Catching Smells in the Act: A GitHub Actions Workflow Investigation

Ali Khatami
Delft University of Technology
Delft, The Netherlands
s.khatami@tudelft.nl

Cédric Willekens
Delft University of Technology
Delft, The Netherlands
cedric@willekens.dev

Andy Zaidman
Delft University of Technology
Delft, The Netherlands
a.e.zaidman@tudelft.nl

***Abstract*—GitHub Actions (GHA) are a way to automate CI/CD workflows within the GitHub platform. The deep integration of GHA into GitHub enables to automate a wide range of social and technical activities. In this study, we investigate *workflow smells*, i.e., characteristics in the workflow that possibly indicate a deeper problem. Through a mining study, we first expose a list of frequent change patterns in the workflows of 83 GitHub projects. We then manually analyze these frequent change patterns to understand the negative effects that the frequent changes try to remove. To validate the list of 22 potential workflow smells that we thus obtain, we carry out a contribution study with 32 projects on GitHub through pull requests that contain a fix to the candidate smell. By qualitatively analyzing the maintainers' comments in 32 pull requests, we settle on 7 confirmed GHA workflow smells.**

***Index Terms*—GitHub Actions, Open Source Software (OSS), GitHub, Continuous Integration (CI), Continuous Deployment (CD), GitHub Actions Optimization, GitHub Actions Security, Software Evolution**

I. Introduction

Continuous Integration (CI) is the software engineering practice in which developers not only integrate their work into a shared mainline frequently, but also verify the quality of their contributions continuously [1]. Continuous Deployment (CD) adds frequent delivery through automated deployments [2]. Both practices have become integral to collaborate software development to streamline the software delivery process and increase the quality of the delivered software [3]–[9].

To support CI/CD developers rely on dedicated platforms that provide the necessary infrastructure and tooling. Among the available platforms are Travis CI [10], Circle CI, Cirrus CI, Azure Pipelines, GitLab CI, and GitHub Actions (GHA). GHA enables software engineers to automate workflows directly from within GitHub repositories, attaching triggers to various events like code pushes, pull requests [11], and releases. Since its introduction in November 2019, GHA emerged as the predominant workflow automation tool on GitHub [12]. GHA's integration with the GitHub ecosystem and open-source library of reusable workflow actions have been key contributing factors to its rapid adoption [6].

The growing adoption of GHA has attracted considerable research interest, with numerous studies empirically investigating its adoption [12]–[17]. These studies have highlighted concerns related to security [16], [18]–[21], dependency management [6], [22], and resource usage [23] in GHA workflows.

Zampetti et al. identified bad smells for continuous integration [24], i.e., factors in continuous integration that create barriers or challenges when it comes to setting up, maintaining, or ensuring the high-quality outcome of a CI workflow. Additionally, we know that GHA workflow configurations may degrade in quality and accrue technical debt [25]. As such, given that five years have passed since the introduction of GHA, and a substantial commit history for GHA workflow configurations has accumulated, this study sets out to investigate whether GHA workflows suffer from *workflow smells*.

While earlier research has identified CI/CD workflow smells [24], [26], the unique proposition of GitHub Actions workflows warrants a deeper investigation into GitHub Actions-specific workflow smells. More specifically, the fact that GitHub Actions are more fine-grained in nature than typical CI/CD workflows, their wider range of use case scenarios (e.g., GHA are also applicable to issue management, or code reviewing), and the reusability and composability of GHA workflows may lead to different smells than those previously identified. Additionally, earlier studies have identified issues related to security [18], [21], and opportunities for optimization [23] in GHA workflows, but our investigation is set up to be holistic, also targeting maintainability and efficiency smells.

Our investigation is steered by four research questions.

RQ1 Are there common patterns of frequent changes in GHA workflow configurations?

To answer our first research question, we analyze the evolution of GHA workflow configurations over 10,012 commits in 83 projects, thus aiming to understand whether there are common patterns of frequent changes in GHA workflow configurations.

Next, we inspect the common change patterns in GHA workflow configurations, to determine whether some of these common change patterns could indicate a fix to a problematic workflow configuration. This leads to our second research question:

RQ2 Are frequent change patterns in workflow configurations indicators of *workflow smells*?

Once we have analytically established a set of smells, we turn our attention to their automatic detection in RQ3:

RQ3 Can we automatically detect GHA configuration smells?

Finally, to validate our list of candidate GHA smells, we conduct a contribution study in 32 open source GitHub projects by opening 40 pull requests that contain a fix to the candidate smell. Based on the developer's decision to accept or reject our contribution, and the feedback that the developer's provide, we aim to answer our fourth research question:

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

In summary, this paper makes the following contributions:

- A set of 7 frequently found and validated GitHub Actions workflow smells. One of these smells is completely unique to our study.
- A set of scripts designed to automatically detect these workflow smells.
- A contribution study that provides insight into the relevance and importance of the workflow smells.

II. Background

In their seminal blog post of 2000, Fowler and Foemmel advocate for Continuous Integration (CI) as a mechanism to ensure that integrating new code contributions does not break the build, and an integrated automated test suite to detects errors [27]. In a series of papers, Elazhary et al. have established that many companies follow CI practices, but with quite some variations in how they use CI [28], [29]. Both Vasilescu et al. [5] and Hilton et al. [7] highlight that CI improves productivity, leading to more PRs being processed, accepted and merged, PRs being accepted faster, and projects releasing a new version more often.

GitHub Actions. (GHA) is a continuous integration and continuous delivery (CI/CD) service that is tightly integrated into the GitHub platform [16] and has rapidly gained in popularity [12]. While GitHub Actions provides CI/CD functionality similar to other CI/CD tools, it is unique in that it enables to run workflows for several events that happen in a repository, e.g., code reviewing, communication with developers, verifying licence agreements, and monitoring and fixing dependencies and security vulnerabilities [30].

GHA Workflows. A *GHA workflow* contains one or more *jobs* and can be triggered based on different events in the repository. A *job* contains a list of *steps* which specify a command, for example a bash script or a *action*, reusable code that implements common CI/CD tasks. For *jobs*, the operating system on which they run can be defined as well as a *strategy* which allows the same job to be run with different inputs [23].

Security. Security is one of the five major challenges when automating workflows [20]. Poorly secured CI platforms can lead to code being stolen or injected, as well as bypassing code reviews resulting in potentially malicious code being added to the code base [16]. In order to eliminate security risks, four security properties have been identified for CI including: admittance control, execution control, code control, and access to secrets [21]. Koishybayev et al. investigated these security properties for GHA and found that none of them always hold, noting that these properties can be fixed through proper workflow configuration [21]. Benedetti et al. created a tool to automatically identify security issues in GHA workflows [19].

CI/CD Smells. Previous research has investigated CI smells which affect the performance of the development cycle and CI pipelines. Duvall et al. identified 16 general patterns and anti-patterns for CI/CD [31], [32] Zampetti et al. expanded this catalogue by interviewing experts and analysing Stack Overflow[1] discussions [24]. A total of 79 (of which 44 new) smells were identified in 7 categories and evaluated in terms of relevance by professional developers [24].

Gallaba et al. was the first to define, detect, and remove configuration smells in CI. Firstly, a catalogue of four smells was defined for which they created a tool, *hansel* to detect these instances of smells for Travis CI. Through 49 pull requests with removed smells, they got 36 improved CI configurations accepted by the project maintainers [25].

Vassallo et al. investigated automatically detecting CI anti-patterns [33]. Their tool detects the existence of four relevant anti-patterns, and they found 3,823 instances of these smells accross a set of projects [33]. Vassallo et al. created a similar tool – *CD-Linter* – for GitLab [26]; it focuses on smells that are detectable through configuration files. This resulted in four candidate smells: fake success; retry failure; manual execution; and fuzzy version. Using their tool, 145 issues were opened that addressed these smells. They received a response rate of 74% with 53% of maintainers reacting positively to the smells.

III. Study Set-Up

Our study follows a bottom-up approach towards the identification of GHA workflow smells. In Section IV, we start by identifying common change patterns in workflow configuration files using a mining study (RQ1). Subsequently, in Section V two authors analyze these common change patterns to understand whether there are issues underlying the change patterns that ultimately led to the change. This analysis leads to a set of candidate workflow smells (RQ2). In Section VI we investigate the feasibility and effectiveness of a set of custom-developed workflow detection scripts for these candidate smells (RQ3). Finally, in Section VII we carry out a contribution study in which we file pull requests with fixes to the candidate workflow smells in GitHub projects. In observing the acceptance, rejection, and pull request comments, we aim to validate the candidate smells as recognized and relevant workflow smells (RQ4). Within Section IV to VII, we first describe the methodology that we followed to answer each RQ, after which we describe the results.

IV. RQ1: What are the common patterns of frequent changes in GHA workflow configurations?

RQ1 methodology: To discover potential smells in GitHub Actions (GHA) workflow configurations, we employed an

[1]https://stackoverflow.com

TABLE I: Number of changes in GHA workflow configuration files across projects, categorized by programming language.

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

exploratory approach by analyzing the evolution of the GHA configuration files in open-source software repositories.

We initially selected the 100 most popular repositories on GitHub, based on the number of stars received, using the SEART GitHub search service [34]. We selected the 20 most popular projects for five different programming languages: JavaScript, Java, C#, Python, and TypeScript. This approach ensured a diverse dataset of changes to workflows that are not specific to any particular programming language. For each selected repository, we collected all commits that modified at least one ".yml" or ".yaml" file in the ".github/workflows" directory, if such a directory existed. Repositories without this directory were excluded, resulting in a final dataset of 83 projects. We treated each modified workflow file within a commit as a single change, enabling a granular analysis of workflow evolution and identification of common modification patterns. In total, we collected 10,012 commits for 83 projects. Each commit could contain multiple changes, resulting in a total of 20,037 changes across all projects (Table I).

To analyze and categorize the changes in GHA workflows, we employed an open card sorting approach [35], allows categories to emerge naturally from the data without imposing pre-defined groups [36]. We treated each change as a single code. The second author performed the initial coding and categorization, which was then reviewed and finalized through a discussion between the first two authors, until reaching a negotiated agreement [37]. To accelerate the process, we developed automated scripts based on the observed patterns of changes, which automatically labeled and grouped changes according to the established categories. An additional round of manual verification was conducted to ensure the accuracy and reliability of the categorization. This approach allowed for efficient and accurate categorization while maintaining the integrity of the open card sorting methodology.

RQ1 answer: In our analysis we have identified 64 distinct types of frequent changes made to the GHA configuration files. We categorized these 64 types of changes into 8 higher-level categories, based on their purpose, displayed in Table II. The most prominent high-level pattern *run step configuration* appears in 36.5% of changes, and can be found in 89.2% of the projects. We now discuss the categories of changes.

Run step configuration is the most prevalent pattern, encompassing changes such as run command updates, action configuration, adding or removing run steps, and updating run steps. We observed that developers frequently adjust their run steps to optimize execution, leverage GHA capabilities, and improve the efficiency of their CI/CD processes.

TABLE II: Categories of common changes in the evolution of the GHA YAML configuration files. The complete table with more fine-grained categories of changes is available in the provided dataset [38]. Please note that 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.

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**	

Dependency versioning primarily involves bumping versions and hash versions, as well as using hashes instead of versions. These changes highlight the importance of keeping dependencies and runtime environments up to date and ensuring the reproducibility of builds.

Workflow organization includes adding, removing, moving, or refactoring workflows, and updating workflow names. Developers frequently adjust their workflow structure to possibly better suit their project's needs and improve maintainability.

Trigger conditions, job configuration, environment setup, and scheduling play important roles in the evolution of GHA workflows, with developers adjusting triggers, job requirements, environment variables, and scheduling configurations to ensure proper execution and optimize resource usage.

Job configuration and environment setup involves adjusting job requirements and environment variables to ensure proper execution and optimize resource usage. The most common changes in job configuration include adding or removing jobs, matrix configuration, and updating job names. Environment setup primarily involves updating environment variables and runs-on configurations. These changes indicate that developers frequently adapt their workflows to different environments and job requirements.

RQ1 summary. Our analysis reveals 64 common patterns of frequent changes in GHA workflow configurations, grouped into 8 categories based on their purposes. These common change patterns suggest that developers frequently adjust their GHA configurations to optimize execution, improve maintainability, update dependencies, and ensure proper workflow execution.

V. RQ2: What types of smells exist in GHA workflow configurations?

RQ2 methodology: We hypothesize that some of the frequent change patterns identified in RQ1 encompass *bad practices* that software engineers address in their GHA configurations. The updating of versions, refactoring of workflows, and optimization of run steps, for example, can be seen as efforts to improve the quality and maintainability of GHA configurations by eliminating potential smells. In our context, we consider a smell to be a suboptimal pattern or practice that can negatively affect the maintainability, performance, security, or reliability of the workflow. The frequent change patterns motivated us to delve deeper in changes to identify these potential smells.

We started this by critically examining each labeled change from RQ1 and performed the following steps: (1) We excluded changes that directly and immediately modify the workflow's behavior, such as adding or removing jobs, as these changes are not indicative of configuration smells. (2) We retained changes that do not immediately alter workflow behavior but enhance maintainability and reliability, such as using hashes instead of versions to ensure consistent action versions and mitigate the risk of unexpected behavior changes due to action updates. (3) We grouped related changes together based on their context and purpose, such as adding and updating permissions, which both pertain to access control management in GHA workflows. We further categorized changes based on their specific purposes, like adding conditional statements to prevent execution on forks or for other particular reasons.

For the labeled changes from steps 2 and 3, the second author critically grouped them into named smells based on their purpose and previous research on *GHA security*, *GHA performance/optimization*, and *Other CI/CD smells*. To ensure validity, the first and second authors discussed the grouping and naming of these potential smells, reaching agreement on 35 candidate smells. We then removed smells with only one occurrence and those requiring project contextual knowledge beyond the configuration file (e.g., repository state or available actions). This resulted in 22 potential smells, defined in Table II. The complete list of 35 candidate smells, including removed ones is available in the replication package [38].

RQ2 answer. Our analysis of smells in common patterns of changes in GHA workflows revealed 22 distinct smells, which we categorized into three main groups: security (3 smells), performance/optimization (10 smells), and other CI/CD smells (9 smells). Table III provides an overview of these smells, including the number of fixes for each smell among the 83 analyzed projects, the number of projects in which the smell was fixed by at least one commit, and the supporting evidence from previous research. To trace the fixes to these smells back to the specific changes in GHA configuration commits of these projects, see our study's replication package [38].

Security smells were identified based on changes addressing potential vulnerabilities. Table III lists 3 smells in this category, their description, and their frequency of occurrence in projects. These smells can lead to unauthorized access, code injection, and other security breaches. Fixing these smells can reduce the attack surface and ensures more secure workflows. The identified security smells are backed by evidence from previous studies [16], [20], [21].

Performance/optimization smells were identified based on changes addressing resource usage and limitations. In Table III we observe 10 smells with this type, including their descriptions. These smells can lead to inefficient resource usage, longer build times, and increased costs; addressing them may result in optimizations. A recent study on resource usage and limitations in GHA supports most of the identified performance/optimization smells [23]. While *Smell 10 and 11* were not mentioned in that study, GitHub documentation provides rationale for addressing them to optimize storage usage [39].

Other CI/CD smells were identified based on changes addressing bad practices, configuration smells, and the need for pipeline restructuring. Table III provides more details on the 9 instances of this type of smell. These smells can lead to maintainability issues, inconsistent build results, and difficulty in understanding and debugging workflows. Fixing these smells can improve readability, maintainability, and reliability, making it easier for team members to understand and modify workflows. The identified CI/CD smells are backed by research on bad practices in CI/CD [24], [26], [31], [32].

RQ2 summary. Our analysis of common change patterns in GHA workflow configurations revealed 22 distinct smells, which we categorized into three main groups: security smells (3 instances), performance/optimization smells (10 instances), and other CI/CD smells (9 instances). These smells were identified based on changes addressing potential vulnerabilities, resource usage and limitations, and bad practices or configuration issues. The identified smells are backed by evidence from previous research on GHA security, performance, and CI/CD.

VI. RQ3: Can we automatically detect GHA configuration smells?

RQ3 methodology: To facilitate the automated detection of the smells identified in RQ2, we developed a suite of Python scripts. These scripts can detect lines in GHA configuration files that contain smells. We provide our Python scripts in our replication package [38].

TABLE III: Identified fixed smells in the history of GHA configuration files changes.

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	[21]
	2	**Use commit hash instead of tags for action versions** Problem: Tags can be modified, leading to inconsistent behavior. Solution: `uses: actions/checkout@<commit-sha>`	8	11	[16], [20]
	3	**Set permissions for GitHub Token** Problem: Default token has overly permissive access. Solution: Set permissions under `permissions` key.	4	8	[21]
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	[23]
	5	**Avoid jobs without timeouts** Problem: Can run indefinitely, wasting resources and blocking workflows. Solution: Set timeout for jobs.	11	14	[23]
	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	[23]
	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	[23]
	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	[23]
	9	**Avoid executing scheduled workflows on forks** Problem: Inefficient resource usage for inactive forks. Solution: Add condition checking repository owner.	6	10	[23]
	10	**Avoid uploading artifacts on forks** Problem: Inefficient resource usage. Solution: Add condition checking repository owner.	5	6	[39]
	11	**Use 'if' for upload-artifact action** Problem: Unnecessary uploads waste resources and storage. Solution: Add condition to run only when needed.	4	5	[39]
	12	**Avoid deploying jobs on forks** Problem: Inefficient resource usage due to unnecessary deployments. Solution: Add condition checking repository owner.	2	2	[23]
	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	[23]
Other CI/CD Smells	14	**Correct indentation** Problem: Incorrect indentation reduce readability. Solution: Use YAML linter to ensure consistent indentation.	25	57	[31], [32]
	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	[24], [26]
	16	**Name run steps** Problem: Unnamed steps reduce readability and debugging. Solution: Use descriptive names for steps.	11	12	[31], [32]
	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	[24]
	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	[31], [32]
	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	[24]
	20	**Specify package versions** Problem: Unspecified versions can lead to non-reproducible builds. Solution: Specify exact package versions in install commands.	3	5	[26]
	21	**Add comments to workflows** Problem: Lack of documentation reduces maintainability. Solution: Add comments explaining purpose and function.	3	3	[31]
	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	[24]

To select popular and active projects, we started with the 100 most popular projects for each of the five programming languages used in RQ1 (JavaScript, Java, C#, Python, and TypeScript). We then applied additional criteria: (1) having GHA workflows, and (2) not in our dataset of 83 projects for RQ1. We ordered them by the number of pull requests merged in the last 30 days and selected the top 40 projects, thus ensuring these projects are in active development. To evaluate the effectiveness of our automated smell detection scripts, we curated a new dataset comprising 119 GHA configuration files from these 40 projects.

We first established a ground truth by manually analyzing the configuration files and identifying smells. This step took 1 person 2 days. We then executed our smell detec-

TABLE IV: Evaluation of automated detection of GHA smells.

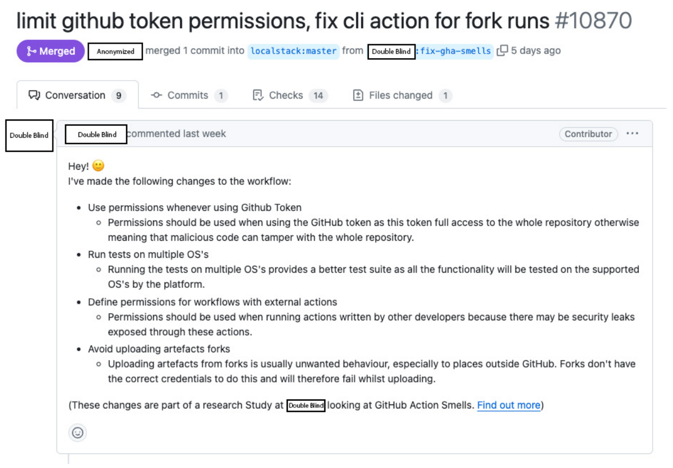
Smell #	Ground truth	FP	FN	Recall	Precision	F1-score
1	59	0	0	100%	100%	1
2	88	0	1	98.9%	100%	0.99
3	17	0	0	100%	100%	1
4	27	2	6	85.7%	92.6%	0.86
5	101	0	0	100%	100%	1
6	38	3	5	87.5%	92.1%	0.90
7	37	1	1	97.3%	97.3%	0.97
8	33	4	7	80.6	87.9%	0.84
9	16	0	3	84.2%	100%	0.91
10	18	9	2	81.8%	50%	0.62
11	13	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

tors on the dataset and recorded false positives and false negatives. Recall, precision, and F1-scores were calculated for each of the 22 smell detectors 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, FP, and FN represent true positives, false positives, and false negatives.

RQ3 Answer. The results in Table IV demonstrate the overall effectiveness of our automated smell detection scripts. Notably, 8 out of the 22 smell detectors (36.4%) achieved perfect precision, recall, and F1-scores of 100%. Furthermore, the median recall, precision, and F1-score across all smell detectors were 91.8%, 100%, and 0.93, respectively, signalling the overall good performance of our automated detection approach.

The detectors for smells 12, 19, 20 and 22 exhibited relatively low recall values of 19%, 40%, 42.9%, 52.6% respectively. These specific detectors struggle, because more contextual understanding is required to identify all instances of these specific smells. Detecting workflows that deploy the project (*Smell 12*) or locating where test suites are run (*Smell 19* and *22*) proved non-trivial. The detectors, focused on a single workflow, missed instances across multiple workflows. *Smell 20* also received a lower recall score because not all install commands support version specification, which the detector relied upon.

RQ3 summary. Our evaluation demonstrates the feasibility and effectiveness of automatically detecting GHA configuration smells using custom-developed Python scripts. While the majority of our smell detectors exhibited high precision, recall, and F1-scores, for 4 out of 22 smells our detectors exhibit low recall.


limit github token permissions, fix cli action for fork runs #10870

Merged Anonymized merged 1 commit into `localstack:master` from Double Blind`:fix-gha-smells` 5 days ago

Conversation 9 · Commits 1 · Checks 14 · Files changed 1

Double Blind

Double Blind commented last week — Contributor

Hey! 🙂
I've made the following changes to the workflow:

- Use permissions whenever using Github Token
  - Permissions should be used when using the GitHub token as this token full access to the whole repository otherwise meaning that malicious code can tamper with the whole repository.
- Run tests on multiple OS's
  - Running the tests on multiple OS's provides a better test suite as all the functionality will be tested on the supported OS's by the platform.
- Define permissions for workflows with external actions
  - Permissions should be used when running actions written by other developers because there may be security leaks exposed through these actions.
- Avoid uploading artefacts forks
  - Uploading artefacts from forks is usually unwanted behaviour, especially to places outside GitHub. Forks don't have the correct credentials to do this and will therefore fail whilst uploading.

(These changes are part of a research Study at Double Blind looking at GitHub Action Smells. Find out more)

Fig. 1: An example of the submitted pull request, #35.

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

RQ4 methodology: To externally validate and assess the practical relevance of the identified GHA workflow smells, we conducted an open-source contribution study [40]: we created pull requests (PRs) to popular and actively maintained GitHub repositories, addressing the smells detected in their GHA workflow configurations. By submitting these pull requests, we aimed to gauge the receptiveness of the open-source community to our proposed changes and gather insights into the smells that we observed in RQ2. We acquired approval from the Human Research Ethics Council of our university and followed their guidelines throughout the study.

For our contribution study, we selected 40 projects on GitHub using the same criteria detailed in RQ3. For each of these projects we ran our tool discussed in RQ3 on their GHA workflows to identify their smells. Subsequently, we strategically prepared PRs by addressing multiple smells per PR, to ensure that each smell received at least one reaction across all submissions. The PRs included explanations of the smells they aimed to fix (see Figure 1). To ensure transparency, we explicitly mentioned that the contribution was part of a research study and provided a link to a separate web page containing further information about the study.

Table VI displays the details of the submitted PRs, including their IDs, associated GitHub projects, status, and the fixed smells. The order of projects is based on the time of opening their associated PR. We submitted the PRs in April-May 2024, ensuring that maintainers of the projects had at least 3 weeks to provide feedback. In total, out of the 40 submitted PRs, 32 received at least one response at the time of writing this paper, either in the form of a comment, or a decision regarding merging or closing the PR. The responses to the proposed fixes for the smells are as follows: 46 accepted smells '✓', 34 rejected smells 'X', and 37 pending final decisions ' O '. At the time of writing this paper, 15 PRs were merged, 8

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

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

closed, and 17 pending. Among the pending PRs, 9 received comments and are included in Table VI.

The analysis focused on the 165 comments received on our pull requests (PRs). We used an open card sorting approach [35], which allows categories to emerge naturally from the data without imposing pre-defined groups [36]. Initially, the first and second authors independently coded 20% of the comments each. After agreeing on the codes, the second author proceeded to code the remaining comments. The codes were then sorted using sticky notes on a Miro board[2] for each smell, taking into account the state of the PR related to that smell (merged, pending, or closed). The first and second authors collaboratively performed the categorization by manually sorting the codes into groups during a joint session. To ensure reliability and validity, they conducted a discussion meeting to review the codes and categories until reaching a negotiated agreement [37].

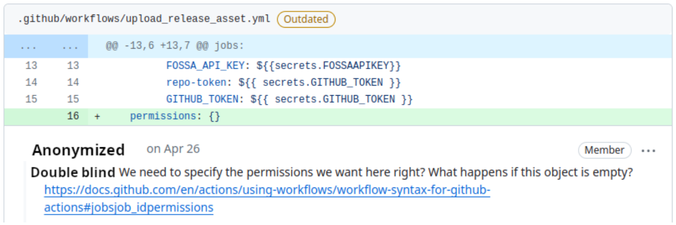
RQ4 answer: The qualitative analysis of the comments received on the submitted pull requests adds to quantitative data on accepted and rejected pull requests; it enables us to better understand how maintainers perceive the identified GHA configuration smells, and the proposed fixes.

Our qualitative analysis revealed seven general types of comments: (1) **Clarification Question**: comments asking for further information or explanations regarding the proposed changes; (2) **Suggested Edit**: comments providing suggestions to improve the submitted PR; (3) **Feedback**: comments expressing opinions or concerns about the proposed fixes for the smells, or discussing the judgments of the PR; (4) **Automated messages**: Comments from software bots [41] that automate tasks or report to facilitate contribution; (5) **Decision-related comments**: comments regarding the decision to accept or reject the contribution; (6) **Appreciation**: comments expressing gratitude for the submitted contribution; and (7) **Unrelated comments**: comments not directly related to the proposed changes, e.g., mentioning other maintainers to review the PR.

We focus on the first three types of comments received on PRs related to each smell, as they provide relevant information about how maintainers perceived our identification and fixes of the GHA configuration smells. The details of this qualitative analysis are available in our replication package [38].

Clarification Question: were related to the purpose and impact of the proposed changes to fix a smell (*Smell 1, 2, 3, 4, 6, 11, 12, 13, 15, 19,* and *20*). Maintainers sought to better understand how fixing the smells will impact their workflow. Figure 2 shows an example related to *Smell 1*, where the developer asks questions to better understand how permission configuration of the workflow works.

In 5 cases of a smell fix, our PRs were merged after addressing clarification questions, while in 7 cases, the proposed fixes were rejected (see Table V). The reasons for rejection varied. For *PR 29273*, maintainers stated that they would consider an alternative solution for *Smell 2*, but since the workflow's task was not critical, they decided against making changes. In *PR 28909*, a maintainer expressed concerns about the trade-offs of fixing *Smells 11, 15, and 19*, as modifying the functioning workflow could lead to unintended consequences, potentially compromising its stability. Another maintainer in the same PR argued that addressing *Smells 4 and 12* would introduce a trade-off between maintainability and resource optimization. Fixing these smells would require adding more "if" statements to prevent running the workflow on forks, which would optimize resource usage. However, this optimization comes at the cost of reduced maintainability, as the additional conditional statements make the workflow harder to understand and modify. We see indications that the maintainers tend to prioritize maintainability over resource usage optimization of their project's forks. Lastly, in *PR 23965*, the attempt to fix *Smell 19* by running tests on multiple operating systems using a matrix strategy led to a trade-off between test coverage and developer productivity. The increased number of jobs resulted in a higher occurrence of false positives due to flaky tests, requiring developers to manually verify more test cases. Consequently, the expanded testing effort reduced developer productivity, as they spent more time on testing-related tasks rather than other aspects of development.



```
.github/workflows/upload_release_asset.yml
@@ -13,6 +13,7 @@ jobs:
          FOSSA_API_KEY: ${{secrets.FOSSAAPIKEY}}
          repo-token: ${{ secrets.GITHUB_TOKEN }}
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
+     permissions: {}
```

Anonymized on Apr 26

Double blind We need to specify the permissions we want here right? What happens if this object is empty? https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#jobsjob_idpermissions

Fig. 2: Example of a comment asking a clarification questions.

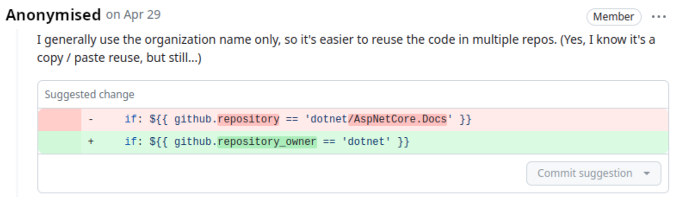
Suggested Edits: maintainers suggested edits for 10 of the smells, falling into two categories: **1) alternative smell fixes** and **2) typographical corrections**. In terms of alternative smell fixes (*Smell 4, 9, 10,* and *12*), developers proposed different approaches to address these smells compared to our original fixes, aiming to enhance workflow reusability and maintainability (Figure 3). These alternative solutions can provide insights into resolving smells while aligning with project-specific needs. Secondly, maintainers identified and suggested typographical corrections for our our submitted PRs, such as incorrect capitalization. The suggested edits confirm the maintainers' receptiveness to accepting smell fixes, as

[2]https://miro.com/

TABLE VI: List of repositories that received at least a response from the maintainers. The complete list of opened PRs is available in our replication package [38].

#	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

confirmed by Table II, which shows that no PRs receiving edit suggestions were closed.


Anonymised on Apr 29 Member

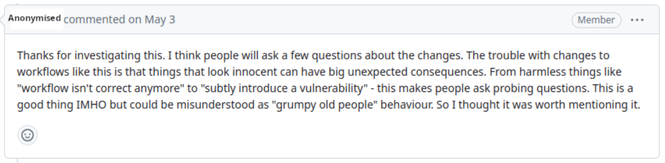
I generally use the organization name only, so it's easier to reuse the code in multiple repos. (Yes, I know it's a copy / paste reuse, but still...)

Suggested change

```
-    if: ${{ github.repository == 'dotnet/AspNetCore.Docs' }}
+    if: ${{ github.repository_owner == 'dotnet' }}
```

Fig. 3: Developer providing a different fix to Smell 4.

Feedback: maintainers provided three categories of feedback: **1) positive feedback on the impact of fixing the smell** (*Smell 3* and *7*): maintainers acknowledged that fixing *Smell 3* improves the developer experience, and for *Smell 7*, they confirmed that the workflow functions as expected after the changes. **2) skepticism about the value of the contribution** (*Smell 1, 2, 4, 6, 10, 11, 12, 14, 15, 17, 20, 21*): maintainers expressed doubts about the usefulness of the proposed fixes, finding the purpose of the PRs unclear or questioning the necessity of modifying functioning workflows (Figure 4). In *PR 32326*, *Smell 14 and 21* were rejected because the maintainers found the explanations regarding the smells unclear. Moreover, *Smell 4, 10 and 12* received conflicting feedback in two PRs, where maintainers were uncertain about GitHub's policy on running workflows on forks, leading them to question the relevance of the smells. Ultimately, *PR 2097* was merged, while *PR 28909* was closed, with the maintainers' doubts about GitHub's policy about running project's workflows on its forks being the primary reason for their skepticism. Additionally, *PR 28909*, which aimed to fix *Smell 11, 12, 15, and 20*, was closed due to the developers' concerns about potential side effects. However, since at least one other PR addressing these smells was merged, we believe that a more thorough explanation of the smells could have helped alleviate the maintainers' skepticism. Moreover, reporting a calculation of the impact of smell fixes along with the PR, similar to what Durieux has done in a contribution study of Docker smells [42], can address this skepticism about fixing GHA smells in future studies. Two PRs addressing *Smell 1, 2, 15, 16, and 18* also received feedback indicating that the maintainers preferred discussing changes to workflows before submitting PRs. In these cases, the maintainers' reluctance to accept external contributions that change infrastructure-related aspects of the project, such as GHA workflows, was evident. Overall, the skepticism regarding the value of the proposed smell fixes appears to stem from two main factors: first, maintainers' uncertainty about the relevance and necessity of the fixes, particularly for functioning workflows, highlighting the importance of clear and comprehensive explanations when proposing changes. Second, the culture of open-source projects, which tends to be cautious about accepting external contributions that modify infrastructure-related aspects, such as GHA workflows, without prior discussion [*"appreciate the contribution but please in the future discuss changes to infrastructure...", PR 69915*]. **3) explanations for why the smell is not applicable** (*Smell 2, 4, 9, 10, 12, 15, 17, 19, 22*): maintainers indicated that the workflows in question do not perform critical tasks (*Smell 2, 15, 17, 19, 22*), or expressed deliberately keeping upload artifacts actions on forks because they need it (*Smell 10 and 11*), [*"we want to upload artifacts because we need them later, also forks will or might need them, I see no reason to add a condition for this.", PR 29164*].


Anonymised commented on May 3

Member

Thanks for investigating this. I think people will ask a few questions about the changes. The trouble with changes to workflows like this is that things that look innocent can have big unexpected consequences. From harmless things like "workflow isn't correct anymore" to "subtly introduce a vulnerability" - this makes people ask probing questions. This is a good thing IMHO but could be misunderstood as "grumpy old people" behaviour. So I thought it was worth mentioning it.

Fig. 4: An example of maintainers skeptical about changing the workflow.

To answer RQ4, we analyzed the reactions received for each smell (X and ✓ in Table VI) and identified three groups: 1) *mostly accepted smells*, which received at least 2 acceptances and at most 1 rejection, indicating maintainers' receptiveness to fixing these smells in their workflows; 2) *mostly rejected smells*, which had more rejections than acceptances, with at least two non-pending PRs, suggesting that developers were generally not open to accepting the proposed fixes; and 3) *smells with mixed opinions or insufficient responses*, which received both positive and negative reactions or had only one non-pending PR, indicating no clear consensus regarding the importance or relevance of these smells.

Mostly accepted: 7/22 smells are considered mostly accepted (*Smell 3, 5, 6, 7, 8, 9*, and *10*) and were merged in 14 projects (Table VI). Out of these smells, 3 received no rejections (*Smell 3, 7*, and *9*), while 4 received 1 rejection each (*Smell 5, 6, 8*, and *10*). *Smell 5* was rejected in *PR 69915* because of project's culture of not accepting external contribution to infra-structure related aspects of the project. Similarly, *Smell 6* was rejected in *PR 29273* through feedback indicating that the maintainer did not consider the contribution valuable in case of their simple workflow. *Smell 8* and *10* were rejected in *PR 29164* due to the existing workflow logic already handling the skipping of runs based on modified code areas, and the maintainer mentioned that artifacts are needed by other workflows being run on forks, aligning with the category of "explanations for why the smell is not applicable." While a few rejections occurred, e.g., due to project-specific factors, the overall positive receptiveness of developers regarding these smells highlights their recognition of the value in fixing them.

Mostly rejected: 6/22 smells were mostly rejected: *Smell 2, 14, 15, 17, 18*, and *19*. *Smell 14*, and *19* were rejected in all PRs, with no acceptance. *Smell 15* and *19* had the most rejections (4 each), suggesting developers were not receptive to the proposed changes. For *Smell 14* and *15*, maintainers showed "skepticism about the value of the contribution". *Smell 19* was rejected because maintainers "did not find the smell applicable to their project". These findings highlight that certain smells were rejected due to maintainers skepticism about the value of the contribution or its relevance to their projects, or project-specific reasons making the smells inapplicable.

Mixed opinions/inconclusive: 9/22 smells received mixed opinions or insufficient responses: *Smell 1, 4, 11, 12, 13, 16, 20, 21*, and *22*. *Smell 4, 16*, and *20* have an equal number of rejections and acceptances, preventing strong conclusions about their perception by developers. *Smell 1, 11*, and *12* have 3, 4, and 5 accepted PRs, respectively, but were also rejected twice due to developers showing "skepticism about the value of the contribution." Despite the higher acceptance rate, the presence of multiple rejections prevents us from confidently stating that developers were receptive about these smells. *Smell 13, 21*, and *22* received only one response each, with the remaining PRs pending, providing insufficient data to draw conclusions about their perception by developers. In summary, the mixed opinions and limited responses for these smells necessitate further investigation.

RQ4 summary. While there was a consensus among maintainers regarding the relevance and value of fixing 7/22 smells (*Smell 3, 5, 6, 7, 8, 9*, and *10*), as indicated by their acceptance, 6/22 smells (*Smell 2, 14, 15, 17, 18*, and *19*), were mostly rejected, suggesting a lack of consensus on the applicability or importance of the proposed fixes for these smells in their projects. Additionally, there were 9/22 smells (*Smell 1, 4, 11, 12, 13, 16, 20, 21*, and *22*) that received mixed opinions or insufficient responses, preventing a clear consensus about their perception by developers.

VIII. Discussion

This study provides insight into the existence and relevance of **seven GHA workflow smells**, which have been validated by open-source developers through our contribution study. The novelty of this work lies in the bottom-up approach that we have taken to identify these seven workflow smells: from identifying frequent change patterns in GitHub Actions workflows, to analyzing the negative side effects that are alleviated through the change patterns and thus establishing a set of candidate workflow smells, to validating the candidate smells through a contribution study. Through this process, we were also able to identify a new smell, namely: *Smell 10: "Avoid uploading artifacts on forks"*.

The identification of *Smell 10* is a key contribution of this study, as it highlights the resource usage optimization with avoiding uploading artifacts on forked repositories. The validation of this smell through open-source contributions, demonstrates its relevance and importance in the context of GHA workflow optimization.

The other six smells identified in this study were previously discussed in the literature, with five of them (*Smell 5, 6, 7, 8, and 9*) presented as optimization techniques, and one (*Smell 3*) flagged as a security concern (Table IV). However, prior to our study, there was a lack of evidence regarding developers' perceptions and willingness to accept fixes for these smells. Our contribution study fills this gap by providing valuable insights into how developers view these smells and their readiness to accept the PRs fixing them. The acceptance of fixes by developers for all seven smells, including the newly introduced *Smell 10*, suggests that addressing these issues is considered important and beneficial by the maintainers of the projects that we worked with.

We developed automated smell detectors for the GHA configuration smells. The evaluation of these detectors showed promising results, with the majority exhibiting high precision, recall, and F1-scores (0.84 - 1). However, *Smell 10* had a lower F1-score (0.62), as our detector only considered two common artifact upload actions on GHA. Future work should analyze more upload actions to improve the detector's F1-score.

The rejection and mixed opinion of the maintainers of the projects in our contribution study other 15 candidate workflow smells, makes us less confident about their relevance. In future research, we intend to expand the contribution study and set up interviews with maintainers to better understand the relevance of these smells in different project contexts.

A. Implications & Future Work

The findings of this study have several implications for researchers, tool developers, and practitioners working with GitHub Actions:

Awareness of GHA workflow smells: the identified smells provide a foundation for understanding common issues in GHA configurations. Researchers and practitioners can leverage this knowledge to develop best practices and create educational materials that raise awareness [43], [44] about these smells, ultimately helping developers prevent, identify, and address them in their GHA workflows.

Tool support for smell detection and refactoring: the automated smell detectors developed in this study can be integrated into existing GHA development tools or integrated development environments (IDEs) to provide real-time feedback and suggestions for improving workflow configurations [45]–[47].

Empirical studies on workflow smells: for some candidate workflow smells, we received mixed opinions on their relevance, which could be explained by project-specific circumstances. Further study is needed to better understand how context influences the perception and importance of smells.

B. Threats to Validity

This section discusses potential threats to the validity of our study and the steps taken to mitigate them [48]:

External validity: the identified smells and their validation are based on a sample of 83 and 40 GitHub projects, respectively, which may not be representative of all GHA workflows. To mitigate this threat, we selected popular, active projects with diverse programming languages, sizes, and domains.

Internal validity: the identification of workflow smells and the development of automated detectors relied on the authors' expertise and interpretation of the frequent change patterns which formed the internal validation of the potential smells. To mitigate potential biases, two authors independently reviewed the change patterns and discussed any disagreements until a consensus was reached. Additionally, we associated the identified potential smells with other studies on GHA and CI/CD to mitigate our own bias.

Conclusion validity: the conclusions drawn from the contribution study (RQ4) are based on a limited number of PRs and subjective opinions, potentially affecting generalizability. Many candidate smells received mixed or insufficient feedback due to the limited number of contributions. Future larger-scale studies, or survey/interview studies with more developers could strengthen the findings.

We followed a rigorous methodology and thoroughly documented the source code, scripts, and procedures to ensure replicability, enhancing transparency and reliability. We provide an replication package with all our materials [38].

IX. Conclusion

This study provides insights into the existence and relevance of seven GitHub Actions (GHA) workflow smells. Employing a bottom-up approach, by analyzing frequent change patterns in 10,012 commits from 83 projects, we identified 64 change patterns grouped into 8 categories (RQ1).

From these frequent change patterns, we identified and defined a candidate list of of 22 potential GHA workflow smells (RQ2). The external validation confirmed the relevance of six previously discussed smells and identified a new smell, *Smell 10: "Avoid uploading artifacts on forks"*, highlighting optimization of resource usage. Custom smell detectors (RQ3) showed promising results, with F1-scores ranging from 0.84 to 1 for most smells, except for *Smell 10* with a lower recall due to considering limited actions for uploading artifacts in out detector. The contribution study (RQ4), involving 32 pull requests, provided insights into developers' perceptions and willingness to accept fixes, which enabled us to validate 7 out of the 22 candidate GHA workflow smells as actual smells.

The insights of this study can inform future research and tool development efforts in GHA workflow optimization and maintenance, supporting the growing adoption of GHA in the GitHub software development ecosystem.

Acknowledgment

This research was partially funded by the Dutch science foundation NWO through the Vici "TestShift" grant (No. VI.C.182.032).

References

[1] M. Beller, G. Gousios, and A. Zaidman, "Oops, my tests broke the build: an explorative analysis of Travis CI with GitHub," in *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*. IEEE Computer Society, 2017, pp. 356–367.

[2] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Software*, vol. 32, no. 2, pp. 50–54, 2015.

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

[4] B. Fitzgerald and K. Stol, "Continuous software engineering: A roadmap and agenda," *J. Syst. Softw.*, vol. 123, pp. 176–189, 2017.

[5] B. Vasilescu, Y. Yu, H. Wang, P. T. Devanbu, and V. 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)*. ACM, 2015, pp. 805–816.

[6] H. O. Delicheh, A. Decan, and T. Mens, "A preliminary study of GitHub Actions dependencies," in *Proceedings of the 15th Seminar on Advanced Techniques & Tools for Software Evolution*, ser. CEUR Workshop Proceedings, vol. 3483. CEUR-WS.org, 2023, pp. 66–77. [Online]. Available: https://ceur-ws.org/Vol-3483/paper7.pdf

[7] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. 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)*. ACM, 2016, pp. 426–437.

[8] A. Khatami and A. Zaidman, "State-of-the-practice in quality assurance in Java-based open source software development," *Software: Practice and Experience*, vol. 54, no. 8, pp. 1408–1446, 2024.

[9] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella, "A tale of CI build failures: An open source and a financial organization perspective," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 183–193.

[10] M. Beller, G. Gousios, and A. Zaidman, "TravisTorrent: synthesizing Travis CI and GitHub for full-stack research on continuous integration," in *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*. IEEE Computer Society, 2017, pp. 447–450.

[11] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *37th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2015, pp. 358–368.

[12] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of CI services in github," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 662–672.

[13] T. Chen, Y. Zhang, S. Chen, T. Wang, and Y. Wu, "Let's supercharge the workflows: An empirical study of github actions," in *21st IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2021, pp. 1–10.

[14] J. Ayala and J. Garcia, "An empirical study on workflows and security policies in popular github repositories," in *1st IEEE/ACM International Workshop on Software Vulnerability, SVM@ICSE*. IEEE, 2023, pp. 6–9.

[15] T. Kinsman, M. S. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use GitHub Actions to automate their workflows?" in *18th IEEE/ACM International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 420–431.

[16] A. Decan, T. Mens, P. R. Mazrae, and M. Golzadeh, "On the use of GitHub Actions in software development repositories," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 235–245.

[17] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude, "GitHub Actions: The impact on the pull request process," *Empir. Softw. Eng.*, vol. 28, no. 6, p. 131, 2023.

[18] H. O. Delicheh, A. Decan, and T. Mens, "Quantifying security issues in reusable JavaScript actions in GitHub workflows," in *21st International Conference on Mining Software Repositories (MSR)*. ACM, 2024.

[19] G. Benedetti, L. Verderame, and A. Merlo, "Automatic security assessment of GitHub Actions workflows," in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022, pp. 37–45.

[20] S. G. Saroar and M. Nayebi, "Developers' perception of github actions: A survey analysis," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 2023, pp. 121–130.

[21] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry, "Characterizing the security of Github CI workflows," in *31st USENIX Security Symposium, USENIX Security 2022*. USENIX Association, 2022, pp. 2747–2763.

[22] H. O. Delicheh and T. Mens, "Mitigating security issues in github actions," 2024.

[23] I. Bouzenia and M. Pradel, "Resource usage and optimization opportunities in workflows of github actions," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, 2024, pp. 25:1–25:12.

[24] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. C. Gall, and M. Di Penta, "An empirical characterization of bad practices in continuous integration," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1095–1135, 2020.

[25] K. Gallaba and S. McIntosh, "Use and misuse of continuous integration features: An empirical study of projects that (mis)use travis CI," *IEEE Trans. Software Eng.*, vol. 46, no. 1, pp. 33–50, 2020.

[26] C. Vassallo, S. Proksch, A. Jancso, H. C. Gall, and M. Di Penta, "Configuration smells in continuous delivery pipelines: a linter and a six-month study on gitlab," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2020, pp. 327–337.

[27] M. Fowler and M. Foemmel, "Continuous integration," 2000, last accessed 18 June 2024. [Online]. Available: https://martinfowler.com/articles/originalContinuousIntegration.html

[28] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M.-A. Storey, "Uncovering the benefits and challenges of continuous integration practices," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2570–2583, 2022.

[29] O. Elazhary, M. D. Storey, N. A. Ernst, and A. Zaidman, "Do as I do, not as I say: Do contribution guidelines match the GitHub contribution process?" in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 286–290.

[30] A. Decan, T. Mens, and H. O. Delicheh, "On the outdatedness of workflows in the github actions ecosystem," *J. Syst. Softw.*, vol. 206, p. 111827, 2023.

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

[32] P. M. Duvall and M. Olson, "Continuous delivery: Patterns and antipatterns in the software life cycle," *DZone refcard*, vol. 145, p. 64, 2011.

[33] C. Vassallo, S. Proksch, H. C. Gall, and M. Di Penta, "Automated reporting of anti-patterns and decay in continuous integration," in *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE / ACM, 2019, pp. 105–115.

[34] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*. IEEE, 2021, pp. 560–564.

[35] T. Zimmermann, "Card-sorting: From text to themes," in *Perspectives on data science for software engineering*. Elsevier, 2016, pp. 137–141.

[36] J. M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[37] D. Garrison, M. Cleveland-Innes, M. Koole, and J. Kappelman, "Revisiting methodological issues in transcript analysis: Negotiated coding and reliability," *The Internet and Higher Education*, vol. 9, no. 1, pp. 1–8, 2006.

[38] A. Khatami, C. Willekens, and A. Zaidman, "Replication Package for "Catching Smells in the Act: A GitHub Actions Workflow Investigation" (SCAM 2024)," Jun. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.12207164

[39] GitHub. (2024) About billing for GitHub Actions. Accessed: 15-05-2024. [Online]. Available: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions

[40] C. E. Brandt, A. Khatami, M. Wessel, and A. Zaidman, "Shaken, not stirred: How developers like their amplified tests," *IEEE Trans. Software Eng.*, vol. 50, no. 5, pp. 1264–1280, 2024.

[41] E. Shihab, S. Wagner, M. A. Gerosa, M. Wessel, and J. Cabot, "The present and future of bots in software engineering," *IEEE Software*, vol. 39, no. 5, pp. 28–31, 2022.

[42] T. Durieux, "Empirical study of the docker smells impact on the image size," *arXiv preprint arXiv:2312.13888*, 2023.

[43] A. Khatami and A. Zaidman, "Quality assurance awareness in open source software projects on GitHub," in *23rd IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2023, pp. 174–185.

[44] A. Khatami, C. Brandt, and A. Zaidman, "Software quality assurance analytics: Enabling software engineers to reflect on QA practices," in *24th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2024.

[45] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2016, pp. 470–481.

[46] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1419–1457, 2020.

[47] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall, "Context is king: The developer perspective on the usage of static analysis tools," in *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 38–49.

[48] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.