Catching smells in the Act: A GitHub Action Workflow Investigation

Cedric Willekens

Catching smells in the Act: A GitHub Action Workflow Investigation

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Cedric Willekens
born in Diest, Belgium

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Catching smells in the Act: A GitHub Action Workflow Investigation

Author: Cedric Willekens
Student id: 4530373
Email: please define authoremail

Abstract

In recent years, GitHub Actions (GHA) has emerged as the leading platform for Continuous Integration and Continuous Deployment (CI/CD) within the GitHub ecosystem, offering developers seamless workflow automation. However, as with other CI/CD tools, GHA workflows are susceptible to ”smells” which are suboptimal practices that can lead to technical debt, reduced maintainability, and performance issues. This thesis investigates the prevalence and nature of these workflow smells in GHA configurations. Through an extensive analysis of commit histories from 83 projects, we identify common patterns of frequent changes in GHA workflows that may indicate the presence of smells. We propose a set of potential GHA-specific smells, develop a tool to automatically detect these smells, and validate our findings through a contribution study involving 40 pull requests to open-source projects. After qualitatively analysing the comments on 32 pull requests we settle on 7 confirmed GHA workflows smells, including one novel smell previously unrecognised in the literature, This work contributes to improving the quality of GHA workflows and offers insights for developers to optimise their CI/CD processes. Finally, this research was also accepted as a paper to the SCAM 2024 conference.

Thesis Committee:

Chair:	Prof. dr. A Zaidman, Faculty EEMCS, TU Delft
University supervisor:	Dr. Benedikt Ahrens, Faculty EEMCS, TU Delft
Committee Member:	Ali Khatami Member, Faculty EEMCS, TU Delft

Preface

The completion of this thesis signifies the end of my Master's journey in Computer Science, one that has been both enlightening and profoundly transformative. Throughout this program, I have had the chance to deeply explore software engineering, investigate state-of-the-art technologies, and participate in the broader academic community.

This thesis embodies the completion of many hours of study, analysis, and refinement, marking not only a significant milestone in my academic career but also showcasing the skills and knowledge I have gained during the Master's program at the TU Delft. One of the most rewarding aspects of this journey has been the opportunity to contribute to the field by publishing a paper based on this research. This achievement has improved my understanding of the subject and academia.

I would like to thank Andy Zaidman and Ali Khatami for providing excellent guidance and support during the nine-month process of writing this thesis and publishing a paper. I am very grateful that Andy provided me with the opportunity to contribute to academia through this research and for the help provided by Ali whenever a second academic was required during the research.

Finally, I would like to thank my peers and staff members I have met over the past eight years for the great interactions, including numerous mental support coffee breaks, brainstorm sessions, and deep conversations.

Cedric Willekens
Delft, the Netherlands
August 17, 2024

Contents

List of Figures

Chapter 1

Introduction

Continious Integration (CI) and Continuous Deployment (CD) are emerging as one of the biggest success stories in automated software engineering [18]. These concepts were first described by Beck in 2000 as part of the *Extreme Programming* methodology [2] and have become vital for software development to ensure software quality and streamline the delivery process [9, 12, 29, 18, 32].

For many years CI/CD tools such as Travis-CI[1] and CircleCi[2] were very popular CI/CD platforms for GitHub users [7]. In order to facilitate the CI and CD process, GitHub has released their own tool namely *GitHub Actions (GHA)*, bringing automation directly into the software life cycle on GitHub [27]. The tool was released in november of 2019 and has seen a rapid increase in adoption rate among many popular open source projects [5, 21, 7]. The GHA's tight integration into GitHub ecosystem and the open-source library of reusable actions have been a key contributor to the current success of GHA [8], making GHA currently the most popular CI/CD tool [17, 32].

Both Gallaba et al. and Zampetti et al. have identified smells for continuous integration [35, 14] which can lead to technical debt in terms of maintainability, performance and usage of CI. Gallaba et al. studied Travis CI specification code and identified a number of anti-patterns. They created an automatic detection tool for these patterns that allowed them to fix these problems for open-source software projects [14]. Whilst Zampetti et al. conducted interviews with technical experts and mined stackoverflow posts to compile a list of smells which they were able to verify in literature [35].

1.1 Problem statement

We hypothesise that GHA configuration files are also affected by a set of smells, and that fixing them would help developers in creating a more effective CI/CD setup for their project. Given that five years have passed since the introduction of GHA, we believe that currently a substantial commit history has accumulated for GHA workflow configuration files in which we can identify recurring fixes to smells.

[1] `https://www.travis-ci.com/`
[2] `https://circleci.com`

Furthermore, earlier studies have brought attention to specific problems in GHA concerning security and resource utilisation [23, 7, 27, 4]. Nonetheless, there is either a lack of tools to identify these problems, or the issues have not been validated by the open-source community. Therefore, our goal is to identify anti-patterns in existing workflow configuration files by looking at the history of changes which we can then verify with the open-source community in order to judge their relevance.

1.2 Research Questions

In order to address the problem statement, we formulated the following research questions:

RQ1

Are there common patterns of frequent changes in GHA workflow configurations?

To answer this question we mine commits from 83 projects using GHA and analyse the evolution of the GHA workflow configuration files. The goal is to understand the existence of common patterns of frequent changes in GHA workflow configuration files.

Afterwards, we inspect these common patterns of frequent changes to determine whether some of these are fixes to frequently occurring problems in GHA workflow configuration files to answer our second research question:

RQ2

Are frequent change patterns in workflows indicators of workflow smells?

After establishing a set of potential GHA workflow configuration smells, our aim is to automatically identify these smells for our third research question:

RQ3

Are we able to automatically detect these workflow smells?

With the help of our tool in RQ3, we finally, aim to validate our list of potential GHA workflow configuration smells. We conduct a contribution study by opening 40 pull requests (PRs) to open-source projects and analysing the feedback provided by maintainers. This analysis will answer our fourth research question:

RQ4

To what extent do developers find the proposed fixes for the identified GHA configurations smells relevant?

1.3 Contributions

There are four main contributions for this thesis:

1. A list of seven relevant smells strongly supported by our contribution study. Within this list, we have identified one novel smell which was never discussed before in the literature.
2. A list of nine smells for which we have some indication regarding their relevance. However, more research is required to confirm the relevance. This list also includes a novel smell that has never been discussed before in the literature.
3. A tool that can automatically identify 18/22 smells with a reliable recall, precision, and F1 score.
4. A dataset with labelled changes for GHA workflows files in our replication package [33].

1.4 Thesis outline

In Chapter 2 we start with presenting some contextual background information on GitHub, CI/CD and known optimisation and security concerns in GHA. Then, in Chapter 3, we we describe the steps we took in order to label commits, identify potential smells and perform a contribution study. Chapter 4 describes our findings for each research question after following our methodology described earlier and answers the research questions. We discuss current research on CI/CD smells in Chapter 5. Finally, Chapter 6 concludes the thesis and identifies potential improvements and possibilities for future research.

Chapter 2

Background

Before delving into the specifics of our study, we will initially present some contextual background information. We start by introducing GitHub[1], the provider of GHA. Afterwards, we discuss the role of CI/CD in the software development cycle. Next, we discuss how GitHub enables CI/CD on GitHub by explaining how workflows are automated with GHA. Finally, we will discuss known optimisation techniques and security concerns related to GHA.

2.1 GitHub

GitHub is a web-based platform built around Git, a tool that allows easy version control and collaborative software development. On GitHub, users can create repositories in which they can upload their code, known as *commit*, and store it in the *mainline* which represents the latest state of the project. A developer can divert from the mainline, using a *branch*, in order to make changes to the project, such as adding features or fixing bugs, without having to worry about changes made by other developers [25]. GitHub uses the concept of a Pull Request (*PR*) to allow software developers to add these changes back to the mainline. A developer can open a new PR that contains changes to the current project, and other developers can comment and make suggestions for potential improvements before adding the changes to the main line [34].

2.2 What is CI/CD?

Continuous Integration (*CI*) and Continuous Delivery (*CD*) is the key enabler of DevOps, and plays a crucial role in the extreme programming methodology described by Beck in 2000. The goal of CI is to allow developers to frequently integrate their work into the mainline, while ensuring the quality of the contributions of developers [36]. This integration is triggered by a PR on platforms such as GitHub which will cause a variety of (automated) checks to start and lets the developer know if they have made any breaking changes in their

[1] `https://github.com`

code [13]. CI allows developers to pretend they are the only programmer working on a project. This allows them to not take into account the changes made by the other developers and rely on CI to successfully integrate the work done by all developers [2]. Furthermore, frequently integrating reduces the time spent doing integrations, decreases the number of bugs, and allows frequent refactoring of the code to prevent decay and incorporate new lessons learnt during the development process [13].

CD is a software development practice where software can be released to production at any time. It is considered an extension of CI because the software is verified to be ready for production by automating the release process. CD allows for faster feedback from stakeholders and quickly identifies problems that would otherwise be too hard to detect [24].

2.3 Workflow Automation with GHA

GitHub Actions is a platform integrated into GitHub in 2018. The goal is to automate CI/CD activities by defining workflows that can be triggered by several repository activities, such as opening a pull request. Workflows are defined in the *.github/workflows* folder using YAML files [22]. An example of a workflow can be seen in listing 2.1.

```yaml
name: Build Report
on:
  push:
    paths:
      - .github/
      - report/
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Get Date
        id: get-date
        run: |
          echo "::set-output␣name=minute::$(/bin/date␣-u␣"+%M")"
          echo "::set-output␣name=week::$(/bin/date␣-u␣"+%U")"
          echo "::set-output␣name=date::$(/bin/date␣-u␣"+%Y-%m-%d")"
          echo "::set-output␣name=hour::$(/bin/date␣-u␣"+%H")"
        shell: bash
      - name: Checkout code
        uses: actions/checkout@v2
      - name: LaTeX compilation
        uses: xu-cheng/latex-action@v2
        with:
          root_file: thesis.tex
          working_directory: report/
          continue_on_error: true
```

```
      compiler: latexmk
  - name: Upload thesis
    if: ${{always()}}
    uses: actions/upload-artifact@v2
    with:
      path: report/build/thesis.pdf
      name: Current State of Thesis
```

Listing 2.1: Workflow to compile latex report - release-report.yml

Workflows must contain at least a trigger condition, for Listing 2.1 this is a push to the repository containing changes to *.github/* or *report/* folder and a list of one or more jobs. Listing 2.1 contains exactly one job, *build*. Each job requires an operating system, *runs-on*, and a list of steps the job needs to run. There are two category of steps: 1) running a bash command or script and 2) running a predefined Action. Developers can create their own actions using JavaScript or Docker, which can be published on the GitHub Marketplace so that other users can also use them.

Once a workflow has been executed successfully, the results can be shown in various forms, such as through a GitHub Action bot. Similar to other GitHub bots, this bot functions as a GitHub user capable of submitting code changes, engaging in comments, and managing pull requests by merging or closing them.

GitHub Actions provides a free plan which includes a total of 500MB of storage and 2,000 minutes of runtime for workflows per month. When a user exceeds this limit, they will be billed using a per-minute rate depending on the runner they are using [16].

2.4 Optimisations for GHA

Resource optimisation in GitHub Actions has been discussed by Bouzenia et al. [4]. During their research they have compiled a list of six optimisations developers are already using in their workflows and four optimisations not currently being used but would also improve workflows. Below we list the optimisations for each category.
A total of six optimisations are currently being applied by developers:
Caching: Using caching prevents runners from having to download and install the same dependencies every time a new workflow is started.
Fail-Fast: This optimisation is applied to workflows using a *matrix* to create multiple jobs with similar configurations. Using fail-fast will allow GitHub Actions to cancel all the jobs related to the matrix that are queued.
Cancel-in-progress: This optimisation causes the workflow to stop running if the same workflow is triggered in the meantime.
Skip workflow: Workflows can be triggered by pushes or PR's. Developers can skip these workflow runs by adding *[skip ci]*, *[ci skip]*, *[no ci]*, *[skip actions]* or *[actions skip]* to their commit message.
Filtering target files: Developers can specify whether a workflow should run based on the type of file that has been modified.
Custom timeout: Developers can specify custom timeouts, over the default timeout of 360

minutes, to stop running unnecessary long workflows.
Additionally, four optimisation opportunities are presented by Bouzenia et al. [4], including:
Deactivate scheduled workflows after k consecutive failures: In the data set, it was discovered that 13% of the scheduled runs result in a failure. Upon manual inspection, it was noted that numerous scheduled workflows continue to fail before any intervention from the developer takes place. In order to save resources, a scheduled workflow should therefore stop running after k failures and notify the developer so that it is fixed sooner and does not waste resources on failures.
Deactivate scheduled workflows during repository inactivity: A scheduled workflow may build or release software everyday even if the state of the repository does not change, which is considered a waste of resources. Therefore, Bouzenia et al. [4] suggest a mechanism where these scheduled workflows do not run when there is no change to the repository, preventing unnecessary builds.
Run previously failed jobs first: In the dataset it was noted that, in workflows with multiple jobs, if a job fails, it sometimes also fails in the subsequent run of the workflow. Therefore, it is suggested to reorder jobs by the number of times they have failed. This allows for early identification of a problem without the need to run every job.
Job-specific timeouts: The current six hour time limit is considered too long, considering that the average run time is only 6.3 minutes. Instead, the time limit should be more dynamic, based on the maximum time (t_{max}) a non-timed-out run of the job has every consumed. The new timeout should be $timeout = 1.1 \times t_{max}$ as to give a 10% buffer in case jobs start to take longer caused by for example: an increase in test-suite size.

2.5 Security in GHA

Security is one of the five primary challenges in automating workflows [27]. Inadequately secured CI platforms pose risks like code theft, injection, and the circumvention of code reviews, which could result in malicious code being introduced into the code base [7]. To mitigate security risks, four security attributes have been defined for CI: admittance control, execution control, code control, and access to secrets [23]. Koishybayev et al. examined these security attributes for GHA, concluding that none of them are consistently upheld but can be addressed with proper workflow configurations [23]. Benedetti et al. developed a tool to automatically detect security vulnerabilities in GHA workflows [3].

Chapter 3

Methodology

The study follows a bottom-up approach to identify GHA workflow smells. In Section 3.1 we start with identifying common change patterns in workflow configuration files through mining commits. Subsequently, in Section 3.2 we analyse the common change patterns to identify our candidate smells. For these candidate smells we have implemented and analysed an automatic detector in Section 3.3. The goal is to investigate the feasibility and effectiveness of such a detector. Finally, we conduct a contribution study in which we fix our candidate smells in 40 open source software projects in Section 3.4.1. The whole process has been represented in Figure 3.1.

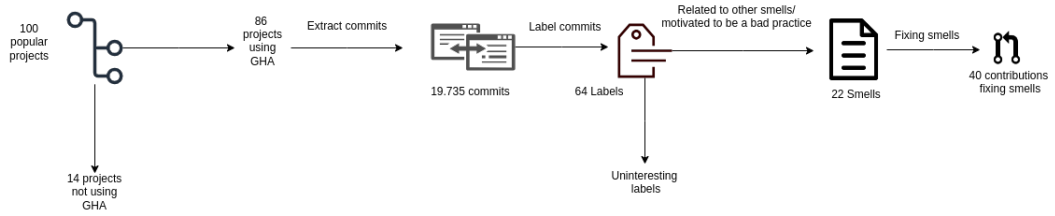

Figure 3.1: visual representation of methodology

3.1 RQ 1. Are there common patterns of frequent changes in GHA workflow configurations?

To answer our first research question, we will start by collecting projects from which we can mine commits. We will label these commits based on the change they provide to the GHA Workflow configuration files which will allow us to identify frequent change patterns.

3.1.1 Collecting Commits

To collect 100 different projects, we analysed the SEART [6] database because it provides a consistent dataset that was created after the introduction of GitHub Actions. Using a

consistent data set allows for easier reproduction of the experiment at a later point in time. Furthermore, SEART provides repository data for the five most popular programming languages: Javascript, Java, C#, Python and Typescript. Using the SEART database, we gathered repositories for each programming language sorted language, ordered by the highest number of stars descendingly. From this ranked list, we chose the top 20 projects for each programming language providing us with a total of 100 projects.

Of these 100 projects, we found that 17 projects did not use GitHub Actions and are therefore filtered out. For the remaining 83 projects, we want to collect all commits that modify a GitHub Actions workflow configuration. A GHA workflow configuration file can be identified by the expected location, *.github/workflows*, and the file extension, *.yaml* or *.yml*. All commits that changed at least one file following these criteria are included in our collection of commits. Considering that there can be multiple files changed per commit, we consider each GHA workflow changed per commit as a single change.

3.1.2 Labelling changes

To identify the common patterns of changes in GHA workflow configuration files, we coded the commits collected in Section 3.1.1.

Initially, we started manually labelling each change for a single project. In order to speed up the process of coding commits, automated scripts were developed for the frequently occurring categories. For each automated script, we verified that it detected the previous coded commits correctly and, we manually checked a subset of the newly coded commits by the automated script.

3.2 RQ2. Are there frequent change patterns in workflows indicators of workflow smells?

We assume that changes made to GHA workflow configuration files are an effort to improve the GHA worfklows in terms of quality and maintainability. Therefore, we hypothesise that the frequent change patterns identified in Section 3.1.2 are to some extent improvements to the quality and maintainability of the GHA workflow. Our goal is to identify these patterns which fix a certain *smell*. In this context we consider a smell to be a suboptimal configuration or practice for GHA worfklows which can negatively effect affect the maintainability, performance, security and maintainability.

Initially, we started by critically examining the labelled changes from Section 3.1.2 and exclude the changes which directly affect the behaviour of the workflow. These changes included, for example adding or removing a job, as these changes do not fix a configuration problem for a workflow. On the otherhand, we kept the changes which do not immediatly alter the behaviour of a workflow but are an indication of an improvement in terms of maintainability, performance, security and maintainability. These changes included, for example adding a name to a run-step, as this change helps developers understand the workflow and thus improve maintainability. Additionally, we grouped together closely related labels based on their pupose and context such as adding permissions and updating permissions.

On the other hand, for specific labels, such as adding if statement, we delve deeper into their context and purpose in order to better understand their purpose and further categorise them.

For the remaining labels, we grouped them into named smells based on their purpose and previous research on *GHA Security*, *GHA Performance/Optimsation* and *Other CI/CD smells*. The names and grouping were deliberated with an academic staff member of the TU Delft to ensure validity and consistency among the candidate smells.

Finally, we removed the candidate smells which require contextual knowledge (e.g. repository state or available actions) and candidate smells which have less than 2 occurrences.

3.3 RQ3. Are we able to automatically detect these workflow smells?

To speed up the contribution study process in Section 3.4, an automatic detector was implemented to help identify smells. We created python scripts which analyse GHA workflow configuration files and report the smells found. After implementing the detector, we selected 35 projects for on which we evaluated the performance of our tool.

3.3.1 Selecting projects

Our goal is to contribute to popular and currently active projects. Therefore, we again looked at the SEART [6] database for the same five programming languages as Section 3.1.1. However, now we looked at the 100 most popular projects for each programming language, combined these lists and sorted them based on the number of PR's they have accepted during February 15th 2024 and March 15th 2024. From this ranking, we took the first 40 projects which do not overlap with the projects used in Section 3.1.1 and which have a GitHub Actions workflow.

3.3.2 Tool evaluation

Before doing the contribution study we want to analyse the effectiveness of the detector. We initially established a ground truth by manually analysing the configuration files and identifying smells in the projects selected in Section 3.3.1. Then we ran our automatic detector on the dataset and recorded the false-postive and false-negatives. This allows us to calculate the precision, recall and F1-score using the following formulas in Equation (3.1).

$$\begin{aligned} precision &= \frac{TP}{(TP+FP)} \\ recall &= \frac{TP}{(TP+FN)} \\ F1-score &= 2 \times \frac{(precision \times recall)}{(precision + recall)} \end{aligned} \tag{3.1}$$

3.4 RQ4. To what extent do developers find the proposed fixes for the identified GHA configurations smells relevant?

In this section, our aim is to do a contribution study that will fix our previously identified smells in popular and active open-source projects. During the contribution study, our aim is to better understand the relevance of smells and the perception of the open source community regarding our identified smells in GitHub Actions.

We fixed the smells for 40 open-source repositories on GitHub and opened a PR to gather information about the developers' perception. Finally, we will label and categorise this feedback so that we can report our findings in a structured manner.

3.4.1 Creating PR's

We will run our tool on the projects selected in Section 3.3.1. For each project we strategically select between 2 and 7 smells to fix depending on the size and complexity of the fix. The goal is to keep the PRs relatively small so that they are easy to review for maintainers and they can provide specified feedback for each smell. In each PR we included a short description and motivation about the smells we are fixing in the PR. See Figure 3.2 for an example of our description.

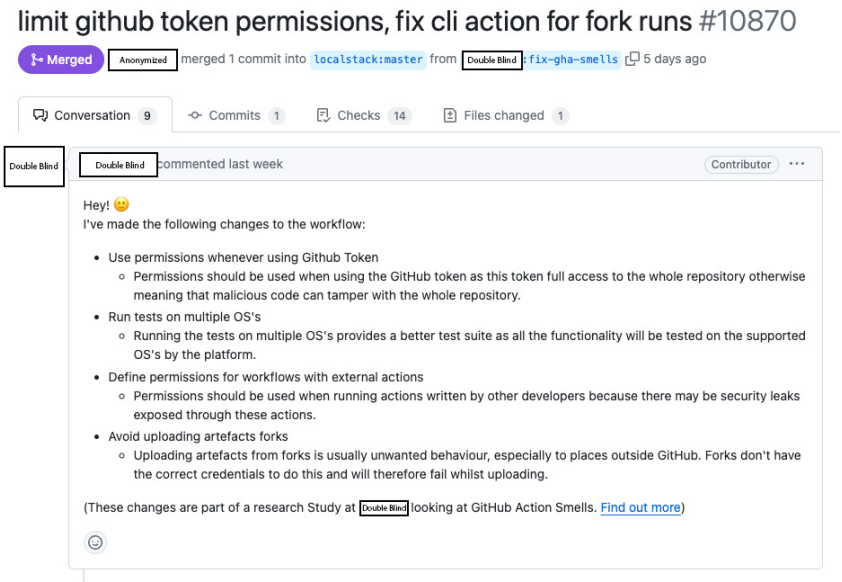

Figure 3.2: An example of the descriptions provided to the maintainers for each PR. This is a screenshot taken from PR10870.

3.4.2 Labelling comments

For the comments we received, we used an *open card sorting approach* [37]. This approach allows categories to emerge naturally from the data. To start, two people coded 20% of the comments separately. The codes were compared and discussed to get a consensus. After agreeing on the codes, the author of the thesis continued coding the remaining comments. Afterwards, the codes are sorted on a Miro[1] board for each smell and split between merged, pending and closed. Finally, these codes are be merged into larger categories to better understand the feedback provided for each smell.

[1] `https://www.miro.com`

Chapter 4

Results and analysis

To answer our research questions, we thoroughly followed the methodology described earlier. In this chapter, we present and analyse the results of our study for each research question. In Section 4.1, we start with describing the common patterns of changes we found during our mining study. Then, in Section 4.2, we discuss the potential smells we have identified based on the common patterns from RQ1. For these potential smells, we have created a tool which can automatically identify them and evaluated the tool in Section 4.3. Lastly, we discuss the results of the contribution study in Section 4.4

4.1 RQ 1. Are there common patterns of frequent changes in GHA workflow configurations?

To answer the first research question, we first look at what projects we were able to find that use GHA workflows. Afterwards, we look at the categories of common changes we were able to identify and which of these are the most frequent ones.

4.1.1 Collecting changes

We collected commits for five different programming languages, namely: JavaScript, Java, C#, Python and TypeScript. For each language, the 20 most popular projects were selected for analysis. After eliminating projects that do not use GitHub Actions, there were a total of 83 projects remaining. Table 4.1 highlights the number of number of commits and number of changes (a commit can have more than one change) we found for each programming language.

4.1.2 Categories of common changes

Using our tool described in Section 3.3, we identified a total of 64 distinct categories of changes made in GHA workflow configuration files. The 64 categories are grouped together into 8 higher level categories based on their purpose. Table 4.2 lists the 8 categories of higher level and the number of occurrences each category has in our data set. We identified that most of the changes are *Run Step Configurations*, which account for 36.5% of

Language	Number of projects	Number of commits	Number of changes
JavaScript	18	1,825	2,933
Java	15	519	856
C#	15	825	1,399
Python	15	3,105	7,665
Typescript	20	3,738	7,184
Total	83	10,012	20,037

Table 4.1: Number of changes in a workflow file per language

all the analysed changes and can be seen in 89.2% of the analysed projects. Now we will discuss each category in more detail.

1. Run step configuration is the most common change we have identified. These changes mainly add, remove, move or update run steps in order to modify the functionality of the workflow. The changes also included developers converting their run step to an action call, thereby leveraging one of the selling points of GHA.

2. Dependency versioning are changes related to updates made to a version in GHA workflow configurations. The most common version changes are made to Actions where either the version used is increased or the version used is converted to a commit hash instead. Other version updates we have seen are in the *runs-on* configuration and during package installation in a *run* command.

3. Scheduling is related to when a Workflow is started based on a schedule and how long the workflow is allowed to run for. The most common change in this category is the addition of timeouts. Adding timeouts allows developers to reduce the time that a workflow wastes when it gets stuck in an infinite loop.

4. Workflow organisation represents the category of changes made at workflow level, i.e., the adding/removing workflows, adding/updating names of a workflow, moving a workflow to a different file. The *5. Add workflow* category is found in every project because this is required to start using GHA.

6. Trigger conditions are used to configure when a workflow is run. The most common change here is the updating of the *on* configuration where developers can specify to, for example, only run on a commit push or only run on a push to a PR. This category also includes developers preventing running (parts-of) their workflow on a fork and white/black listing specific files or directories for which a workflow can be triggered.

7. Environment Setup are changes made to the software on which a workflow is run. This includes changing the operating system on which a workflow is run and changes made to environment variables within the operating system.

8. Job configuration involves adjusting the conditions and parameters under which a job is executed. The most common change is to add and remove jobs, followed by adding/updating the matrix used by jobs to prevent duplicate jobs.

9. Miscellaneous are small changes we only saw occurring a few times. These include adding/updating/removing "if" statements, comments, permissions, and changing formatting and styling of a workflow to make it more readable.

RQ1 analysis

Our analysis identified 64 patterns of frequent changes in GHA workflow configuration files. We were able to group these into eight distinct categories based on their purpose. We see that developers most often modify their run commands to improve workflow execution. Furthermore, developers also very often update action versions in order to benefit from new features and security enhancements.

Category	**Occurrences**		**Projects**	
Sub-category	#	%	#	%
Workflow Organization	**2,177**	**9.2%**	**83**	**100%**
Add/Remove workflow	1,907	86.5%	83	100%
Move/Refactor workflow	137	6.2%	33	39.8%
Add/Update workflow name, etc.	161	7.3%	33	39.8%
Run Step Configuration	**8,709**	**36.5%**	**74**	**89.2%**
Run command updates	3,395	39%	59	71.1%
Action configuration	2,741	31.5%	63	76%
Add/Remove run step	2,301	26.4%	60	72.3%
Update run step, etc.	272	3.1%	50	60.2%
Dependency Versioning	**4,508**	**18.9%**	**70**	**84.3%**
Bump version	2,160	47.9%	64	77.1%
Bump hash version	2,151	47.7%	19	22.9%
Use hash instead of version, etc.	197	4.4%	34	41%
Job Configuration	**1,159**	**4.9%**	**63**	**75.9%**
Add/Remove job	659	56.9%	45	54.2%
Matrix configuration	290	25%	44	53%
Update/Add job name, etc.	210	18.1%	37	44.6%
Environment Setup	**1,589**	**6.6%**	**45**	**54.2%**
Update env/env variable	1455	91.6%	43	51.8%
Update runs-on, etc.	134	8.4%	16	19.2%
Trigger Conditions	**1,973**	**8.3%**	**66**	**79.5%**
Update “on”	1,553	78.7%	65	78.3%
Prevent running on forks, etc.	420	21.3%	34	41%
Scheduling	**2,408**	**10.1%**	**28**	**33.7%**
Add timeout	2,254	93.6%	13	15.6%
Add/Update concurrency, etc.	154	6.4%	26	31.1%
Miscellaneous	**1,308**	**5.5%**	**70**	**84.3%**
Add/Update/Remove “if”	362	27.7%	34	41%
Add/Update/Remove comment, etc.	97	7.4%	27	32.5%
Access control configuration	373	28.5%	50	60%
Formatting and Styling	476	36.6%	57	69%
Total	**23,862**		**83**	

Table 4.2: Categories of common changes in the evolution of the GHA YAML configuration files. The complete table with all the distinct categories can be found in the replication package [33]. The percentages of sub-categories in the “Occurrences” column are calculated relative to their parent category. However, the percentages of sub-categories in the “Projects” column are calculated based on the total number of projects.

4.2 RQ 2. Are frequent change patterns in workflow indicators of workflow smells?

After evaluating all the frequent changes, we discovered 22 distinct smells grouped into 3 different categories: Security (3 smells), Performance/Optimisation (10 smells) and Other CI/CD smells (9 smells). Table 4.3 lists all the distinct smells we found organised by their respective category. Additionally, the number of projects in which there is at least one commit that fixes the smell, the total number of commits that fix this smell, and the backed research are listed. In the following sections, we explain each category and smell in more detail.

4.2.1 Security

Security-related smells are recognised through changes that address potential security vulnerabilities. A total of three security smells are identified:
(1) Define permissions for workflows with external actions requires workflows which are using external actions, meaning actions created by external developers, to have the *permissions* configuration. Such permissions can be applied on Workflow level or Job level. The permissions specified must be as restrictive as possible. The smell addresses the problem proposed by Koishybayev et al. where workflows are configured to be overprivileged in 99.8% of their dataset [23].
(2) Use commit hash instead of tags for action versions to prevent malicious changes to actions without the maintainer being aware. Tags in Git can be moved between commits, which allows a developer to tag a malicious version of an action to be run on all workflows. When the commit hash is used, the maintainer is certain about what is being run because commits cannot be modified without changing the hash. This issue is discussed by both Decan et al. and Saroar et al. stressing the importance of using commit hashes over tags when specifying an action version [7, 27].
(3) Set permissions for GitHub Token is closely related to *Define permissions for workflows with external actions* in that sense that this smell also requires a workflow to be configured with *permissions*. The GitHub Token used in Workflows is by default configured to have all the permissions. This means that, any malicious code which may enter a workflow, will then be able to use the GitHub Token in order to perform any modification to the repository. Also this smell is related to an observation made by Koishybayev et al. as workflows should be prevented from running over-privileged [23].

4.2.2 Performance/Optimisation

Performance and optimisation related smell are identified based on changes addressing resource usage. The goal for these smell is to limit resource usage, improve build times and decrease any potential costs as a result of unnecessary long running workflows. In Table 4.3 we observer 10 smells for this category. Below we go into more detail about the smell.
(4) Prevent running issue/PR actions on forks makes sure that actions which are expected to fail on forks, i.e. actions trying to make changes to an issue or PR, will not be run on a

fork. This avoids unnecessary resource usage through the workflow. This problem was also identified by Bouzenia et al. during their study on optimising resource usage [4].
(5) Avoid jobs without timeouts because timeouts are by default set to 6 hours by GHA. Repositories should use this to prevent workflows from running unnecessarily long [4].
(6) Stop running workflows when there is a newer commit in PR makes sure we do not waste resources building commits for which developers will not be looking at the result. The status of a PR is dependent on the result of the latest commit in the PR. Therefore, the result of the penultimate commit will not affect the result of a PR and can therefore be ommitted. Bouzenia et al. identified a similar problem with their *Cancel-in-progress* optimisation which causes the workflow to stop running if the same workflow is triggered in the meantime [4].
(7) Stop running workflows when there is a newer commit in branch is similar to the previous smell, as we assume that developers are only interested in the latest state of their branch to know if there are any problems which still need to be resolved before they can open a PR. Also this smell is related to the *Cancel-in-progress* optimisation discussed by Bouzenia et al. [4]
(8) Avoid running CI actions when no source code has changed will not lead to any new insights of the code base because there is nothing new to be checked. This can occur when, for example, only the README file is updated in a repository. In that scenario, it is unnecessary to build, test, lint, etc., the entire codebase since no modifications were made, and therefore the results will guaranteed be the same as previous run. Bouzenia et al. also suggested this optimisation as *Filtering target files* such that developers can specify wether a workflow should run based on the files that have been modified [4]. This smell can be fixed through specify the *paths* or *paths-ignore* configuration which acts as a white or blacklist respectively. With this configuration developers can specify changes to which file, file type or directory are allowed to trigger the workflow.
(9) Avoid executing scheduled workflows on forks will cause inefficient resource usage on inactive forks. Bouzenia et al. noted that scheduled workflows should be deactivated during repository inactivity. Because we often see that forks become inactive very quickly, scheduled workflows should not be run on forks [4]. In order to prevent this smell, developers should add an *if* condition checking for the repository name in each job which is part of a workflow that is being run on a scheduled interval.
(10)Avoid uploading artifacts on forks causes inefficient resource usage by wasting precious storage space of a contributor who will most likely never look at the generated artifacts. Instead, maintainers must prevent this from happening by adding a check to the upload to step to make sure that it only runs when the workflow is run for the main repository.
(11) Use 'if' for upload-artifact action to prevent unnecessary uploads. Maintainers should be aware of when artifacts are uploaded as the need for an artifact may vary depending on the result of the workflow. In case of a failing test, it it useful to upload the test output. However, in the same case, it is not worth uploading the compiled software to be used by others as this cleary contains a problem or bug which first requires fixing. Being mindfull of when artifacts are uploaded makes sure that resources are used efficiently [4]. In order to prevent this smell, maintainers should add and *if* condition to the *upload-artifact*

action, where they can specify in which case the artifact should be uploaded.
(12) Avoid deploying jobs on forks to prevent unwanted and failed upload attempts. When workflows try to perform a deployment of software on a fork there are two problems, either the deploy is successful which results in maintainers not having control of what is being deployed, or the deployment fails because of lacking credentials which results in a waste of computing resources as this will always fail [16]. To prevent this smell from occurring, maintainers should add a condition to the upload job or action to ensure that it is only run on the original repository.
(13) Avoid starting new workflow while previous one is running as this can cause inconsistent states and can be considered inefficient resource usage [16]. Therefore, developers should add *concurrency* to their workflows to prevent this from happening.

4.2.3 Other CI/CD Smells

Other CI/CD smells are changes addressing bad practices, configuration smells and the need for refactoring. These smells can lead to non-reproducible builds, inconsistent results, difficulty maintaining, and understanding workflows. A total of 9 smells are identified in this category which are listed in Table 4.3 and described in more detail below.
(14) Correct indentation makes sure that workflows are consistently formatted and are readable to other developers which makes it easier to maintain workflows [10, 11]. Similar to code in a repository, workflow configurations should be linted using a YAML linter which ensures consistent formatting and the correct indentation.
(15) Use fixed version for runs-on argument ensures that a workflow always runs on the same operating system (OS) and that changing the version of the OS on which the code is tested is a conscious decision by the developer. Furthermore, this also allows builds to be more reproducible as developers are aware of what OS version is being used [31, 35].
(16) Name run steps allows developers to better understand what a workflow is doing. Hence, developers can more easily identify the cause of a workflow failure which allows them to fix the problem swiftly [10, 11].
(17) Use cache parameter instead of cache option Using the *cache* option provided by the "install-*language*" action reduced the complexity and mis-configuration risk of the workflow. This makes the workflows more maintainable and easier to modify in the future [35].
(18) Use single-command steps allows developers to better understand at which point a workflow failed [10, 11]. If a workflow were to run both the lint and test command in one run step and either of them would fail, a developer would need to dive into the logs to understand which of the two failed. However, if they were split into two steps, the GitHub workflow UI will show which step exactly failed without the need to look in the logs.
(19) Run tests on multiple OS's allows developers to test their code against multiple production environments which gives developers more confidence that their new feature will not break the mainline [35]. To achieve this, developers can use the `matrix` strategy in which they can define multiple OS's such that the same job is run on each OS.
(20) Specify package version allows the developer to have more control over the environment in which their code is being run. This increases the reproducibility of the workflow and results in a more debuggable CI environment [31]. To fix this smell, package managers

such as *npm* and *apt* should be used which allow a version to be specified when programs or libraries are installed.

(21) Add comments to workflow allows developers to better understand the workflow which improves the maintainability and understandability of the workflow. Allowing developers to better understand the workflow will help them in case the workflow were to fail. Developers will be able to more easily understand the reason for the failure and thus more easily fix the problem [10].

(22) Run CI on multiple language versions allows developers to test their code against multiple production environments which gives developers more confidence that their new feature will not break the mainline [35]. To achieve this, developers can use the `matrix` strategy in which they can define multiple language versions which can then be used as a variable when installing the specific language. For each language version specified, a new job will be started in which the specific language version will be used.

Cat	ID	Smell	# Projects	# Total	Backed-research/ Motivation
Security	1	**Define permissions for workflows with external actions** Problem: Overly permissive access increases security risks. Solution: Specify minimal permissions.	39	82	[23]
	2	**Use commit hash instead of tags for action versions** Problem: Tags can be modified, leading to inconsistent behaviour. Solution: `uses: actions/checkout@<commit-sha>`	8	11	[7, 27]
	3	**Set permissions for GitHub Token** Problem: Default token has overly permissive access. Solution: Set permissions under `permissions` key.	4	8	[23]
Performance/Optimisation	4	**Prevent running issue/PR actions on forks** Problem: Actions fail due to lack of permissions. Solution: Add condition checking repository owner.	12	20	[4]
	5	**Avoid jobs without timeouts** Problem: Have a very long default timeout (6 hours), wasting resources and blocking workflows. Solution: Set timeout for jobs.	11	14	[4]
	6	**Stop running workflows when there is a newer commit in PR** Problem: Inefficient resource usage and inconsistent results. Solution: Use concurrency to cancel in-progress runs.	8	10	[4]
	7	**Stop running workflows when there is a newer commit in branch** Problem: Inefficient resource usage and inconsistent results. Solution: Use concurrency to cancel in-progress runs.	8	10	[4]
	8	**Avoid running CI actions when no source code has changed** Problem: Unnecessary resource usage when irrelevant files change. Solution: Specify trigger files using paths or paths-ignore.	7	15	[4]
	9	**Avoid executing scheduled workflows on forks** Problem: Inefficient resource usage for inactive forks. Solution: Add condition checking repository owner.	6	10	[4]
	10	**Avoid uploading artifacts on forks** Problem: Inefficient resource usage. Solution: Add condition checking repository owner.	5	6	[16]
	11	**Use 'if' for upload-artifact action** Problem: Unnecessary uploads waste resources and storage. Solution: Add condition to run only when needed.	4	5	[16]
	12	**Avoid deploying jobs on forks** Problem: Inefficient resource usage due to unnecessary deployments. Solution: Add condition checking repository owner.	2	2	[4]
	13	**Avoid starting new workflow while previous one is running** Problem: Inefficient resource usage and inconsistent states. Solution: Use concurrency groups to ensure only one runs at a time.	2	2	[4]
Other CI/CD Smells	14	**Correct indentation** Problem: Incorrect indentation reduce readability. Solution: Use YAML linter to ensure consistent indentation.	25	57	[10, 11]
	15	**Use fixed version for runs-on argument** Problem: Environment changes can lead to unexpected behavior. Solution: Specify exact version for runs-on.	12	17	[**?** 35]
	16	**Name run steps** Problem: Unnamed steps reduce readability and debugging. Solution: Use descriptive names for steps.	11	12	[10, 11]
	17	**Use cache parameter instead of cache option** Problem: Cache options increase workflow complexity and misconfiguration risk. Solution: Update workflows to use cache parameter.	9	11	[35]
	18	**Use single-command steps** Problem: Multiple commands per step reduce clarity and complicate debugging. Solution: Split complex steps into simpler single-command steps.	5	5	[10, 11]
	19	**Run tests on multiple OS's** Problem: Testing on single OS might miss OS-specific issues. Solution: Use `matrix` strategy to run on multiple OSs.	4	10	[35]
	20	**Specify package versions** Problem: Unspecified versions can lead to non-reproducible builds. Solution: Specify exact package versions in install commands.	3	5	[31]
	21	**Add comments to workflows** Problem: Lack of documentation reduces maintainability. Solution: Add comments explaining purpose and function.	3	3	[10]
	22	**Run CI on multiple language versions** Problem: Single version might miss version-specific issues. Solution: Use `matrix` strategy for multiple language versions.	2	13	[35]

Table 4.3: Identified fixed smells in the history of GHA configuration files changes.

RQ2 analysis

In conclusion, we have found 22 unique smells related to three different categories. Of these 22 smells, 20 we managed to relate to previous research in order to support our claim. Additionally, we have found 2 novel smells *Avoid uploading artifacts on forks* and *Use 'if' for upload-artifact action* which are not discussed in literature to the best of our knowledge, but for which we found motivation in the GHA documentation. Our next goal is to automatically detect these 22 smells so that we can perform a contribution study to verify our results from this section.

4.3 RQ3. Can we automatically detect GHA configuration smells?

After identifying the 22 smells in Section 4.2 our goal is to be able to automatically detect these smells. To do this, we have created a tool which is able to identify each smell. Our tool analyses each workflow independently, checking for the presence of a smell in a step, job or workflow depending on the smell.

In order to evaluate our tool we first manually identified all the smells in the dataset described in Section 3.3.1. This process took 1 person 2 days. Afterwards, we ran the tool against the same dataset and recorded our results in Table 4.4. For each smell, we report the precision, recall and F1 score using the following formulas: $precision = \frac{TP}{(TP+FP)}$, $recall = \frac{TP}{(TP+FN)}$ and $F1 - score = 2 \times \frac{(precision \times recall)}{(precision+recall)}$ where TP is true positive, FP is false positive and FN is false negative.

The results in Table 4.4 show the overall good performance achieved in the detection of smells with a median recall, precision, and F1 score of 91. 8%, 100% and 0.93 respectively. More notably, we see that 6 out of the 22 smells detectors (27%) achieve perfect precision, recall and F1-score.

On the other hand, we do see that Smell 12, 19, 20 and 22 scored relatively low recall values of 19%, 40%, 52.6% and 42.9% respectively. These detectors are less effective because more contextual information is required to identify all instances of these smells. Smell 12 relies on knowing which workflows perform a deploy, which proved to be non-trivial. Similarly, smells 19 and 22 require contextual information from other workflows, as these smells might be fixed in a separate workflow. For example, for smell 19 we verify whether tests are on multiple OS's. We test for this by check if a job is being run on multiple OS's in a single workflow. However, a maintainer can also decide to create a completely new workflow to run tests on a different OS. This means that, in practice the tests were run on different OS's, however, our detector would still report the smell because both worklfows only use one OS. Furthermore, Smell 20 also received a lower score because it relied on all commands containing the *install* keyword to support versioning; however, this was not the case.

Smell #	Ground truth	FP	FN	Recall	Precision	F1-score
1	59	0	0	100%	100%	1
2	88	0	1	98.9%	100%	0.99
3	17	0	0	100%	100%	1
4	27	2	6	85.7%	92.6%	0.86
5	101	0	0	100%	100%	1
6	38	3	5	87.5%	92.1%	0.90
7	37	1	1	97.3%	97.3%	0.97
8	33	4	7	80.6	87.9%	0.84
9	16	0	3	84.2%	100%	0.91
10	18	9	2	81.8%	50%	0.62
11	13	0	2	86.7%	100%	0.93
12	20	1	81	19%	95.0%	0.32
13	38	4	1	97.1%	89.5%	0.93
14	54	0	35	60.7%	100%	0.76
15	99	1	0	100%	98.99%	0.99
16	61	0	0	100%	100%	1
17	7	0	0	100%	100%	1
18	59	0	0	100%	100%	1
19	20	0	30	40%	100%	0.57
20	12	2	9	52.6%	83.3%	0.65
21	25	0	1	96.2%	100%	0.98
22	7	1	8	42.9%	85.7%	0.57
Total	849	**Median**		91.8%	100%	0.93

Table 4.4: Evaluation of automated detection of GHA smells.

RQ3 analysis

In conclusion, we created a detector that can achieve good overall performance when detecting the 22 smells. We were able to achieve a perfect score for 6 smells and a very good score for 12 smells. We noticed that the detector requires some more work for 4 smells in order to provide more contextual information to the detector, and hereby improve the detection of these smells. With this detector, we can now do our contribution study and evaluate the results in the next section.

4.4 RQ4. To what extent do developers find the proposed fixes for identified GHA configuration smells relevant?

In order to externally validate the relevance of the identified GHA smells in Section 4.2, we conducted a contribution study. For the contribution study we used the dataset described in Section 3.3.1. For each project in our dataset a PR was opened, fixing a subset of the smells our detector was able to find. A total of 40 PRs were opened between the end of April and the end of May 2024, giving developers at least three weeks to respond to our PRs. On 12th of June, we considered our contribution study finished, which means that comments placed after this are not considered.

#	Project	# Stars	Status	PR ID	Fixed Smells																					
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	Jackett/Jackett	11.4k	Merged	15274	–	–	–	–	–	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
2	jquery/jquery	58.9k	Closed	5480	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	X	X	X	–	–	X
3	oracle/graal	19.8k	Pending	8836	–	–	–	–	–	–	O	–	–	O	–	–	O	–	–	–	–	–	O	–	–	
4	prisma/prisma	37.5k	Merged	23965	–	–	–	–	–	–	–	–	–	✓	✓	–	–	–	–	–	–	–	X	–	–	–
5	nuxt/nuxt	52.4k	Merged	26937	–	–	–	–	–	–	–	–	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
6	cypress-io/cypress	46.2k	Merged	29416	–	–	✓	–	–	–	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
7	dotnet/AspNetCore.Docs	12.4k	Merged	32420	–	–	–	–	–	–	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
8	ray-project/ray	31.5k	Merged	44990	–	–	–	–	✓	–	–	–	–	–	–	–	–	–	X	–	–	–	–	–	–	–
9	parcel-bundler/parcel	43.2k	Pending	9672	–	–	–	–	–	O	–	O	–	O	O	–	–	–	–	–	–	–	–	–	–	–
10	halo-dev/halo	32k	Merged	5809	–	–	–	–	–	✓	–	–	–	–	✓	✓	–	–	–	–	–	–	–	–	–	–
11	doocs/leetcode	29.2k	Merged	2677	–	–	–	–	–	✓	✓	✓	–	–	–	✓	–	–	✓	–	–	–	–	–	–	–
12	spacedriveapp/spacedrive	29.1k	Merged	2412	✓	✓	–	–	–	–	–	–	✓	–	–	–	–	–	–	✓	–	✓	–	–	–	–
13	unoplatform/uno	8.5k	Merged	16508	–	✓	–	–	–	–	✓	–	–	–	–	–	–	–	–	✓	–	X	–	–	–	–
14	scikit-learn/scikit-learn	58.4k	Closed	28909	–	–	–	X	–	–	–	–	–	–	X	X	–	–	X	–	–	–	X	–	–	
15	microsoft/semantic-kernel	18.6k	Pending	6041	–	O	–	O	O	–	–	–	–	–	–	–	–	–	O	O	–	O	–	–	–	–
16	keycloak/keycloak	20.2k	Closed	29164	–	–	–	–	–	–	–	X	–	X	X	–	–	X	–	–	–	–	–	–	–	–
17	getsentry/sentry	37.1k	Closed	69915	X	X	–	–	X	–	–	–	–	–	–	–	–	–	X	X	–	X	–	–	–	–
18	dbeaver/dbeaver	37.8k	Closed	29273	X	X	–	–	–	X	–	–	–	–	–	–	–	–	X	–	–	–	–	–	–	–
19	commaai/openpilot	48.2k	Closed	32326	–	–	–	–	–	–	–	–	–	–	–	–	–	X	–	–	–	–	–	X	–	
20	abpframework/abp	12.3k	Pending	19665	–	O	–	–	O	–	–	–	–	–	–	–	–	O	–	O	–	O	–	O	O	–
21	App-vNext/Polly	1.2k	Merged	2097	–	–	–	✓	–	–	–	–	–	✓	✓	✓	–	–	–	–	–	–	–	–	–	–
22	jenkinsci/jenkins	22.5k	Pending	9236	O	–	–	–	–	O	–	–	–	–	–	O	–	–	–	–	O	–	–	–	–	–
23	trpc/trpc	33k	Closed	5702	–	–	–	–	–	–	–	–	–	–	–	X	–	–	–	X	–	–	X	–	–	–
24	appwrite/appwrite	41.5k	Pending	8075	–	–	–	O	–	O	–	–	–	–	–	O	–	–	–	–	–	–	–	–	–	–
25	gui-cs/Terminal.Gui	9.2k	Pending	3449	–	–	–	–	O	–	–	O	–	–	–	O	–	–	–	–	–	–	–	–	–	–
26	gpt-engineer-org/gpt-engineer	50.8k	Merged	1156	✓	–	–	–	–	✓	–	–	–	–	–	–	✓	–	–	–	–	–	–	✓	–	–
27	cheeriojs/cheerio	27.9k	Merged	3826	–	–	–	–	–	–	–	✓	–	–	–	✓	–	–	–	–	–	✓	–	–	–	–
28	remix-run/remix	28.1k	Pending	9478	O	O	–	–	O	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
29	localstack/localstack	52.5k	Merged	10870	✓	–	✓	–	–	–	–	–	–	–	–	✓	–	–	–	–	–	X	–	–	–	
30	netty/netty	32.9k	Closed	14077	–	X	–	–	–	–	–	–	–	–	–	–	–	–	–	–	X	–	–	–	–	–
31	openzipkin/zipkin	16.8k	Merged	3770	–	–	–	–	✓	–	–	–	–	–	–	–	–	–	–	–	✓	–	–	–	–	–
32	google/gson	23k	Pending	2684	–	–	–	–	O	–	–	–	–	–	–	–	–	–	O	–	–	O	–	–	–	–
Summary: 40 Opened, 15 Merged, 8 Closed, 17 Pending PRs;			Total Accepted:		3	2	2	1	2	4	4	3	5	3	4	5	1	0	1	2	1	2	0	1	0	0
			Total Rejected:		2	3	0	1	1	1	0	1	0	1	2	2	0	2	4	2	2	3	4	1	1	1

Table 4.5: List of repositories that received at least a response from the maintainers. The complete list of opened PRs is available in the replication package [33]

We received feedback on 32 PRs, which are listed in Table 4.5. Of the 32 PRs that received feedback, 15 have merged (46. 9%), 8 have closed (25%) and 17 are still pending (53.1%). Furthermore, in these 32 PRs, we received 165 comments, which were labelled. After analysing the labels using the miro board, they were grouped together into 7 categories:

1. Clarification questions: These are comments asking for further explanation about the smell, the reasoning behind the smell or the proposed fix for the smell.

2. Suggested Edit: These comments are related to our proposed fixes and point out small mistakes in our PRs or suggest a different fix for a smell.

3. Feedback: These are comments where the maintainers expressed their opinions or concerns regarding the proposed fixes.
4. Automated messages: These are comments created by software bots that automate certain tasks or reports such as signing the *Contributor Licencing Agreement* or reporting test results.
5. Decision-related comments: Comments about the decision to accept or reject the PR.
6. Appreciation: Comments expressing appreciation for the submitted PR and fixing the smells.
7. Unrelated comments: These are comments which are unrelated to the changes made in the PR.

Table 4.6 shows the number of labels for each category. We can see that most of the labels, hence also the comments, are related to feedback. In particular, feedback on closed on PRs where developers provide reasoning for closing the PR. Additionally, we see that PRs that received *suggested edits* are likely to be merged, and very unlikely to be rejected. We believe that this might be the case because the maintainers suggesting different ways of fixing a smell or highlighting small mistakes we made in the PR agree with the smells we have found in their workflows. A complete list of all labels and their number of occurrences is provided and the Miro board used to create the categorisations can be found in the replication package [33].

Category	**Merged**	**Pending**	**Closed**
Clarification questions	9	6	12
Suggested edits	30	15	0
Feedback	18	11	76
Automated messages	20	18	12
Decision related messages	9	2	10
Appreciation	26	4	8
Unrelated Comments	8	1	8
Total	121	57	126

Table 4.6: Number of comments per category. Note that a comment can have multiple labels and therefore the sum will exceed the number of comments we originally analysed.

For categories *Clarification questions*, *Suggested edits* and *Feedback* we will look at the smells to which they are related specifically. We believe that these three categories contain the relevant information on how maintainers perceive our identified smells. Table 4.7 shows the smells that received each type of category, including the state of the PR the smell was fixed in. In the following, we will discuss the results per category.

Clarification questions

These questions are posed by developers to better understand the impact of fixing the smell. We see that these questions were posed for *smell 1, 2, 3, 4, 6, 11, 12, 13, 15, 19* and *20*. For five smells, the PRs were merged after answering the questions posed by the maintainers.

PR state	Comment Category		
	Clarification Question	Suggest Edit	Feedback
Merged	2, 3, 4, 5, 20	4, 6, 7, 9, 10, 11, 12, 18	1, 3, 4, 6, 7, 9, 12
Pending	1, 2, 4, 6, 13	4, 5, 7, 10, 11, 12, 13	2, 13, 15
Closed	2, 4, 11	–	1, 2, 4, 5, 6, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22

Table 4.7: Type of comments received per each smell and the state of the associated PR(s).

This means that for the future we should provide as much detail as possible for maintainers when doing a contribution study as to avoid any potential misunderstandings or confusion about the impact and purpose of the PR.

On the other hand, seven smells were rejected after addressing the questions from the maintainers. The reasons for rejections varied, below we describe for each PR the reason for rejection:

PR29273: In this PR the maintainers initially ask for the motivation behind fixing the smell. After providing the answer to their question, the maintainers did not consider the smell a problem for their use case because "the whole purpose of this job is to validate commit messages so they match a particular pattern. Nothing fancy." [1]. The maintainers therefore decided to close the PR.

PR28909: A maintainer voiced their concerns about fixing *Smells 11, 15* and *19*. They are concerned that modifying a functioning workflow can lead to unintended consequences and potentially lead to regression in the workflow. A different maintainer was concerned about *Smells 4* and *12*, as they see a trade-off between maintainability and resource optimisation. The fix for these smells introduce multiple 'if' statements to prevent running the workflows on forks. However, introducing these statements comes at a maintainability cost, as the workflow becomes more verbose, and therefore more difficult to understand and modify. In this PR we see evidence that maintainers prefer maintainability over resource optimisation.

PR23965: For this PR, *Smell 19* was rejected by the maintainers. A maintainer originally asked for our motivation for running tests on multiple OS's. After addressing their question, the maintainer decided to reject our smell fix because they are concerned about the significant increase in jobs that our smell fix causes. The maintainer argues that applying our fix causes the number of jobs to increase from 206 to 380. Additionally, they do not see the benefit of running their tests on multiple OS's and are therefore worried about the waste of time and resources caused by the increase in jobs.

Suggested Edits

Within this category maintainers suggested edits for 10 of the smells which can be split across two different categories:

1) Alternative smell fixes: For *Smells 4, 9, 10* and *12*, maintainers have proposed an alter-

native way of fixing the smells. For each smell the proposed alternative was the same, an example is shown in Figure 4.1. The primary reason developers prefer this fix over ours is that it enables maintainers to more easily copy and paste these jobs or workflows into their other repositories.

Figure 4.1: Developer providing a different fix to *Smell 4*

2) Typographical corrections: These corrections mainly include fixes to capitalisation of repository names in fixes for *Smells 4, 9, 10* and *12* but also include styling errors accidentally introduced whilst fixing a smell(PR15274) or a typo in the name added to a run-step (PR2412). We see that no PR has been closed that received suggested edits which confirms that maintainers are accepting of these smell and are therefore willing to help with improvements to provide even better fixes for their workflows.

Feedback

Similarly to the category above, this category can also be split up into multiple sub categories, namely:
1) Positive feedback on the impact of fixing the smell This feedback was received for *Smell 3* and *7*. For *Smell 3*, the maintainers acknowledged that fixing the smell improves the developer experience. For *Smell 7*, the maintainers confirmed that the workflow works as expected after the smell was fixed.
2) Scepticism about the value of the contribution *Smells 1, 2, 4, 6, 10, 11, 12, 14, 15, 17, 20* and *21* received feedback regarding the usefulness of the proposed fixes and their concern regarding modifying a working workflow. *Smells 14* and *21* were rejected in *PR32326* because the maintainers found the motivation to fix the smell unclear. There were two PRs (insert here) addressing *Smell 1, 2, 15, 16* and *18* in which the maintainers highlighted that they prefer to discuss infrastructure before opening a PR. Ultimatly, these two PRs were closed without further discussion. Furthermore, *Smells 4, 10* and *12* received contradictory feedback in two PRs due to the maintainers' lack of clarity on GitHub's policy about executing workflows on forks, resulting in doubts about the smell's relevance. After discussions in both PRs we saw that *PR2097* was merged whereas *PR28909* was closed due to the uncertainty about the GitHub policy. Furthermore, *PR28909*, fixing *smell 11, 12, 15* and *20*, was closed because the maintainers were concerned about the potential side effects. We did see these smells be accepted in other PRs. We therefore believe that, similar to *Smell 14* and *21* described above, a better explanation should be provided regarding the motivation and result of fixing the smell to maintainers. In general, we find that the scepticism regarding the smells is because of two reasons:
1: The maintainers are uncertain about the necessity of fixing the smell. In particular, we

see that maintainers are reluctant to change a functioning workflow and they require a better explenation to help them understand the motivaton and necessity of fixing the smell.
2: Open-source projects tend to be cautious with modifying their infrastructure, such as GHA workflows, through external contributions. This was clearly highlighted in *PR69915* with the quote *appreciate the contribution but please in the future discuss changes to infrastructure....*
3) Reasoning on why the smell is not deemed applicable For *Smells 2, 4, 9, 10, 12, 15, 17, 19* and *22*, maintainers noted that the workflows in which we fixed the smells do not perform any critical operations and therefore do not see the need for fixing these smells.

To answer RQ4, we analysed the state of each smell in a PR, i.e., merged (✓) or closed (X). We will not consider the PRs that are still pending for this analysis. For the analysis, we have created three categories to which we will assign each smell: *1) mostly accepted smell*, which received at least 2 acceptances and at most 1 rejection showing good evidence of the maintainers' acceptance of the smell; *2) mostly rejected smells*, which have at least two rejections and more rejections than acceptance, this suggests that developers were generally not accepting of the proposed fixes to the smell; and *3) smells with mixed opinions or insufficient responses*, these are the remaining smells which either received insufficient responses, less than 2 accepted or rejected, or which have a mixture of accepted and rejected reactions indicating that there is no real consensus regarding the relevance and importance of these smells. Table 4.8 shows division of smells among these three categories. In the following, we will give a more detailed explanation of the results per category.

	Mostly accepted	**Mostly rejected**	**Mixed opinions**
Smell Ids	*3, 5, 6, 7, 8, 9, 10*	*2, 14, 15, 17, 18, 19, 21, 22*	*1, 4, 11, 12, 13, 16, 20*
Count	7	6	9

Table 4.8: Accumulated results based on Table 4.5

Mostly accepted (7/22 smells): We see that in total, we can assign 7 smells to this category which were merged across 14 PRs. In Table 4.5 we see that 3 smells received no rejections (*Smells 3, 7* and *9*) whilst 4 smells received 1 rejection (*Smells 5, 6, 8* and *10*). *Smell 5* received a rejection in *PR69915* because the maintainers do not accept external contributions to their infrastructure. *Smell 6* was not accepted because in *PR29273* because the maintainers did not see the value in fixing the smell as the workflow only performed simple actions, validating commit messages. Lastly, we saw *Smell 8* and *10* be rejected in *PR29164*. The maintainers provided the following feedback for the rejections of these semlls: they fixed *Smell 8* in a different manner, by adding logic to the workflow, as a job, which verifies whether the remaining part of the flow should run. *Smell 10* cannot be fixed because the workflows rely on artifacts being uploaded and downloaded by other workflows. Therefore, preventing this from happening on forks will cause other workflows, requiring these artifacts, to fail. Despite encountering some rejections in this category, we attribute them to specific project criteria or maintainers' reluctance to alter their infrastructure. Thus, we conclude that there is generally positive acceptance among maintainers for these smells.
Mostly rejected (6/22 smells): In total, 6 smells are assigned to this category. *Smell 2* and

18 were accepted in 2 PRs and rejected in 3 PRs. For both smells, 2/3 rejections were because maintainers did not want to accept external changes to their infrastructure and because maintainers did not see value in fixing the smell as the workflow only performed simple actions, *PR69915* and *PR29273* respectively. For *Smell 2*, the maintainers in *PR14077*, argued that the using hashes instead of tags is not required for *official GitHub actions* but do see their usefulness for *external actions* not maintained by GitHub. For *Smell 18*, the maintainers were concerned about creating duplicate step conditions when fixing the smell. *Smells 14, 15, 17* and *19* were rejected in all PRs. For *Smells 14* and *15*, maintainers demonstrated *scepticism about the value of the contribution* and their concern about *fixing a working system*. *Smell 19* was rejected because maintainers did not find the smell applicable to their project. These findings highlight that most rejections of smells are because of the scepticism of maintainers about the value of the contribution and their worries about fixing something that does not look broken.

Mixed opinions/inconclusive (9/22 smells): In Table 4.5 we see that *Smells 4, 16 and 20* have an equal number of of rejections and acceptances which means we cannot draw any clear conclusion for these smells. Furthermore, *Smells 1, 11* and *12* have 3, 4 and 5 accepted PRs respectively, but also received 2 rejected PRs in which maintainers were *sceptical about the value of the contribution*. Despite the high number of acceptance, having received multiple rejections we cannot confidently state that these smells are accepted amongst maintainers. Lastly, *Smells 13, 21* and *22* received only one response, providing insufficient data to draw any conclusion. In conclusion, because of the mixed responses and lack of responses for certain smells, we cannot draw any confident conclusion and further investigation is necessary.

RQ4 analysis

We found consensus within the open-source community for 7 smells (*Smell 3, 5, 6, 7, 8, 9* and *10*) as they have received at least two acceptances. *Smell 10* was one of our novel smells identified in Section 4.2. Therefore, having this smell also be accepted by the open-source community is a key contribution of the thesis. 9 of our identified smells received mixed opinions (*Smell 1, 4, 11, 12, 13, 16* and *20*). We believe that for these smells a larger contribution study is needed because these smells either only received one response or an equal number of rejections and acceptances.

4.5 Threats to validity

Threats to validity may impact our ability to interpret or draw conclusions from our study. Perry et al. argues that *external*, *internal* and *construct validity* are the three most important threats to take into account [26]. Additionally, we have added conclusion validity as a fourth threat for this study to ensure that we provide sufficient evidence to guide practical decisions [15].

4.5.1 External validity

External validity is concerned with the generalisability of the results. In our study, the smells that we have identified and their validation are based on 83 and 40 open source GitHub projects, respectively. This means that we have two potential external threats to validity:
1) The limited number of projects used may not be representative of all the GHA workflow configurations for every project. We have tried to eliminate this risk by selecting popular and active projects in a range of programming languages. By selecting popular projects, we expect them to be a good representation of the general open-source community because there are a significant number of people interested in and contributing to these projects. For the contribution study, we specifically selected active projects to ensure that we would get valuable feedback from the open-source community and not have the PRs closed or left open because the project is not being actively maintained anymore. For both the mining and contribution study, we included projects from five different programming languages in our dataset. This ensures that the smells we have identified are general and can be applied to multiple programming languages.
2) All the projects we have analysed are open-source projects; this means that closed-source projects might not be dealing with the same smells as open-source projects. To the best of our knowledge, we have not found any research that indicates that open-source and closed-source projects are using GHA in different ways and would thus run into different problems. However, it would be useful for the future to identify potential differences in the use of CI/CD between open-source and closed-source projects.

4.5.2 Internal validity

The identification of worklfow smells (RQ2) is largely based on the experience and knowledge of the author to interpret the patterns of frequent change. This could lead to a potential bias in which the author overlooks some smells or incorporates too much of his own opinion without any specific evidence. We eliminated this bias by having a second academic review of potential smells with their examples and coming to a consensus before continuing the research. Furthermore, we have also identified previous research that validates most of our smells and confirms the soundness of our methodology.

4.5.3 Construct validity

Construct validity refers to the degree to which a test or evaluation correctly assesses the theoretical construct it claims to measure. In our case, we measure the perceived relevance of the smells we fixed in a PR.

In order to ensure that we fully measure the perceived relevance, we evaluated both the status of the PR, i.e. merged equals accepted and closed means rejected. Additionally, we also evaluated the comments posted on the PR by maintainers because within the discussion of the PR we were able to ask them for further clarification on acceptance and rejections of a specific smell. This gives us a more detailed understanding per smell as we fixed several

smells in a single PR, and allowed us to better understand why the maintainers found the smell relevant or not.

4.5.4 Conclusion validity

The conclusion for this thesis drawn in RQ4, our contribution study, are based on a limited number of projects. Hence, we see that the majority of the smells (9/22 smells) have received mixed opinions and are thus inconclusive. In the future, more work should be done to further evaluate the smells to gain a better understanding of these inconclusive smells.

Chapter 5

Related Work

CI smells have been explored in the past both for a number of CI platforms. This chapter discusses a selection of the research done in this area that is related or foundational to our work. We start by identifying research that has been done specifically related to GHA. As far as we know, there is no existing literature that specifically addresses smells in GHA, but we did find studies concerning issues identified within GHA workflows.. We identify two themes closely related to our work, namely: *Security* and *Optimisations*. Afterwards, we look beyond GHA and discuss the literature that focuses on other CI platforms. Lastly, we discuss the literature focused on detecting smells in CI.

5.1 GHA Research

Since the release of GitHub Actions in 2019 the academic community has started to investigate several aspects. In the following, we will discuss GHA research that focusses on improving workflows in various aspects, including security and optimisation.

Koishybayev et al. [23] investigated the security of GitHub actions by first identifying four fundamental security properties that must hold for a CI/CD system, namely: *Admittance Control*, *Execution Control* and *Code Control*. They examined whether GitHub Actions enforces these properties and compared their findings to other popular CI/CD platforms, noting that GHA is enforces the least of these properties. Afterwards, Koishybayev et al. do a mining study to analyse the usage of workflows and the effect on the security properties. They come to three conclusions: 1) 99.8% of workflows are overprivileged, 2) 97% execute at least one action which is created by an unverified creator, and 3) 18% of workflows use actions which are missing security updates.

At the same time Benedetti et al. [3] created seven security checks for four security categories. Using these checks, they implemented an automated checker to automatically assess the presence of these security vulnerabilities in GHA workflows. They identified 24,905 security issues in 131,168 workflows.

Decan et al. [7] highlights the consequences of having an unsecured CI platform ranging from manipulating pull requests to stealing or injecting code as well as bypassing code reviews to push unreviewed code. Regardless of these risks, it is still common practice

to rely on actions made by other developers. GitHub strongly recommends only using actions from trusted creators; however, Decan et al. [7] points out that even these can be compromised and that extra care should be taken. Therefore, they suggests only using the commit SHA as a reference to action and not the version tag as they can be manipulated. However, they find that this is a very rare practice and most actions are still referred to using version tags.

Furthermore, Saroar et al. [27] identified security concerns as one of the five major challenges for developers when automating workflows. Their survey concluded that 1 in 69 (1.4%) participants did not want to use GitHub Actions due to security concerns. Like Decan et al., Saroar et al. [27] express concerns about the possibility of actions being modified without the user's awareness, which could lead to the theft of secrets. Furthermore, they also suggest that developers should be using the commit SHA to reference an action but that this is still uncommon practice.

Lastly, resource optimisation in GitHub Actions has been discussed by Bouzenia et al. [4] 1.3 million workflow runs from 952 open-source repositories were analysed based on a set of metrics to quantify the resource usage of workflows. Using these metrics, Bouzenia et al. [4] is able to identify current optimisations and suggest new potential optimisations that developers could apply to their workflows. They found evidence for six optimisations currently being applied by developers. Furthermore, Bouzenia et al. [4] suggests four further optimisation opportunities for developers to further improve their workflows.

5.2 CI/CD Smells

Previous research has investigated CI smells affecting the performance of the development cycle and workflows. Duvall [11, 10] identified 16 CI/CD patterns and anti-patterns. Building on the work of Duvall [11, 10], Zampetti et al. [35] inferred a catalogue of CI smells through interviewing 13 experts from 6 companies and surveying 2,322 Stack Overflow[1] discussions. This study allowed them to compile a list of 79 CI smells being organized into 7 different categories which is surveyed with 26 other professional developers to identify the relevance of each smell. This extended approach allowed them to identify 44 new CI smells which are not covered by Duvall [11, 10]. These newly discovered smells are specifically related to the CI infrastructure, the configuration of the build, and the testing and quality checks.

5.3 CI/CD Smell detection

Automatically detecting smells helps developers with identifying these early and easily. This results in a better CI/CD setup resulting in better productivity [28, 19].

Gallaba et al. [14] investigated the use and misuse of CI features specifically on TRAVIS CI. They analysed 9,312 open-source systems using TRAVIS CI and defined four anti-patterns including: 'Redirecting Scripts into Interpreters', 'Bypassing Security Checks',

[1] www.stackoverflow.com

'Using irrelevant Properties' and 'Commands Unrelated to the Phase'. Using their proposed tools: HANSEL, detecting these four anti-patterns they observed that 894 out of 9,312 (9.6%) subject systems have at least one anti-pattern and of which 832 (96%) have two or more anti-patterns. These anti-patterns are fixed automatically using their second proposed tool, GRETEL, fixing 174 instances and creating pull requests for each. A total of 49 pull-requests received a response from the developers and 36 of them have been accepted. Out of the 13 rejected pull-requests, one was rejected because the developer did not agree with the benefits of the change, two were rejected because they caused a failing build because of changes made between applying the fixes and opening the pull-requests. In another two rejected pull-requests, the developers did not understand the changes being made. Gallaba et al. [14] concludes that anti-patterns in TRAVIS CI do impact a considerable proportion of users and that teams should mitigate anti-patterns.

Vasallo et al.[30] investigated the automatic detection of anti-patterns of CI. Their tool is able to detect four relevant anti-patterns and were able to detect 3,825 instances of these smells across 18,474 build logs of 36 popular JAVA projects. Vasallo et al.[31] created a similar tool for GitLab, *CD-Linter*, which focuses on detecting smells through configuration files. They identify four detectable smells: *fake success*, *retry failure*, *manual execution*, and *fuzzy version*. To verify their smells and detector they opened 145 issues reporting the existence of these smells and received a response rate of 74% with 53% of maintainers reacting positively to the smell.

Chapter 6

Conclusions and Future Work

In Chapter 1 we presented four research questions to provide insights into the existence and relevance GHA workflow smells. We used a bottom-up approach by first performing a mining study on 83 projects and collecting 20,037 changes to GHA workflow configuration files. Through the combination of automatic scripts and manual labelling, we were able to categorise each change. We conclude that the maintainers most often change their run commands, followed by updating version of actions and adding timkeouts to workflows.

For each category of change, we critically evaluated its purpose and created a candidate list of 22 potential GHA workflow configuration smells. Amongst these smells, we found two novel potential smells: *Smell 10: "Avoid uploading artifacts on forks"* and *Smell 11: "Use 'if' for upload-artifact action.*

For each smells we implemented an automated checker to easily identify the smells during our contribution study. We evaluated our automatic checker using precision, recall, and F1 score. We concluded that for 6 smells, the detector achieved a perfect score, whereas for *Smell 12, 19, 20 and 22* the detector was only able to achieve a recall score of 19%, 40%, 52,6% and 42.9% respectively.

Using the automatic detector, we opened 40 PRs to popular and active open-source projects. For each PR we collected the feedback provided by the maintainers, which we coded and categorised in order to better understand the relevance of each smell. For 9 smells the results are inconclusive because we received mixed opinions from the PRs. These smells include our other novel smell: *Smell 11: "Use 'if' for upload-artifact action*. To get a better understanding of the relevance of these smells a larger contribution study should be performed. Furthermore, we identify 7 smells for which the maintainers agree with the relevance of the smell, including one of our novel smells: *Smell 10: "Avoid uploading artefacts on forks"*.

Finally, we have also presented this research as a paper at the SCAM2024[1] conference and are pleased to report that it has been accepted and will be included in the conference proceedings.

[1] `https://www.ieee-scam.org/2024/`

6.1 Contributions

The thesis provides three main contributions:

1. A list of seven smells for which we have strong indication for their relevance through our contribution study. Within this list, we have identified one novel smell which was never discussed before in the literature.

2. A list of nine smells for which we have some indication regarding their relevance. However, more research is required to confirm the relevance. This list also includes a novel smell that has never been discussed before in the literature.

3. A tool that can automatically identify 18/22 smells with a reliable recall, precision, and F1 score.

4. A dataset with labelled changes for GHA workflows files in our replication package [33].

6.2 Future work

The results of this thesis provide a good starting point for researchers, developers, and educators to continue working with GitHub Actions. Below we provide some potential next steps which can be taken in order to continue researching and improving GHA workflow configuration smells.

Improve automated detector and automated fixing of smells: we noted that 6/22 smells were difficult to automatically identify because they require contextual information. As future work, we could concentrate on providing this contextual information to improve the automatic detector. In addition, we can also suggest automated fixes for automatically detected smells. This tool can then be integrated into IDE's in order to help developers with improving their workflows.

Extended contribution study: during our contribution study we identified 9 smells, including one novel smell, which received mixed opinions. It would be very useful to perform another, larger contribution study for these potential smells to get a better understanding of their relevance. During our contribution study, we learnt the importance of providing a good and concise explanation about the purpose of (fixing) the smell. This should reduce the number of closed PRs because the goal of the PR was unclear.

Awareness of GHA workflow smells: finally, during the contribution study we identified multiple occasions where maintainers were uncertain about specific policies or best practices for GHA. We believe that our 7 identified smells can provide a good foundation to help maintainers better understand certain policies and best practices for GHA. Therefore, researchers and practitioners can use this new knowledge to create educational material and raise awareness about these smells [20].

Bibliography

[1] Anonymous. Fix potential github action smells by ceddy4395 - pull request 29273 - dbeaver/dbeaver, 2024. URL https://github.com/dbeaver/dbeaver/pull/29273#discussion_r1584612239.

[2] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[3] Giacomo Benedetti, Luca Verderame, and Alessio Merlo. Automatic security assessment of github actions workflows. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, CCS '22. ACM, November 2022. doi: 10.1145/3560835.3564554. URL http://dx.doi.org/10.1145/3560835.3564554.

[4] Islem Bouzenia and Michael Pradel. Resource usage and optimization opportunities in workflows of github actions. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3623303. URL https://doi.org/10.1145/3597503.3623303.

[5] Tingting Chen, Yang Zhang, Shu Chen, Tao Wang, and Yiwen Wu. Let's supercharge the workflows: An empirical study of github actions. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 01–10, 2021. doi: 10.1109/QRS-C55045.2021.00163.

[6] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for MSR studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*, pages 560–564. IEEE, 2021.

[7] Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. On the use of github actions in software development repositories. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 235–245. IEEE, 2022.

[8] Hassan Onsori Delicheh, Alexandre Decan, and Tom Mens. A preliminary study of github actions dependencies. In *SATToSE*, pages 66–77, 2023.

[9] Paul Duvall, Stephen M. Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*. Addison-Wesley Professional, 2007. ISBN 0321336380.

[10] Paul M. Duvall. Continuous integration: Patterns and antipatterns in the software lifecycle. https://dzone.com/refcardz/continuous-integration, 2010.

[11] Paul M. Duvall. Continuous delivery: Patterns and antipatterns in the software lifecycle. https://dzone.com/refcardz/continuous-delivery-patterns, 2011.

[12] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[13] Martin Fowler. Continuous integration. https://martinfowler.com/articles/continuousIntegration.html. Accessed: 01-07-2023.

[14] Keheliya Gallaba and Shane McIntosh. Use and misuse of continuous integration features: An empirical study of projects that (mis)use travis ci. *IEEE Transactions on Software Engineering*, 46(1):33–50, 2020. doi: 10.1109/TSE.2018.2838131.

[15] Miguel A. García-Pérez. Statistical conclusion validity: Some common threats and simple remedies. *Frontiers in Psychology*, 3, 2012. ISSN 1664-1078. doi: 10.3389/fpsyg.2012.00325. URL https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2012.00325.

[16] Inc. GitHub. About billing for github actions, 2024. URL urhttps://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions. Accessed: 15-05-2024.

[17] Mehdi Golzadeh, Alexandre Decan, and Tom Mens. On the rise and fall of ci services in github. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 662–672, 2022. doi: 10.1109/SANER53432.2022.00084.

[18] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE '16, page 426–437, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450338455. doi: 10.1145/2970276.2970358. URL https://doi.org/10.1145/2970276.2970358.

[19] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, pages 426–437, 2016.

[20] Ali Khatami and Andy Zaidman. Quality assurance awareness in open source software projects on github. In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 174–185, 2023. doi: 10.1109/SCAM 59687.2023.00027.

[21] Timothy Kinsman, Mairieli Wessel, Marco A. Gerosa, and Christoph Treude. How do software developers use github actions to automate their workflows? In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, May 2021. doi: 10.1109/msr52588.2021.00054. URL http://dx.doi.org/10.1109/MSR52588.2021.00054.

[22] Timothy Kinsman, Mairieli Santos Wessel, Marco Aurélio Gerosa, and Christoph Treude. How do software developers use github actions to automate their workflows? In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021*, pages 420–431. IEEE, 2021. doi: 10.1109/MS R52588.2021.00054. URL https://doi.org/10.1109/MSR52588.2021.00054.

[23] Igibek Koishybayev, Aleksandr Nahapetyan, Raima Zachariah, Siddharth Muralee, Bradley Reaves, Alexandros Kapravelos, and Aravind Machiry. Characterizing the security of github CI workflows. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2747–2763, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1. URL https://www.usenix.org/conference/usenixsecurity22/presentation/koishybayev.

[24] Eero I. Laukkanen, Juha Itkonen, and Casper Lassenius. Problems, causes and solutions when adopting continuous delivery - A systematic literature review. *Inf. Softw. Technol.*, 82:55–79, 2017. doi: 10.1016/J.INFSOF.2016.10.001. URL https://doi.org/10.1016/j.infsof.2016.10.001.

[25] Leticia Montalvillo and Oscar Díaz. Tuning github for SPL development: branching models & repository operations for product engineers. In Douglas C. Schmidt, editor, *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 111–120. ACM, 2015. doi: 10.1145/2791060.2791083. URL https://doi.org/10.1145/2791060.2791083.

[26] Dewayne E. Perry, Adam A. Porter, and Lawrence G. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, page 345–355, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132530. doi: 10.1145/336512.336586. URL https://doi.org/10.1145/336512.336586.

[27] Sk Golam Saroar and Maleknaz Nayebi. Developers' perception of github actions: A survey analysis. *arXiv preprint arXiv:2303.04084*, 2023.

[28] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github.

In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 805–816, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336758. doi: 10.1145/2786805.2786850. URL https://doi.org/10.1145/2786805.2786850.

[29] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 805–816, 2015.

[30] Carmine Vassallo, Sebastian Proksch, Harald C Gall, and Massimiliano Di Penta. Automated reporting of anti-patterns and decay in continuous integration. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 105–115. IEEE, 2019.

[31] Carmine Vassallo, Sebastian Proksch, Anna Jancso, Harald C. Gall, and Massimiliano Di Penta. Configuration smells in continuous delivery pipelines: a linter and a six-month study on gitlab. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 327–337, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370431. doi: 10.1145/3368089.3409709. URL https://doi.org/10.1145/3368089.3409709.

[32] Mairieli Wessel, Joseph Vargovich, Marco A Gerosa, and Christoph Treude. Github actions: the impact on the pull request process. *Empirical Software Engineering*, 28 (6):131, 2023.

[33] Cedric Willekens. Replication Package for "Catching Smells in the Act: A GitHub Actions Workflow Investigation" - Thesis, August 2024. URL https://doi.org/10.5281/zenodo.13331530.

[34] Alexey Zagalsky, Joseph Feliciano, Margaret-Anne D. Storey, Yiyun Zhao, and Weiliang Wang. The emergence of github as a collaborative platform for education. In Dan Cosley, Andrea Forte, Luigina Ciolfi, and David McDonald, editors, *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW 2015, Vancouver, BC, Canada, March 14 - 18, 2015*, pages 1906–1917. ACM, 2015. doi: 10.1145/2675133.2675284. URL https://doi.org/10.1145/2675133.2675284.

[35] Fiorella Zampetti, Carmine Vassallo, Sebastiano Panichella, Gerardo Canfora, Harald Gall, and Massimiliano Di Penta. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering*, 25, 03 2020. doi: 10.1007/s10664-019-09785-8.

[36] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. The impact of continuous integration on other software development practices: A large-scale empirical study. In *2017 32nd IEEE/ACM International*

Conference on Automated Software Engineering (ASE), pages 60–71, 2017. doi: 10.1109/ASE.2017.8115619.

[37] T. Zimmermann. Card-sorting: From text to themes. In Tim Menzies, Laurie Williams, and Thomas Zimmermann, editors, *Perspectives on Data Science for Software Engineering*, pages 137–141. Morgan Kaufmann, Boston, 2016. ISBN 978-0-12-804206-9. doi: https://doi.org/10.1016/B978-0-12-804206-9.00027-1. URL https://www.sciencedirect.com/science/article/pii/B9780128042069000271.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

GHA: GitHub Actions - The CI/CD tool created by GitHub

CD: Continious Delivery - A software engineering practice where new functionality is automatically released to end users.

CI: Continious Integration - A software engineering practice where new functionality is automatically evaluated for correctness and quality.

Contribution study: A study where changes to code bases are presented to maintainers and collect their response and feedback on the change.

Job: A configurable list of steps usually used to build, test and lint newly committed code.

PR: Pull Request - Used by developers to allow maintainers of software projects to review their new code before merging it into the project.

Smell: A pattern indicating a potential technical debt.

Workflow: A configurable automated process made up of one or more jobs which can be triggered by events on GitHub.