A positive value means growth, while a negative means decline.

Table 9. The Wilcoxon hypotheses statement and the test results. The symbols ✓ and ✗ indicate the result of the null hypothesis test (✓ fail to reject, and ✗ reject).

Wilcoxon Hypothesis	Value of p-value
$H_0^{\overline{x}} : \overline{x}(\texttt{ACR}_\texttt{b}) = \overline{x}(\texttt{ACR}_\texttt{a})$ (✓) $H_1^{\overline{x}} : \overline{x}(\texttt{ACR}_\texttt{b}) > \overline{x}(\texttt{ACR}_\texttt{a})$ (✗) $H_2^{\overline{x}} : \overline{x}(\texttt{ACR}_\texttt{b}) < \overline{x}(\texttt{ACR}_\texttt{a})$ (✗)	0.84
$H_0^{\widetilde{x}} : \widetilde{x}(\texttt{ACR}_\texttt{b}) = \widetilde{x}(\texttt{ACR}_\texttt{a})$ (✓) $H_1^{\widetilde{x}} : \widetilde{x}(\texttt{ACR}_\texttt{b}) > \widetilde{x}(\texttt{ACR}_\texttt{a})$ (✗) $H_2^{\widetilde{x}} : \widetilde{x}(\texttt{ACR}_\texttt{b}) < \widetilde{x}(\texttt{ACR}_\texttt{a})$ (✗)	0.70
$H_0^{\hat{x}} : \hat{x}(\texttt{ACR}_\texttt{b}) = \hat{x}(\texttt{ACR}_\texttt{a})$ (✓) $H_1^{\hat{x}} : \hat{x}(\texttt{ACR}_\texttt{b}) > \hat{x}(\texttt{ACR}_\texttt{a})$ (✗) $H_2^{\hat{x}} : \hat{x}(\texttt{ACR}_\texttt{b}) < \hat{x}(\texttt{ACR}_\texttt{a})$ (✗)	0.37

of atoms considering both metrics and if atoms tend to be more or less prevalent after the implementation of CI/CD.

First of all, it is not ideal to compare the prevalence before and after CI/CD for the FP and RP using the Wilcoxon Test. Table 12 shows that the OCB atom type never occurred in the analyzed projects. As such, its percentage is 0 before and after CI/CD for both metrics. `Pre-Increment/Decrement`, while not 0, did not change from before to after CI/CD for the FP. For the purposes of the Wilcoxon test, tied data is ignored, leaving us with only 8 samples before and after for the FP, and only 9 samples before and after for the RP. While it is still possible to use the Wilcoxon test, the results will not be as relevant as when used properly. As such, we decided

Table 10. The Wilcoxon hypotheses statement and the test results. The symbols ✓ and ✗ indicate the result of the null hypothesis test (✓ fail to reject, and ✗ reject).

Wilcoxon Hypothesis	Value of p-value
$H_0^{\overline{x}} : \overline{x}(\texttt{ACDIF}_\texttt{b}) = \overline{x}(\texttt{ACDIF}_\texttt{a})$ (✓) $H_1^{\overline{x}} : \overline{x}(\texttt{ACDIF}_\texttt{b}) > \overline{x}(\texttt{ACDIF}_\texttt{a})$ (✗) $H_2^{\overline{x}} : \overline{x}(\texttt{ACDIF}_\texttt{b}) < \overline{x}(\texttt{ACDIF}_\texttt{a})$ (✗)	0.57
$H_0^{\widetilde{x}} : \widetilde{x}(\texttt{ACDIF}_\texttt{b}) = \widetilde{x}(\texttt{ACDIF}_\texttt{a})$ (✓) $H_1^{\widetilde{x}} : \widetilde{x}(\texttt{ACDIF}_\texttt{b}) > \widetilde{x}(\texttt{ACDIF}_\texttt{a})$ (✗) $H_2^{\widetilde{x}} : \widetilde{x}(\texttt{ACDIF}_\texttt{b}) < \widetilde{x}(\texttt{ACDIF}_\texttt{a})$ (✗)	0.60
$H_0^{\hat{x}} : \hat{x}(\texttt{ACDIF}_\texttt{b}) = \hat{x}(\texttt{ACDIF}_\texttt{a})$ (✗) $H_1^{\hat{x}} : \hat{x}(\texttt{ACDIF}_\texttt{b}) > \hat{x}(\texttt{ACDIF}_\texttt{a})$ (✓) $H_2^{\hat{x}} : \hat{x}(\texttt{ACDIF}_\texttt{b}) < \hat{x}(\texttt{ACDIF}_\texttt{a})$ (✗)	0.03

Table 11. The Wilcoxon hypotheses statement and the test results. The symbols ✓ and ✗ indicate the result of the null hypothesis test (✓ fail to reject, and ✗ reject).

Wilcoxon Hypothesis	Value of p-value
$H_0^{\overline{x}} : \overline{x}(\texttt{ACDEN}_\texttt{b}) = \overline{x}(\texttt{ACDEN}_\texttt{a})$ (✓) $H_1^{\overline{x}} : \overline{x}(\texttt{ACDEN}_\texttt{b}) > \overline{x}(\texttt{ACDEN}_\texttt{a})$ (✗) $H_2^{\overline{x}} : \overline{x}(\texttt{ACDEN}_\texttt{b}) < \overline{x}(\texttt{ACDEN}_\texttt{a})$ (✗)	0.55
$H_0^{\widetilde{x}} : \widetilde{x}(\texttt{ACDEN}_\texttt{b}) = \widetilde{x}(\texttt{ACDEN}_\texttt{a})$ (✓) $H_1^{\widetilde{x}} : \widetilde{x}(\texttt{ACDEN}_\texttt{b}) > \widetilde{x}(\texttt{ACDEN}_\texttt{a})$ (✗) $H_2^{\widetilde{x}} : \widetilde{x}(\texttt{ACDEN}_\texttt{b}) < \widetilde{x}(\texttt{ACDEN}_\texttt{a})$ (✗)	0.57
$H_0^{\hat{x}} : \hat{x}(\texttt{ACDEN}_\texttt{b}) = \hat{x}(\texttt{ACDEN}_\texttt{a})$ (✓) $H_1^{\hat{x}} : \hat{x}(\texttt{ACDEN}_\texttt{b}) > \hat{x}(\texttt{ACDEN}_\texttt{a})$ (✗) $H_2^{\hat{x}} : \hat{x}(\texttt{ACDEN}_\texttt{b}) < \hat{x}(\texttt{ACDEN}_\texttt{a})$ (✗)	0.43

Table 12. Summary of projects and their frequency and relative percentages. $\texttt{FP}_\texttt{b}$ (FP before CI/CD adoption), and $\texttt{FP}_\texttt{a}$ (FP after CI/CD adoption). The applies to the FP

AoC Acronym	($\texttt{FP}_\texttt{b}$)	($\texttt{FP}_\texttt{a}$)	($\texttt{RP}_\texttt{b}$)	($\texttt{RP}_\texttt{a}$)
Post-Inc/Dec	75.93	76.85	4.36	3.71
Pre-Inc/Dec	59.26	59.26	1.30	0.91
IOP	86.11	99.07	8.29	10.51
CO	90.74	98.15	22.22	25.40
AaL	1.85	7.41	0.02	0.05
LaCF	96.30	99.07	47.91	44.82
RV	12.96	15.74	0.11	0.07
CoLE	26.85	36.11	0.26	0.24
OCB	0.00	0.00	0.00	0.00
TC	89.81	97.22	15.54	14.27

Note: ($\texttt{FP}_\texttt{a}$), ($\texttt{FP}_\texttt{b}$), ($\texttt{RP}_\texttt{a}$), and ($\texttt{RP}_\texttt{b}$) values are all percentages.

to not use it for this analysis. Instead, we manually checked the data before and after CI/CD for both metrics.

The FP is important because it can show us if the types of atoms are appearing or disappearing from the projects after CI/CD. By checking the data manually, it is possible to see an increase in the FP for all types of atoms, except for the `Omitted Curly Braces` and `Pre-Increment/Decrement`, which means almost all types of atoms are appearing more.

The RP works as an alternative way to check for the prevalence of atoms. As the name suggests, it's prevalence relative to other atoms. This means that when the percentage of an atom increases, the percentage of another must decrease since the sum of all RP always results in 100%. The purpose of RP is to be manually analyzed and compared to the FP to avoid having just one way to measure specific prevalence and determine the most and least prevalent types of atoms.

For both FP and RP, `Omitted Curly Braces` is 0, and, in absolute terms, it is the least prevalent type of atom, but we will disregard it from now on since it is not even present on any projects nor periods. `Logic as Control Flow` is the most prevalent atom type according to both FP and RP, before and after CI/CD, although, in the case of FP, after CI/CD it ties with the `Infix Operator Precedence`, but since it is more prevalent before CI/CD, it wins the tie. `Arithmetic as Logic` is the least prevalent atom type (that is present) according to both FP and RP, before and after CI/CD, even though it got a considerable increase after CI/CD when considering FP.

4.3 Discussion

Individually, some projects, like commons-dbcp, actually had their ACR, ACDIF and ACDEN reduced after the implementation of CI/CD, going against the tendency of increase, but the behavior is not consistent throughout the projects no matter the metric as we can see on the project-specific plots shown in Figures 4, 5, 6, 7, 8 and 9 with some even having their rates increased. The magnitude of the changes is also inconsistent. This inconsistency is still true, to a lesser extent, when checking the growth rates of the metrics. Visualizing the data together, we can see that the impact, when considering all projects, was practically none. The ACR changed for all projects, as mentioned before and shown in Table 3, and the diffusion and density changed for most projects, as seen in Tables 5 and 7. These changes can also be visualized in the previously mentioned project-specific plots. However, when analyzing the general violin plots in Figures 1, 2 and 3 of both before and after CI/CD, we see the difference in distribution between them is hard to visualize, as they are both very similar. In the case of the density, a tendency of increase is actually noticeable, but as mentioned before, the relation between it and CI/CD was rejected, and most probably is just a result of the great natural growth of the number of AoC. The violin plots of Figures 1, 2 and 3 consider the means of the metrics, but the violin plots for the medians are virtually the same. The growth rates for the metrics seem to have a general tendency to decrease, but the only significant tendency seems to be the diffusion growth rate.

The FP results made sense when thinking about atoms in general, that is, their natural tendency is to grow fast and appear more. So even considering the general prevalence increase after the implementation of CI/CD, it is probably not related to CI/CD, but just to the passage of time.

The RP, in theory, could conflict with the FP results. Hypothetically, an atom type that appears on every project and in every release version but only once would have an FP of 100%, but its RP would be quite low when compared to other atoms. The actual results did not show that, most likely because of the AoC tendency to grow in num-

bers. While new atom types appeared, the ones that were already present grew in numbers. Because of that, we did not have much difficulty determining the most and least prevalent of the atom types. We also tried checking for patterns on the shifting RP values between before and after CI/CD. While the most prevalent, `Logic as Control Flow`, got even more frequent, it lost prevalence when considering the RP, while the `Infix Operator Precedence` grew in both, probably because the growth of its frequency was much higher. `Pre-Increment/Decrement` actually got less relatively prevalent because it maintained the same frequency. That said, the `Type Conversion` atom type got a substantial increase in FP that did not reflect on RP.

We then verified our detailed analysis of the three projects selected as mentioned by the end of Section 3.7. Project size may influence atom behavior, but all three projects had similar behavior, even with Struts being much larger than the other two considering both classes and lines of code. The most prevalent atom types had some variation, but we confirmed `Logic as Control Flow` as the most prevalent in Struts and GestureViews. In the most recent analyzed version of ColorPickerView, it was second place, after `Infix Operator Precedence` and tied with `Type Conversion`. The two most prevalent atoms compose more than 60% of all atoms in these projects and even reach more than 90% in Struts. These analyses do not change significantly through versions, except for the most prevalent atoms in ColorPickerView, which changed frequently.

While analyzing their repositories, we focused on the projects' age, contributors, data of implementation of CI/CD, and any possible drastic changes that may have occurred that may have influenced our data. We knew from our data that Struts' versioning was inconsistent, but after analyzing its history, the versioning changes and problems do not seem to be related to AoC or CI/CD. The atom metrics generally increased before Struts' massive growth in version 2.5.0, for example. Struts is the oldest project of the three, starting on GitHub in 2006. Because of its age and size, it may be considered bad to change code for a less confusing version since these changes may cause a bigger impact than expected and may not be considered a priority. However, ColorPickerView and GestureViews are considerably smaller and younger, being created in 2014 and 2017 respectively, and they seem to behave similarly to Struts. This may happen because of the number of contributors, and which of them has only one that made significant contributions to their projects, which means there was probably little effort to make their code clearer to anyone else. They had more contributors, 11 and 4 in total respectively, but the others had very few contributions. Struts has the opposite situation, where it has quite a few important contributors, and constantly has the number of contributors increasing through the years, but it may not be enough to stimulate a possibly large change in the code to deal with confusion. However, in the last version we analyzed for Struts, there was a significant drop in the number of AoC and AoC Rate, so the culture may be changing, although it is not necessarily related to CI/CD, as it was implemented for over 7 years. The other two projects do not seem to be improving.

Finally, we checked one other project, this time that had positive results after CI/CD, for a quick comparison with the three projects. Compress is very similar to Struts in size and number of contributors while being quite different from the other two. It has considerably more atoms than Struts, even though they're almost the same size. However, we could not find an obvious reason why all of its metrics have a steady tendency to decrease. What we know is that this tendency does not start with CI/CD, as shown by its plots in Figures 7, 8, and 9.

4.4 Result Implications

Implications For Researchers. Our study and results can bring implications for future works. It is possible to expand on this analysis with different and more projects. Since the tools we used were made with Java in mind, we limited ourselves to Java projects, but other works may choose other programming languages or just different Java projects. The addition of more projects to the analysis could confirm that there is little to no impact on AoC distribution and prevalence caused by CI/CD, or that perhaps there is an impact for some projects. A comparative analysis could also be made with projects from different programming languages. More distinct analyses could also be made with different metrics and focus, or possibly to study more variables besides the implementation of CI/CD that may also influence the prevalence of AoC such as the projects' size, number of contributors, and how much they contribute.

Implications For Developers. According to our results, if developers are interested in reducing the prevalence of AoC in their code, common CI/CD practices and pipelines may not be enough since there was little to no impact on AoC prevalence because of CI/CD. As such, it is of interest to these developers to apply more specialized tools and solutions, such as BOHR, to locate and identify these atoms so that they can be replaced by better code in future refactorings.

5 Threats to Validity

We will use the threats to the validity of our study as presented by Wohlin et al. (2012). Those are threats to conclusion, construct, internal, and external validity.

Conclusion Validity. To deal with threats to our study's conclusion, we chose the Wilcoxon Signed-Rank Test so that our conclusion was statistically relevant, and that it didn't depend on the type of data we used. Also, to analyze the specific atom prevalences, we chose two metrics, FP and RP, that could result in different candidates for most and least prevalent atom types.

Internal Validity. It is possible that some other variables could interfere by increasing or decreasing the rate of AoC in ways we do not understand. However, it is not the intention of this paper to imply causal relationships.

Construct Validity. There is a possibility of human bias being present when the projects are chosen. To avoid this, the manual choice of projects was made at random after the filtering, being invalidated only if the project did not fit our criteria. Other parts of our analysis were automated to avoid mistakes and bias.

External Validity. The sample size of this study is only 20 projects, which can threaten its generalization. To mitigate it, we tried to diversify our projects with not only long-lived Java libraries but also projects from another context entirely, that is, Android projects. We also filtered the projects, getting only relevant ones, while avoiding personal projects.

6 Related Work

Gopstein et al. (2017) introduced the concept of Atoms of Confusion as the smallest piece of code that can cause confusion in developers, making them misunderstand what the code actually does, which can lead to mistakes during development and when doing tasks. This work also focused on AoC in the context of the C programming language, while other works studied the AoC in different programming language contexts, such as Castor (2018), with Swift; Langhout and Aniche (2021), with Java; Torres et al. (2023), with JavaScript; and Costa et al. (2023), with Python. Other works also study the relationship of AoC with different metrics, such as Bogachenkova et al. (2022) which analyzed the possible relation between AoC, code review, and pull requests.

Gopstein et al. (2018) also made a study to check the prevalence of AoC in the context of open-source C and C++ projects, while also studying the possible impact on the number of bugs, and number of bug fixes that contain atoms. Tahsin et al. (2023) made an empirical study of the Prevalence of Atoms of Confusion in Open Source Java Systems, also showing the most common atoms in this context. Mendes et al. (2022) similarly studied AoC and their prevalence in a specific context, open-source long-lived Java libraries, but also analyzed the co-occurrence of atoms, and created the tool for AoC detection we used in our study.

On CI/CD, Almeida et al. (2022) studied the relation of CI/CD with bug-fixing time, where it was found that there was a decrease in bug-fixing time after CI/CD implementation. Our work is a derivative of this paper, as we consider a different variable, within a different context but made our studying process in similar ways. Fairbanks et al. (2023) verified the impact of CI/CD on commit velocity and the number of reported issues, analyzing over $12,000$ repositories from GitHub, and GitLab, with roughly $4,500$ of them having CI/CD. Liu et al. (2023) analysis was focused on the context of Android apps, analyzing more than $80,000$ repositories from GitHub, GitLab, and Bitbucket, and finding the presence on CI/CD in roughly 10% of them. Their focus was to check the extent of CI/CD adoption, and the use of different CI/CD services in the projects.

7 Conclusion and Final Remarks

We analyzed 20 open-source Java projects, 10 long-lived libraries, and 10 Android projects, with the intention of checking if the implementation of CI/CD had an impact on the metrics `ACR`, `ACDIF` and `ACDEN`, which we used as proxies to measure the distribution of atoms of confusion, and the metrics `FP` and `RP` to measure the prevalence of specific types of atoms. We filtered and chose the repositories following specific criteria to get relevant projects for the analysis. Then, we made a static analysis of the project release versions to get the important data, such as LoC, Number of Classes, Number of AoC (total and for specific types) and Number of Classes with AoC, to calculate the metrics. We then made a comparison of the metrics before and after the implementation of CI/CD. Our results imply there is no statistically significant relationship between the implementation of CI/CD and the metrics `ACR`, and `ACDEN`, but there was a significant relationship when considering `ACDIF`'s growth rate specifically. Although these metrics changed for all projects individually, as a group they were very stable. Considering `FP`, there seems to be a relationship between the two time periods, but since it follows the common atoms of confusion patterns (to get more prevalent with project growth), the significance of CI/CD here is dubious. And finally, we found that, for both `FP` and `RP`, the most prevalent atom type is the `Logic as Control Flow` and the least prevalent atom type is the `Arithmetic as Logic`.

8 Data Availability

Our data and the code created for this paper are available as a GitHub repository with this link: https://github.com/dnfeijo/atoms-CICD-reproduction-pack.

References

Almeida, C. D. A. d., Feijó, D. N., and Rocha, L. S. (2022). Studying the impact of continuous delivery adoption on bug-fixing time in apache's open-source projects. In 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR), pages 132–136.

Bogachenkova, V., Nguyen, L., Ebert, F., Serebrenik, A., and Castor, F. (2022). Evaluating atoms of confusion in the context of code reviews. In 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 404–408.

Castor, F. (2018). Identifying confusing code in swift programs. In VI Workshop on Software Visualization, Evolution and Maintenance.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. IEEE Software, 32(2):50–54.

Costa, J. A. S. d., Gheyi, R., Castor, F., Oliveira, P. R. F. d., Ribeiro, M., and Fonseca, B. (2023). Seeing confusion through a new lens: on the impact of atoms of confusion on novices' code comprehension. Empirical Software Engineering, 28.

Fairbanks, J., Tharigonda, A., and Eisty, N. U. (2023). Analyzing the effects of ci/cd on open source repositories in github and gitlab. In 2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA).

Feijó, D., de Almeida, C., and Rocha, L. (2023). Studying the impact of continuous delivery adoption on atoms of confusion rate in open-source projects. In Anais do XI

Workshop de Visualização, Evolução e Manutenção de Software, pages 6–10, Porto Alegre, RS, Brasil. SBC.

Fowler, M. and Foemmel, M. (2006). Continuous integration. `https://web.archive.org/web/20061231051206/https://martinfowler.com/articles/continuousIntegration.html`.

Gopstein, D., Iannacone, J., Yan, Y., DeLong, L., Zhuang, Y., Yeh, M. K.-C., and Cappos, J. (2017). Understanding misunderstandings in source code. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, page 129–139, New York, NY, USA. Association for Computing Machinery.

Gopstein, D., Zhou, H. H., Frankl, P., and Cappos, J. (2018). Prevalence of confusing code in software projects: Atoms of confusion in the wild. In 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), pages 281–291.

Humble, J. (2017). Continuous delivery sounds great, but will it work here? it's not magic, it just requires continuous, daily improvement at all levels. Queue, 15(6):57–76.

Humble, J. and Farley, D. (2010). Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education.

Itkonen, J., Udd, R., Lassenius, C., and Lehtonen, T. (2016). Perceived benefits of adopting continuous delivery practices. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16, New York, NY, USA. Association for Computing Machinery.

Langhout, C. and Aniche, M. (2021). Atoms of confusion in java. In O'Conner, L., editor, 29th IEEE/ACM International Conference on Program Comprehension (ICPC 2021), IEEE International Conference on Program Comprehension, pages 25–35, United States. IEEE. Accepted author manuscript; 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC) ; Conference date: 20-05-2021 Through 21-05-2021.

Lenarduzzi, V., Taibi, D., Tosi, D., Lavazza, L., and Morasca, S. (2020). Open source software evaluation, selection, and adoption: a systematic literature review. In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 437–444.

Liu, P., Sun, X., Zhao, Y., Liu, Y., Grundy, J., and Li, L. (2023). A first look at ci/cd adoptions in open-source android apps. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22, New York, NY, USA. Association for Computing Machinery.

Mendes, W., Pinheiro, O., Santos, E., Rocha, L., and Viana, W. (2022). Dazed and confused: Studying the prevalence of atoms of confusion in long-lived java libraries. In 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 106–116.

Minelli, R., Mocci, A., and Lanza, M. (2015). I know what you did last summer - an investigation of how developers spend their time. In 2015 IEEE 23rd International Conference on Program Comprehension, pages 25–35.

Pawlak, R., Monperrus, M., Petitprez, N., Noguera, C., and Seinturier, L. (2016). Spoon: A library for implementing analyses and transformations of java source code. Software: Practice and Experience, 46(9):1155–1179.

Pinheiro, O., Rocha, L., and Viana, W. (2023). How they relate and leave: Understanding atoms of confusion in open-source java projects. In 2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 119–130.

Shahin, M., Ali Babar, M., and Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5:3909–3943.

Stahl, D., Martensson, T., and Bosch, J. (2017). Continuous practices and devops: beyond the buzz, what does it all mean? In 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 440–448.

Tahsin, N., Fuad, N., and Satter, A. (2023). Prevalence of 'atoms of confusion' in open source java systems: An empirical study. Journal of Software: Evolution and Process.

Torres, A., Oliveira, C., Okimoto, M., Marcilio, D., Queiroga, P., Castor, F., Bonifacio, R., Canedo, E., Ribeiro, M., and Monteiro, E. (2023). An investigation of confusing code patterns in javascript. Journal of Systems and Software, 203:111731.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). Experimentation in Software Engineering. Springer Publishing Company, Incorporated.

Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., and Li, S. (2018). Measuring program comprehension: A large-scale field study with professionals. IEEE Transactions on Software Engineering, 44(10):951–976.

Øyvind Hauge, Ayala, C., and Conradi, R. (2010). Adoption of open source software in software-intensive organizations – a systematic literature review. Information and Software Technology, 52(11):1133–1154. Special Section on Best Papers PROMISE 2009.