A Preliminary Study of GitHub Actions Workflow Changes

Pooya Rostami Mazrae[1], Alexandre Decan[1,3], Tom Mens[1] and Mairieli Wessel[2]

[1]University of Mons (UMONS), Mons, Belgium
[2]Radboud University, Nijmegen, The Netherlands
[3]F.R.S.-FNRS Research Associate

Abstract

CI/CD practices play a significant role in collaborative software development. The GitHub social coding platform introduced GitHub Actions as a way to automate different aspects of software production such as testing, building, quality checking, dependency and security management. We report on preliminary findings of a quantitative analysis on how GitHub Actions workflows are being changed over time. The study is based on a dataset of 22,733 GitHub repositories containing 4 million weekly snapshots of workflow files from November 2019 to September 2022. First, we analyse the coarse-grained changes being made to workflows, including when repositories start using them, when they are being added, modified, renamed and removed. Second, we analyse changes made to workflow code, including how many code lines are changed and what types of changes are being made to them. The findings of this quantitative analysis provide preliminary insights on how GitHub Actions workflows are being changed over time, and whether they adhere to the evolution laws of continuing growth and continuing change. It paves the way for studying more evolution laws, as well as more in-depth analyses on the types of changes that CI/CD workflows are subject to as well as the reasons for these changes.

Keywords

collaborative software development, workflow automation, software repository mining, continuous integration and deployment, GitHub, software changes

1. Introduction

Continuous Integration and Development (CI/CD) practices aim to help developers release high-quality software products more efficiently and with less effort [1]. The popularity of CI/CD has increased significantly in recent years, making it a crucial aspect of modern software development [2, 3]. Their widespread adoption can be attributed to the Extreme Programming methodology [4], which emphasizes automating various aspects of software production. Throughout the years, various CI/CD tools have been widely used (e.g., Jenkins, Travis, Azure DevOps, CircleCI, AppVeyor, and GitHub Actions) to automate a broad range of software development activities such as testing, building, quality checking, dependency and security management [5, 6, 7].

Focusing on the GitHub social coding platform in particular, which is the largest collaborative software development platform used by software developers, GitHub Actions (GHA) was introduced as a new CI/CD solution in November 2019. It has quickly gained popularity, replacing Travis as the dominant CI/CD tool in less than 18 months [8]. Decan et al. [9] reported that, out of 68K+ GitHub repositories provided in their dataset, 43.9% are utilizing GHA workflows by the end of January 2022. This indicates the widespread adoption of GHA within the GitHub community.

Workflow configuration files are the main components to configure GHA pipelines. Just like ordinary source code, they are developed and modified throughout the project's lifetime to meet the needs of developers. In the case of GHA, workflows are stored in a YAML format in the .github/workflows/ folder of the corresponding GitHub repository.

Knowing when, why and how developers modify workflow files can be helpful to improve CI/CD practices, to detect common patterns and mistakes developers do in their workflows, and to create tools to assist them in writing and maintaining workflows.

As preliminary steps towards such a comprehension, this article aims to characterise the changes made to these workflow files over their lifetime. To do so, we study two main research goals, based on an extracted dataset of 22,733 repositories accounting for 4,127,760 weekly snapshots of workflow files over a 34-month period from November 2019 to September 2022.

Goal $\mathbf{G_1}$ aims to quantify the **coarse-grained changes** being made to GHA workflows, and is broken down into three specific research questions:

$RQ_{1.1}$ **When do repositories start using GHA?** As the most primitive type of change, we investigate how long it takes for repositories to start using GHA.

$RQ_{1.2}$ **Which types of coarse-grained changes are**

SATToSE'23: Seminar on Advanced Techniques & Tools for Software Evolution, June 12–14, 2023, Salerno, Italy

pooya.rostamimazrae@umons.ac.be (P. R. Mazrae); alexandre.decan@umons.ac.be (A. Decan); tom.mens@umons.ac.be (T. Mens); mairieli.wessel@ru.nl (M. Wessel)
0000-0002-4859-1546 (P. R. Mazrae); 0000-0002-5824-5823 (A. Decan); 0000-0003-3636-5020 (T. Mens); 0000-0001-8619-726X (M. Wessel)

CEUR Workshop Proceedings (CEUR-WS.org)

workflows subject to? We analyse the four possible types of coarse-grained changes that can be made to workflows: adding, renaming or removing a workflow file, as well as modifying the contents of the workflow file.
$RQ_{1.3}$ **When do the different types of coarse-grained changes occur?** For all four types of coarse-grained workflow changes, we analyse when these changes occur.

Goal $\mathbf{G_2}$ focuses on quantifying the **fine-grained changes** to workflow files, by analysing the changes that are made in their contents. In this article, we restrict ourselves to studying **line-based changes** through the two following research questions:
$RQ_{2.1}$ **Which types of line-based changes are workflow files subject to?** We examine the addition, modification and removal of lines in workflow files.
$RQ_{2.2}$ **When do different types of line-based changes occur?** For the three above types of fine-grained workflow changes, we analyse when they occur over time.

2. Related Work

This section explores the existing literature on the evolution of CI/CD configuration files before the emergence of GitHub Actions (Section 2.1) and the use of GHA as a CI/CD tool (Section 2.2). To the best of our knowledge, the current article is the first large-scale study investigating the evolution of GHA workflow file changes.

2.1. Evolution of CI/CD configuration files

Even before introduction of GHA, usage of CI/CD tools in software projects was well established and most well known ones of them (especially Travis) have been using YAML based configuration files. In this section, we go through studies related to the evolution of the configuration file in those CI/CD tools.

Gallaba and McIntosh [10] studied the usage and misuse of features in Travis configuration files based on a dataset of 9,312 GitHub repositories. They found that *job processing nodes* were the most frequently modified, indicating that most of the usage of Travis was for continuous integration rather than continuous deployment. They also developed and evaluated a tool for identifying anti-patterns in Travis configuration files, and a second tool for automatical removal of those anti-patterns.

In a similar vein, Vassallo et al. [11] developed a tool to identify anti-patterns in Java projects relying on Travis. To determine the most critical anti-patterns in Travis log files, they relied on Duvall's work [2], which served as a reference for quality checking and identifying patterns and anti-patterns in continuous integration.

Durieux et al. [12] compiled a dataset of over 35M+ Travis jobs triggered by 272,917 projects. Their findings indicate that out of the 709K+ commits that changed a Travis configuration file, the majority are related to debugging the configuration file. They also suggest the necessity for a more in-depth analysis of the nature of the changes made to those commits.

Zampetti et al. [13] identified 79 bad CI practices by conducting semi-structured interviews with 13 experts and analyzing over 2,300 Stack Overflow posts, considering posts with four different tags: *continuous integration*, *Jenkins*, *Hudson*, and *travis-ci*. Additionally, Zampetti et al. [14] studied the evolution of changes to Travis configuration pipelines. They found that jobs and steps were the most frequently changed. Furthermore, they observed an increasing trend in the adoption of Docker by projects over time.

2.2. Popularity and usage of GitHub Actions

When GitHub introduced GitHub Actions (GHA) this significantly impacted the CI/CD landscape for GitHub repositories. Several studies have been conducted to investigate the impact of GHA on the overall usage of CI/CD tools.

Golzadeh et al. [8] conducted a longitudinal quantitative study on the adoption of CI/CD tools in 91K+ GitHub repositories related to npm packages. The study revealed a growing reliance on CI/CD tools, with more than 50% of repositories utilizing such tools as of May 2021. The authors found that GHA and Travis were the dominant CI/CD tools, being used by 90% of the considered repositories that relied on a CI/CD tool. They also observed that GHA replaced Travis as dominant CI/CD tool in just 18 months after its introduction as a consequence of many repositories migrating from Travis to GHA.

Kinsman et al. [15] analysed the impact of adopting GHA in 3,190 repositories. They found that the adoption of GHA resulted in an increase in the number of rejected pull requests and a decrease in the number of commits in merged pull requests. Based on a manual inspection of 209 GHA-related issues, they concluded that developers had a generally positive perception of GHA. These findings were confirmed by Chen et al. [16], who conducted a replication study on 6,246 repositories.

Decan et al. [9] conducted an analysis of the use of GHA in almost 70K+ GitHub repositories to gain a deeper understanding of the GHA ecosystem. They found that 43.9% of the repositories used GHA workflows and characterized these repositories and their workflow contents in terms of the usage of jobs, steps, and reusable Actions. They observed that almost all workflows rely on Actions, and that workflows are primarily used for automating

the development process, despite the potential for GHA to automate many other types of activities.

Valenzuela-Toledo and Bergel [17] conducted a preliminary study to examine the usage and maintenance of GHA workflows in ten popular GitHub repositories. They analyzed 222 commits to propose an initial taxonomy of workflow modifications.

Rostami Mazrae et al. [18] qualitatively investigated the reasons behind the adoption of CI/CD tools in software projects, the co-usage of multiple CI/CD tools, and the migration from one CI/CD tool to another one. For GitHub projects more specifically, they investigated the factors that led to GHA becoming the dominant CI/CD tool. The majority of reported migrations towards GHA were due to its strong integration with GitHub, its ease of use, and its large marketplace of Actions.

Saroar et al. [19] surveyed 90 Action producers and users to understand the motivations and best practices in using, developing, and debugging Actions, and the challenges associated with these tasks. They report that users prefer Actions with verified producers and more stars when choosing between similar Actions, and often switch to an alternative Action when facing bugs or a lack of documentation. Moreover, they report that more than half of the Action producers consider the composition of YAML files challenging and error-prone and would mainly check question and answer forums to fix issues with these YAML files.

Wessel et al. [20] suggested to consider studying GitHub Actions as a software ecosystem that faces similar challenges as traditional software library ecosystems [21, 22].

3. Motivating example

In order to use GHA for a repository, one or more *YAML* workflow files must be created and stored within the *.github/workflows* folder. As with any other software development component, workflow files evolve over time to better serve their purposes.

Figure 1 presents a visual diff of some changes made to a workflow file that automates the building, testing and code coverage analysis of some Java project. On the left is the old version, with lines highlighted in red representing removed or changed lines, and the dark red parts highlighting the parts of the line that were changed. On the right is the new version, with lines highlighted in green representing new or changed lines, and the dark green parts highlighting the changes that were made w.r.t. the previous version.

One can observe that changes may occur in different locations for different reasons. For instance, line 2 illustrates a change related to the events that trigger the workflow. This particular change can be considered a behaviour-preserving refactoring and simplification of the workflow file, since declaring events without specifying the branch will use the default branch of the repository (in this case: master). A common change consists of adding new steps (e.g., step 'Publish Test Report' on lines 17-19 on the right) or adding more lines to existing steps (e.g., lines 12 and 14 on the right that add extra arguments for the Java setup; line 26 that added an extra argument for the JaCoCo badge generator). Line 5 on the right is an example of a change to a job, by adding an extra name to it. Another common change is modifying the contents of existing lines, for example to update the version of the Action being used (e.g., from actions/checkout@v2 to actions/checkout@v3, from actions/setup-java@v1 to actions/setup-java@v3, and so on), to edit the name of a step (line 31 on the right), to modify the Java version to use for the build (line 13 on the right), and so on.

This example shows the various possible changes in workflow files and justifies the need for studies on when, why, and how developers modify workflow files. Such studies are likely to improve CI/CD practices, for example, by identifying common patterns and issues during workflow modifications and by providing tools to assist developers in writing and maintaining workflows.

4. Data extraction

To study the evolution of workflow files, a large dataset of GitHub repositories relying on GHA is required. We used the SEART [23] GitHub search engine[1] to select repositories. To mitigate the usual threats related to software repository mining [24], we excluded repositories that are used only for experimental or personal purposes, or that exhibit minimal evidence of software development activity. To do so, we selected repositories created before 2022 that were still active in 2022, had at least 100 stars and 100 commits, and were not forks. Considering these constraints, the final list of repositories included 62,673 instances.

On September 2022, we locally cloned these repositories to identify the presence of GHA workflow files in the *.github/workflows* directory of their default branch (as reported by the GitHub API) and found 22,733 repositories satisfying this criterion. Since our goal is to study the evolution of the workflow files in these repositories, we relied on a combination of the `git rev-list` and `git checkout` CLI tools to materialize the content of each workflow file in each repository for every Monday between November 2019 (the official release date of GHA) and September 2022, accounting for 148 time points. By considering weekly snapshots instead of all the commits that modified the workflow files, we mitigate the usual threats related to commits performed on parallel branches that are eventually merged [25]. This

[1] https://seart-ghs.si.usi.ch

```
name: Java CI with Maven
on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]
jobs:
  build:

    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up with Java 11
      uses: actions/setup-java@v1
      with:
        java-version: 11


    - name: Build with Maven (including running of all tests)
      run: mvn -B package --file pom.xml


    - name: Generate JaCoCo Badge
      id: jacoco
      uses: cicirello/jacoco-badge-generator@v2
      with:
        generate-coverage-badge: true
        generate-branches-badge: true

    - name: Log coverage percentage
      run: |
        echo "coverage = ${{ steps.jacoco.outputs.coverage }}"
        echo "branch coverage = ${{ steps.jacoco.outputs.branches }}"
    - name: Commit and push the badge (if it changed)
      uses: EndBug/add-and-commit@v7
      with:
        default_author: github_actions
        message: 'commit badge'
        add: '*.svg'
    - name: Upload JaCoCo coverage report
      uses: actions/upload-artifact@v2
      with:
        name: jacoco-report
        path: target/site/jacoco/
```

```
name: Java CI with Maven
on: [push, pull_request]


jobs:
  build:
    name: build and analyse
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up with Java 17
      uses: actions/setup-java@v3
      with:
        distribution: 'temurin'
        java-version: 17
        cache: 'maven'
    - name: Build with Maven (including running of all tests)
      run: mvn -B package --file pom.xml
    - name: Publish Test Report
      if: ${{ always() }}
      uses: scacap/action-surefire-report@v1
    - name: Generate JaCoCo Badge
      id: jacoco
      uses: cicirello/jacoco-badge-generator@v2
      with:
        generate-coverage-badge: true
        generate-branches-badge: true
        generate-summary: true
    - name: Log coverage percentage
      run: |
        echo "coverage = ${{ steps.jacoco.outputs.coverage }}"
        echo "branch coverage = ${{ steps.jacoco.outputs.branches }}"
    - name: Commit and push the svg badges and the json coverage summary (if it changed)
      uses: EndBug/add-and-commit@v9
      with:
        default_author: github_actions
        message: 'commit coverage badge and summary'
        add: '*.svg *.json'
    - name: Upload JaCoCo coverage report
      uses: actions/upload-artifact@v3
      with:
        name: jacoco-report
        path: target/site/jacoco/
```

Figure 1: Visual diff of some changes made to a workflow file for a Java project hosted in a GitHub repository.

resulted in a dataset of 4,127,760 workflow file snapshots (of which 271,422 are unique) in 22,733 GitHub repositories.

Figure 2 shows the evolution of the number of repositories and workflow files through time. We observe that both numbers are continuously increasing through time, indicating that more and more repositories are making use of GHA and more and more workflow files are created in these repositories. At the end of the observation period, there are 65,067 workflow files spread in 22,733 repositories. The figure also reveals a slight disturbance in the evolution of both numbers in November and December 2020. This coincides with restrictions imposed by Travis CI on its free plan for public repositories, which caused many repositories to switch from Travis CI to GHA during these two months, as already observed by Golzadeh et al. [8].

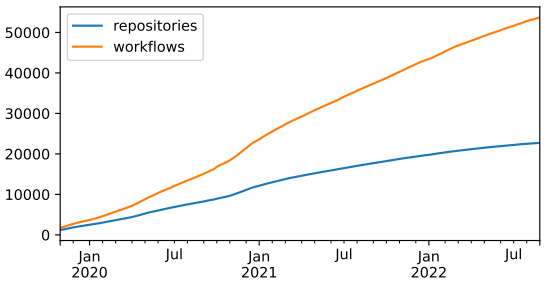

Figure 2: Evolution of the number of repositories and GHA workflows in our dataset.

5. Goal G_1: Coarse-grained changes in GHA workflows

Research goal G_1 aims to analyze and measure the coarse-grained changes that occur to GHA workflows in GitHub repositories. To accomplish this goal, we will answer three research questions.

$RQ_{1.1}$ When do repositories start using GitHub Actions?

To investigate the adoption of GHA in GitHub repositories, we analyzed the time it took for repositories to start using it. Previous studies [8, 18] have reported that GHA has become the dominant CI/CD tool on GitHub. This research question aims to understand the time it takes for repositories to adopt GHA as their CI/CD tool.

We distinguish between the repositories that already existed when GHA was introduced on GitHub and those that were created after. Indeed, repositories that were created before the introduction of GHA were likely to use another CI/CD tool before migrating to GHA and such a migration does not come for free. In their cases,

we are interested in the time they took to adopt GHA since its official release in November 2019. On the other hand, repositories that were created after the introduction of GHA are more likely to adopt it as their CI/CD tool because of its deep integration into GitHub. In their cases, we are interested in the time they took to adopt GHA since these repositories were created.

We therefore divided the repositories in our dataset into two categories: those that already existed before the public release of GHA in November 2019, accounting for 18,805 repositories; and the remaining 3,928 that were created after.

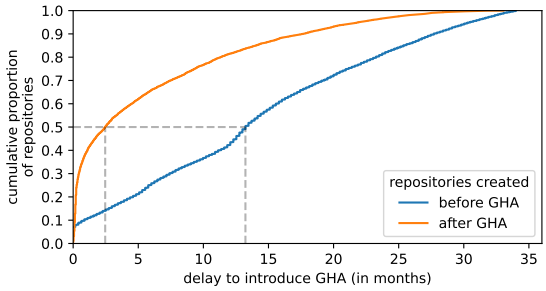

Figure 3: Cumulative proportion of repositories in function of the time (in months) to start using GHA.

Fig. 3 shows the cumulative proportion of repositories in function of the time they took to start using GHA. Its worth recalling that our dataset only contains repositories that are still using GHA in the latest considered snapshot, explaining why the proportions reach 100%.

Focusing first on the repositories that were created after GHA's public release (orange line in Fig. 3), we observe that it takes less than 3 months for 50% of them to adopt GHA after their creation, as indicated by the leftmost dotted line. After only 10 months, this proportion reaches 75%. The adoption rate for the repositories that already existed when GHA was released (blue line) is much lower: only 15% of these repositories adopted GHA after 3 months and it required 10 months to reach 50% of the repositories (as indicated by the rightmost dotted line), and even 21 months to reach 75%.

This supports our hypothesis that it takes more time for repositories already in place to adopt GHA, likely because they were already using another CI/CD tool, implying a longer delay to start using GHA due to the technical or organizational difficulties that may come with a migration to a new CI/CD solution, or simply due to the lack of need to carry out such a migration [8, 18]. On the other hand, adopting GHA in newer repositories is much easier thanks to the tight integration of GHA with GitHub, its ease of use, its low learning curve, and the ease of setting up new workflow files from scratch based on suggested configurable templates [19, 18].

The time to adopt GHA depends on whether the repository was created before or after GHA's official release. A majority of the repositories that were created after GHA adopted it within a few months after their creation. On the other hand, it took more than one year since GHA's release for most of the older repositories to adopt GHA.

$RQ_{1.2}$ Which types of coarse-grained changes are workflows subject to?

This research question aims to identify the kind of coarse-grained changes workflow files are subject to. We distinguish between the four following types of changes: *addition*, *modification*, *renaming*, or *removal*. To keep track of these changes, we attributed to each workflow file a unique identifier that is preserved through renamings. It is worth mentioning that some of these changes can co-occur (e.g., a file can be renamed at the same time than its content is modified) while some changes are causally dependent (e.g., a workflow file can only be removed after having been added previously).

We detect the four different types of changes as follows. The *addition* of a workflow file is detected when the workflow file is seen for the first time in a snapshot. Similarly, the *removal* of a workflow file is detected when the file is no longer visible in a snapshot. The *modification* of a workflow file is detected in a snapshot by comparing its content in the current snapshot with its content in the previous snapshot. These modifications are detected by comparing the SHA-256 hashes in consecutive workflows. Finally, the *renaming* of a workflow file is detected based on the following heuristic. The heuristic detects a renaming from A to B when A is removed at the same time that B is added. If A and B have exactly the same content (i.e., they have the same SHA-256 hash), we consider that A was renamed to B. If A and B differ in their content, we check whether there is no other workflow file C that was added or removed at the same time. If there is no such C, then we consider that A was renamed to B. By doing so, we ensure we are not exposed to false positives even if this implies we may miss renamings, e.g., in the case multiple workflow files are renamed at the same time.

For each repository and each pair of consecutive snapshots, we relied on the above approaches to detect changes. The most frequent workflow change type is *modification*, accounting for 73% of all changes. The second most frequent change type is *addition*, accounting for 22.8% of all changes. *Removal* is less common, accounting for less than 3.9% of changes. *Renaming* is the least common change type, accounting for 0.1% of all changes over entire considered period.

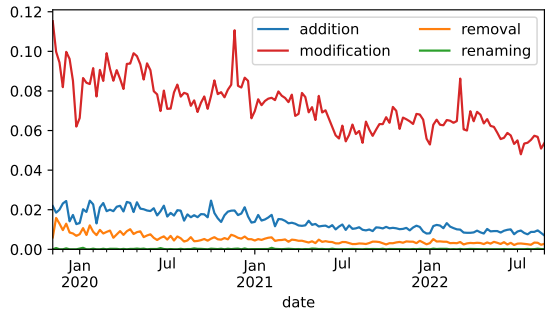

Figure 4: Evolution of the proportion of repositories exhibiting a change in a workflow file.

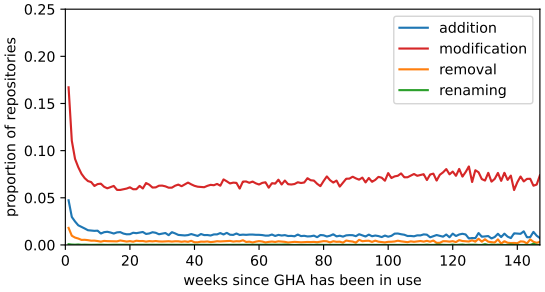

Figure 5: Proportion of repositories exhibiting a change in function of the time elapsed (in weeks) since their adoption of GHA.

Since a repository may have many workflow files being changed at the same time, we also computed for each snapshot the proportion of repositories exhibiting the *addition*, *removal*, *modification* and *renaming* of a workflow file. Figure 4 shows the evolution of these proportions. We observe that the most frequent change is *modification*, exhibited in around 9% of the repositories at the beginning of the observation period and slowly decreasing to around 6% of the repositories at the end of the observation period. Unsurprisingly, *additions* and *removals* are less frequently observed in repositories, ranging from 2.4% to 0.7% and from 1.5% to 0.2% of the repositories, respectively. Finally, *renamings* are barely never observed in the considered repositories.

Each week, on average, 7.2% of the repositories modify a workflow file, while 1.4% of them add a new workflow and 0.5% remove a workflow.

$RQ_{1.3}$ When do different types of coarse-grained changes occur?

Previous research question reported on the various types of changes that occur in workflow files. With the current research question, we aim to understand when those changes occur with respect to the adoption of GHA in each repository. We posit that the first few weeks after introducing GHA, workflow maintainers are likely to make many changes until the workflows reach a stable state and that, afterwards, only occasional changes are needed further. In order to verify this hypothesis, for each change detected in previous RQ, we computed the time between the introduction of GHA in the corresponding repository and the date of the change to the workflow file.

Figure 5 shows the proportion of repositories that exhibit a change in function of the time elapsed (in weeks) since GHA was introduced in each repository. We observe that the proportion of repositories exhibiting a change, regardless of the change type, is higher during the first weeks, and that this proportion quickly decreases through time to reach a quite stable value. For instance, more than 15% of the repositories made changes to their workflow files during the first week, but this proportion decreases to around 6% after six weeks only. Similarly, the proportion of repositories adding workflow files decreased from around 5% in the first week to approximately 1.5% after six weeks.

The slight variations that can be observed starting from week 100 is a consequence of the much lower number of repositories that have been using GHA for 100 weeks or more. For instance, while we have 22K repositories for the first week, we only have 4.8K repositories at week 100, 2.4K at week 120, and 574 at week 140.

These observations suggest that workflow files follow Lehman's evolution laws [26] of *continuing change* (workflows are regularly modified through time) and *continuing growth* (workflows are being added more often than they are removed). Although it is not surprising that workflow files are regularly modified in order to integrate new pipelines or new functionalities, we postulate that part of the observed modifications are the consequence of the difficulty to debug, test and validate workflows. This challenge was already identified by Saroar et al. [19] and in a qualitative study conducted by Rostami et al. [18]: *"You will see that you do a lot of typos and try to run the CI/CD 20 times until it works once. You copy-paste some examples from the Internet, you adapt it, but you forget to like there's a lot of details. It's often YAML files that are really prone to mistakes. So you make [lots of] commits until you get to the result you want to have. And there's no way to pre-test it on your local machine. So you just commit, push, wait for the build to run, and then look at the results."*

Repositories are more likely to change their workflow files within the first weeks after having adopted GHA. Nevertheless, we observe that each week, around 6% of the repositories modify a workflow file. This confirms that workflows are subject to the laws of *continuing change* and *continuing growth*.

6. Goal G_2: Fine-grained changes in GHA workflows

The second research goal $\mathbf{G_2}$ focuses on analysing **fine-grained** changes in workflow files at a **line-based** level.

$RQ_{2.1}$ Which types of line-based changes are workflow files subject to?

As a first step towards reaching our goal of studying the fine-grained changes in GHA workflows, we aim to identify how frequently lines are added, removed and modified in workflow files. To do so, we relied on the CLOC command-line tool [27]. CLOC is a tool that counts the number of lines in files. It has an option to compare two files, and reports on the number of lines that were *added*, *removed*, *modified* and *untouched* between the two files, distinguishing between lines of *code*, *blank* lines and *comments*.

We applied CLOC on all the workflow files that were modified, comparing the content of each workflow file with the content of the previous version of this workflow file. We found that the very large majority of the changes are related to lines of code. For instance, 87.8% of the lines that were either added, removed or modified are lines of code. A further examination of the modifications of the workflow files indicates that 69% of the modifications made to a workflow exclusively involve lines of code, 15% involve code and blanks, 7% involve code and comments, and 6.6% involve all types of lines (i.e., code, comment and blank lines). Overall, that implies that 97.6% of the modifications made to workflow files involve lines of code.

We also looked at how frequently lines are added, removed or modified. Figure 6 shows a Venn diagram reporting on the proportion of workflow modifications in which lines were added, modified or removed (or any combination of those). We observe that 40.11% of the changes consist of line modifications only, and lines are modified in 78.69% (40.11% + 18.86% + 6.09% + 13.63%) of the changes. Adding lines is the second most frequent change observed in workflow files, especially in combination with modifying lines. For instance, half of the changes (50.13% = 14.58% + 18.86% + 3.06% + 13.63%) involves adding lines, while "only" 26.23% of the changes (= 3.45% + 3.06% + 6.09% + 13.63%) involve removing lines. It is noteworthy that removing lines alone is infrequent.

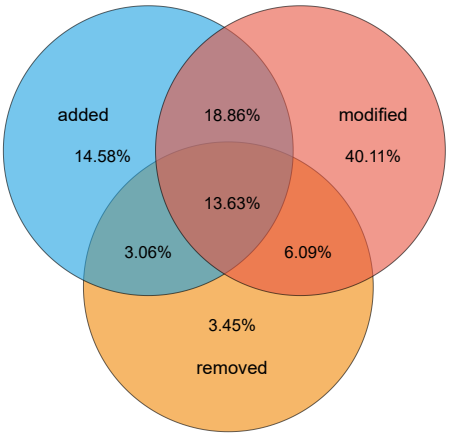

Figure 6: Percentage of workflow file changes containing added, removed or modified lines.

Nearly all the changes made to workflow files involve lines of code accounting for almost 9 out of 10 lines added, removed or modified. Modifying and adding lines are the most frequent operations made to workflow files.

$RQ_{2.2}$ When do different types of line-based changes occur?

We already observed in $RQ_{1.3}$ that, while repositories are more likely to change their workflow files within the very first weeks after having adopted GHA, changes are nevertheless observed during the whole lifetime of these files. With $RQ_{2.2}$, we aim to gain a better understanding of the type of changes that are made to lines of code in workflow files, hypothesizing that many lines of code will be added to the workflow files during their first weeks to add new functionalities until a stable point is reached, and then lines will be mostly modified for maintenance purposes.

As a first step, we start by computing the proportion of lines of code that are *touched* (i.e., lines that are added, removed or modified) during workflow changes. Figure 7 shows the evolution of this proportion in function of the time elapsed since GHA was adapted by the corresponding repositories. The figure shows the median and mean values, as well as the 25th and 75th percentiles, represented by the shaded light blue area. As explained in $RQ_{1.3}$, the higher variation on the rightmost part of the figure are due to the lower number of repositories that have been using GHA for more than 100 weeks.

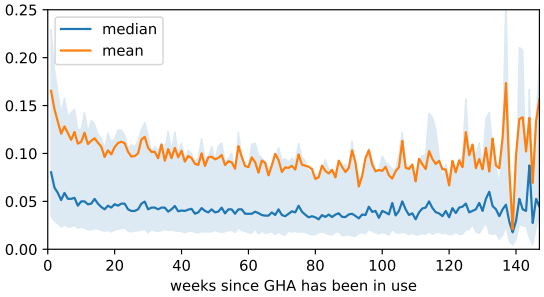

Figure 7: Proportion of lines of code *touched* during workflow changes. The shaded area corresponds to the interval between the 25th and 75th percentiles.

We observe that both the median and mean values exhibit a gradual decrease during the first year. The difference between the median and mean values indicates that the distribution exhibits a positive skew. Focusing on the mean value, the decrease in the number of lines of code touched is particularly visible in the first weeks, going from an average of 13.6% during the first six weeks to an average of 10.2% for the next year. This indicates that more changes are applied to the lines of workflow files in the early phase of the workflows' lifetime.

Since a line can be *touched* either because it is added, removed or modified, the two following analyses focus on the evolution of the number of lines that are added and modified. We do not report on the number of lines that are removed given that we observed in $RQ_{2.1}$ that removing lines is very infrequent.

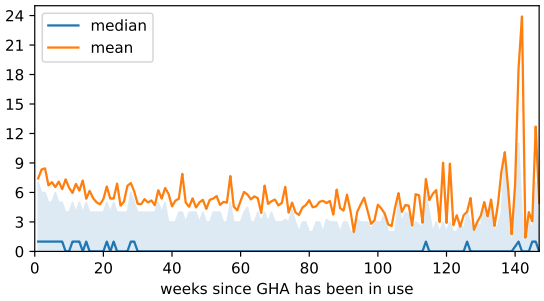

Figure 8: Number of lines of code *added* during workflow changes.

Figure 8 shows the evolution of the distribution of the number of lines of code *added* in workflow files, in function of the time elapsed since GHA has been adapted by their corresponding repositories. As observed in $RQ_{2.1}$, only half of the changes involve adding lines, explaining why the median value is very low (either 0 or 1). Focusing on the mean number of added lines of code, we observe a gradual decrease through time. For instance, the mean number of added lines decreased from an average of 7.4 during the first six weeks to an average of 5.6 during the next year.

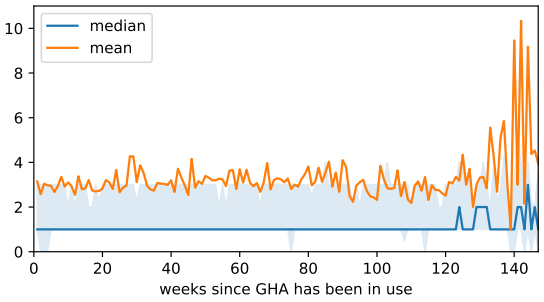

Figure 9: Number of lines of code *modified* during workflow changes.

Figure 9 shows the evolution of the number of lines of code *modified* in workflow files. As for Figure 8, the difference between the median and mean values indicates a positively skewed distribution. Focusing on the mean value, we observe that the number of lines of code modified is quite stable through time, only exhibiting a slight increase during the first weeks. For instance, the mean number of modified lines slightly increased from an average of 2.8 during the first six weeks to an average of 3.1 during the next year.

One out of ten lines of code are touched when a workflow file is modified, and this proportion is higher during the first weeks after the considered repositories started to adopt GHA. On average, 3.1 lines of code are modified and 5.8 lines of code are added during each modification of a workflow file. This suggests that the content of workflow files is subject to *continuing change* and *continuing growth*.

7. Threats to Validity

In this section, we discuss the threats that may affect the validity of our findings.

Internal validity relates to the extent to which the study results are influenced by the experimental treatment or condition being studied [28]. We analyzed the evolution of workflows in software development repositories by studying weekly snapshots.

Although one can study workflows commit by commit, Bird et al. [25] stated that git lacks a mainline and a file can change in parallel in different branches, which makes it difficult to track the linear history of a file. Therefore,

we chose to analyze snapshots instead of checking commits directly.

External validity concerns the generalisability of the results [28]. We only consider public software repositories with more than 100 stars and commits which are still under development and maintenance. These criteria are used to find projects best suited for software evolution studies in the case of GHA CI/CD tool. However, we can not generalize these findings on other GitHub repositories, including personal webpages.

Construct validity concerns the relation between the theory behind the experiment and the observed findings [29]. To detect the use of workflows in GitHub repositories, we identified the presence of a *YAML* file in the *.github/workflows* folder. This approach may lead to an overestimation of the presence and correctness of a *YAML* file. It can be due to problems in the corresponding workflow files or simply not being triggered for use. However, we believe that most of these workflows are indeed used since developers are unlikely to keep GHA workflows without using them or do not solve the issues in the workflow files.

Conclusion validity threats concern the degree to which reasonable conclusions have been derived from our analysis [30]. As our results only report quantitative observations, they are not exposed to such threats.

8. Conclusion

This study investigated the evolution of GitHub Actions workflows in collaborative software development by conducting a quantitative analysis on a dataset of 22,733 GitHub repositories containing over 4 million weekly snapshots of workflow files. This study aimed to provide preliminary findings on the changes made to these workflows over time.

As a first goal, we aimed to quantify the coarse-grained changes in GitHub Actions workflows in software projects. Our findings revealed that the majority of repositories created after the introduction of GHA tend to adopt it as their primary CI/CD tool within a few months. Furthermore, we investigated the types of changes observed in workflow files and found that modifications are the most common type of change, followed by additions, with a significant difference between the two. This seems to confirm that Lehman's software evolution laws of *continuing change* and *continuing growth* [26] also hold for workflow files. A notable difference with the evolution of regular source code, however, appears to be that most of the changes made to workflows occur within the first six weeks of adopting GHA in the project.

As a second goal, we aimed to quantify the fine-grained line-based changes in GHA workflow files. Our findings reveal that changes to code lines constitute 87.8% of all workflow file changes. These changes are present in over 95% of all snapshots studied and are predominantly in the form of line modification, addition, or a combination of both. Furthermore, we observed that the proportion of untouched lines generally increases with the workflow's lifetime, whereas most line additions occur within the first six weeks. In contrast, code modification occurs steadily throughout the workflow files.

This preliminary research showed the prevalence of changes in GitHub Actions workflow files. In future work, we aim to seek evidence for two other laws of software evolution postulated by Lehman [26], namely *increasing complexity* (stating that complexity increases unless work is done to maintain or reduce it) and *declining quality* (stating that quality will decline unless the system is rigorously maintained and adapted to operational environment changes). We will therefore study to which extent we can find evidence of quality-related problems such as code smells in workflow files, as well as the effect of preventive changes such as refactorings aimed at increasing the quality and reducing the complexity of workflow files.

Acknowledgments

This work is supported by the ARC-21/25 UMONS3 Action de Recherche Concertée financée par le Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique, and by the Fonds de la Recherche Scientifique - FNRS under grant number F.4515.23.

References

[1] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, M. Stumm, Continuous deployment at Facebook and OANDA, in: International Conference on Software Engineering (ICSE), IEEE, 2016, pp. 21–30.

[2] P. M. Duvall, S. Matyas, A. Glover, Continuous integration: improving software quality and reducing risk, Pearson Education, 2007.

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

[4] K. Beck, Extreme programming explained: embrace change, Addison-Wesley Professional, 2000.

[5] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in GitHub, in: Joint Meeting on Foundations of Software Engineering (FSE), 2015, pp. 805–816.

[6] M. Hilton, T. Tunnell, K. Huang, D. Marinov, D. Dig, Usage, costs, and benefits of continuous integration in open-source projects, in: International Conference on Automated Software Engineering (ASE), IEEE, 2016, pp. 426–437.

[7] M. Beller, G. Gousios, A. Zaidman, Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub, in: International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 356–367.

[8] M. Golzadeh, A. Decan, T. Mens, On the rise and fall of CI services in GitHub, in: International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022.

[9] A. Decan, T. Mens, P. R. Mazrae, M. Golzadeh, On the use of GitHub Actions in software development repositories, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2022.

[10] K. Gallaba, S. McIntosh, Use and misuse of continuous integration features: An empirical study of projects that (mis) use Travis CI, Transactions on Software Engineering 46 (2018) 33–50.

[11] C. Vassallo, S. Proksch, H. C. Gall, M. Di Penta, Automated reporting of anti-patterns and decay in continuous integration, in: International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 105–115.

[12] T. Durieux, R. Abreu, M. Monperrus, T. F. Bissyandé, L. Cruz, An analysis of 35+ million jobs of Travis CI, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2019, pp. 291–295.

[13] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, M. Di Penta, An empirical characterization of bad practices in continuous integration, Empirical Software Engineering 25 (2020) 1095–1135.

[14] F. Zampetti, S. Geremia, G. Bavota, M. Di Penta, CI/CD pipelines evolution and restructuring: A qualitative and quantitative study, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2021, pp. 471–482.

[15] T. Kinsman, M. Wessel, M. A. Gerosa, C. Treude, How do software developers use GitHub Actions to automate their workflows?, in: International Conference on Mining Software Repositories (MSR), IEEE, 2021, pp. 420–431.

[16] T. Chen, Y. Zhang, S. Chen, T. Wang, Y. Wu, Let's supercharge the workflows: An empirical study of GitHub Actions, in: International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, 2021, pp. 01–10.

[17] P. Valenzuela-Toledo, A. Bergel, Evolution of GitHub Action workflows, in: International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2022, pp. 123–127.

[18] P. Rostami Mazrae, T. Mens, M. Golzadeh, A. Decan, On the usage, co-usage and migration of CI/CD tools: A qualitative analysis, Empirical Software Engineering 28 (2023) 52.

[19] S. G. Saroar, M. Nayebi, Developers' perception of GitHub Actions: A survey analysis, in: International Conference on Evaluation and Assessment in Software Engineering (EASE), 2023.

[20] M. Wessel, T. Mens, A. Decan, P. Rostami Mazrae, The GitHub development workflow automation ecosystems, in: Software Ecosystems: Tooling and Analytics, Springer, 2023.

[21] A. Decan, T. Mens, P. Grosjean, An empirical comparison of dependency network evolution in seven software packaging ecosystems, Empirical Software Engineering 24 (2019) 381–416.

[22] A. Decan, T. Mens, E. Constantinou, On the impact of security vulnerabilities in the npm package dependency network, in: Proceedings of the 15th international conference on mining software repositories, 2018, pp. 181–191.

[23] O. Dabic, E. Aghajani, G. Bavota, Sampling projects in GitHub for MSR studies, in: International Conference on Mining Software Repositories (MSR), IEEE, 2021, pp. 560–564.

[24] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, An in-depth study of the promises and perils of mining GitHub, Empirical Software Engineering 21 (2016) 2035–2071.

[25] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, P. Devanbu, The promises and perils of mining git, in: International Working Conference on Mining Software Repositories (MSR), IEEE, 2009, pp. 1–10.

[26] M. M. Lehman, Laws of software evolution revisited, in: European Workshop on Software Process Technology (EWPST), Springer, 1996, pp. 108–124.

[27] A. Danial, CLOC, 2021. doi:10.5281/zenodo.7455676.

[28] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, A. Chatzigeorgiou, Identifying, categorizing and mitigating threats to validity in software engineering secondary studies, Information and Software Technology 106 (2019) 201–230.

[29] P. Ralph, E. Tempero, Construct validity in software engineering research and software metrics, in: International Conference on Evaluation and Assessment in Software Engineering, 2018, pp. 13–23.

[30] J. Maxwell, Understanding and validity in qualitative research, Harvard educational review 62 (1992) 279–301.