Towards an Empirical Analysis of Code Cloning and Code Reuse in CI/CD Ecosystems

Guillaume Cardoen[1]

[1]*Software Engineering Lab, University of Mons, Belgium*

Abstract

Large open source projects are engaged in collaborative software development through social coding platforms, and use CI/CD practices to automate numerous repetitive tasks through workflows. Most CI/CD tools follow the configuration-as-code paradigm, specifying their workflow configurations as runnable workflow files. We posit that, just as is the case when maintaining regular source code, workflow configuration files are subject to the good and bad practices of reusability and cloning.

This paper provides the plan of my doctoral research, explaining the objectives and research questions, and outlining the research method to reach these objectives. My research focuses on the empirical analysis of how code reuse and code cloning practices emerge and evolve in workflow files. The initial focus is on GitHub, taking GitHub Actions as a case study, given that it is by far the most popular CI/CD used in GitHub.

Keywords

configuration as code, empirical analysis, code cloning, code reuse, collaborative software development, GitHub

1. Introduction

Contemporary collaborative software development relies on a plethora of tools that support or streamline the development process such as version control systems, issue trackers, CI/CD and workflow automation tools, quality checkers, security scanners and many more. Such tools tend to be integrated into collaborative social coding platforms. GitHub [1] is the largest social coding platform today with 100M+ users and hosting 284M+ public projects in 2023 [2]. GitLab [3] is another very popular social coding platform with an estimation of over 30 millions registered users [4].

Continuous integration and delivery (CI/CD) is a pillar for collaborative software development [5], allowing to automate the numerous repetitive tasks being performed by development teams (such as license agreements, automatic answers, code reviewing, quality analysis, test coverage, automatic issue triaging, ...). Many CI/CD services have emerged throughout the years (e.g., Travis, Jenkins, CircleCI, AppVeyor), and other CI/CD services have been directly integrated into social coding platforms (e.g. GitLab CI/CD since September 2015 [3] and GitHub Actions since November 2019 [6]).

Most of today's CI/CD tools rely on *configuration files* (called *workflows* in GitHub Actions and *pipelines* in GitLab CI/CD) to specify automated *jobs*. These jobs can be executed on *runners* provided by the social coding platform (or through dedicated user-provided runners), based on *triggers* that are either manual or event-based (such as a pull request, commit, release, recurrent schedule, signal from an external service, and many more).

Given that CI/CD configuration files can be "executed", they can be seen as an instance of the so-called *configuration-as-code* practice [7]. Considering configuration files as code opens up an entire new spectrum of research opportunities, since it allows to empirically observe and study the same coding practices as those that have been studied for several decades for traditional programming languages.

One of these is the study of good and bad code quality practices, and more in particular the study of code reuse and code cloning practices. Indeed, many of the reported "code smells" in the research literature on software quality are related to code duplication [8, 9]. Previous research has already demonstrated the existence of code duplication in Dockerfiles [10, 11], as well as higher maintenance

BENEVOL24: The 23rd Belgium-Netherlands Software Evolution Workshop, November 21-22, Namur, Belgium
Guillaume.CARDOEN@umons.ac.be (G. Cardoen)
0009-0005-2008-3565 (G. Cardoen)

effort, prolonged fixes and inconsistent changes due to clones in build configuration files for Java systems [12]. I thus posit that cloning and reuse practices are also likely to be used in CI/CD configuration files. This makes it relevant to empirically study the prevalence and impact of such practices in project repositories hosted on social coding platforms like GitHub and GitLab.

I hypothesise that project maintainers often tend to create new CI/CD configurations by copy-pasting existing configurations from elsewhere, either from other repositories they own or know about, or by relying on starter template configurations that are freely provided by the social coding platform. It remains an open question whether such copy-and-paste reuse is actually beneficial in the specific case of CI/CD, how duplicated code evolves, and whether it leads to maintainability problems in the long run. Building further on the preliminary findings of my master's thesis [13], the goal of my PhD research is therefore to evaluate the code cloning and code reuse practices in CI/CD ecosystems, with an initial focus on GitHub Actions, and extending the scope of the study to other CI/CD ecosystems such as GitLab CI/CD in later phases.

The remainder of this article is structured as follows. Section 2 presents the research context, focusing on related research on code reuse and code cloning, and presenting the GitHub Actions CI/CD ecosystem that will be used as the first case study. Section 3 presents the research goals, subdivided into nine research questions. Section 4 presents my current progress.

2. Context and Related Research

2.1. Code Reuse and Code Cloning

According to Krueger [14], software reuse consists of building new software based on existing software artifacts (such as code fragments or design structures). Such reuse is a means to reduce development time and maintenance effort needed to build large and complex programs, as well as to enhance their general software quality posture [14]. Similarly, code reuse is the use of code snippets from existing systems or components developed especially for being reused [15]. Many studies relating to code reuse can be found in the literature [16]. Such code reuse can be typically classified into multiple strategies: copy-paste, clone-and-own and platform orientation [16].

Copy-paste or clone-and-own reuse can lead to similar or identical code fragments, commonly referred to as *code clones* [9]. They could arise accidentally, but are typically introduced due to code duplication, forking, merging of similar systems, automated code generation tools, or due to changes made by developers during maintenance [17, 18]. Code clones are a heavily studied field in software engineering [19, 20, 21].

Code clones can have beneficial effects such as easier understanding of the code, reduced development time, or ensuring robustness [22, 9, 18]. On the other hand, code clones are also considered as bad smells [8] as they tend to increase the code base size, may lead to bug propagation, have a negative impact on code readability, hide the originality of the code, increase maintenance cost or result in inconsistent and repetitive bug fixes [23, 9]. Due to both positive and negative effects of code clones, researchers propose that code clones should at least be detected, even in the absence of refactoring them [17].

Code clones are often categorized into four types [24]: Type I clones represents code fragments that are identical, ignoring differences in comments or whitespace; Type II clones extends Type I by allowing different identifiers, literals, methods names and types; Type III clones includes Type II by allowing added, removed or modified code lines; Type IV clones are so-called *semantic* clones that have the same functional goal and behaviour even if they may look different syntactically.

Many general-purpose code clone detection tools exist [19, 17]. To name but a few: NiCad [25], CCFinder [26], CCFinderSW [27], SourcererCC [28], Deckard [29], and CCSharp [30]. They can be classified into multiple categories, including text-based, token-based, tree-based, graph-based, metric-based, and learning-based. Hybrid clone detection approaches may belong to multiple categories [19]. Some tools also offer automatic refactoring of code clones. Arcelli Fontana et al. [31] developed such a tool for Java project, using NiCad [25] as clone detector.

Code fragment similarity and code fingerprinting are closely related fields to code clone detection. Similarity Preserving Hashing Functions (SPHF) are functions mapping two similar inputs to two similar outputs [32]. Fingerprinting approaches divide documents in multiple term sequences and derive one or multiple fingerprints from these sequences. Documents sharing one or more equivalent fingerprints are susceptible to contain reused code fragments [33]. Both SPHF and fingerprinting approaches have been proposed for code clone detection [34, 33, 32].

Many empirical studies on code clones have been carried out in the past. To name but a few, Pate et al. [35] presented a systematic literature review related to code clone evolution. Mondal et al. [36] analysed the propagation of bugs through code clones, concluding that 18.42% of code fragments where a bug fix occurred are due to propagated bugs. Estefó et al. [37] studied code duplication in Robot Operating System and found that almost half of the analysed launch configuration files contained at least one code clone. Oumaziz et al. [10] analysed duplicated code in Dockerfiles via an index-based duplicate detection, finding that new instructions are frequent and issues experienced by developers explains a majority of modifications. Oumaziz [11] studied code clones in API documentation and Dockerfiles, finding that a majority of instructions in Dockerfiles are duplicated. McIntosh et al. [12] collected and analysed 3,872 open source build system configuration files. Considering only Type I clones, they concluded that half of build logic lines are cloned at least once in Java build systems. However, they noticed some build systems with a limited number of clones, suggesting cloning is not a necessity in build system configuration files. They observed that inconsistent changes and prolonged fixes, known to be clone-related problems in general-purpose code [23], are also affecting build system files.

2.2. GitHub Actions

GitHub is the most prominent social coding platform [38]. GitHub Actions, being the most popular CI/CD system used in GitHub projects [39], is arguably one of the most popular CI/CD ecosystem. This section presents its concepts and existing empirical research that has been conducted on this quite recent ecosystem.

GitHub has started to provide support for CI/CD within GitHub itself since November 2019 through GitHub Actions, which quickly became the most frequently employed CI/CD solution for GitHub projects [39]. Listing 1 shows a workflow configuration file, given by GitHub,[1] for building a Python package and releasing it on PyPI. The workflow is stored in a YAML file respecting a given syntax specified by the GitHub documentation.[2]

A workflow file has at least two main parts describing when it is triggered (`on:` key) and what should be performed (`jobs:` key). The `on:` key describes the list of events upon which the workflow should run. Lines 3-5 of Listing 1 declare that the workflow should run each time a new release is published. The `jobs:` key declares one or more jobs, each composed of one or more steps. A step can use (`uses:` key) a reusable component, called Action, or run (`run:` key) a sequence of shell commands. Actions will be described in Section 2.3.1.

Line 14 in Listing 1 declares a call to the `actions/checkout` Action (version 4) to clone the repository, while line 16 calls version 3 of another Action `actions/setup-python` to install and configure Python in the environment. Lines 17-18 are declaring a variable `python-version` used by the Action in order to specify the python version to install. Lines 20-22 declare the execution of two shell commands, the first one installing `pip` and the second one installing the Python package `build` with `pip`. Line 24 launches the python build system. Finally, line 26 calls the Action `pypa/gh-action-pypi-publish` uploading the built Python package to PyPI. Lines 27-29 declare two parameters that will be passed to the Action.

Prior research on GitHub Actions has focused on the adoption and usage of GitHub Actions. Kinsman et al. [40] studied how developers use GitHub Actions in open-source projects after its adoption. Golzadeh et al. [39] quantitatively studied the usage of different CI/CD tools used in GitHub and

[1]https://github.com/actions/starter-workflows/blob/main/ci/python-publish.yml
[2]https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions

Listing 1: GitHub Actions workflow file for building a Python package

```yaml
name: Upload Python Package

on:
  release:
    types: [published]

permissions:
  contents: read

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - name: Set up Python
      uses: actions/setup-python@v3
      with:
        python-version: '3.12'
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install build
    - name: Build package
      run: python -m build
    - name: Publish package
      uses: pypa/gh-action-pypi-publish@27b31702a0e7fc50959f5ad993c78deac1bdfc29
      with:
        user: __token__
        password: ${{ secrets.PYPI_API_TOKEN }}
```

Rostami Mazrae et al. [41] qualitatively analysed the usage, co-usage and migration among CI/CD tools, noticing a migration towards GitHub Actions. Valenzuela-Toledo and Bergel [42] studied the evolution of GitHub Actions workflow files. Mazrae et al. [43] studied the main changes in workflow files, concluding that files are subject to frequent modifications. Decan et al. [6] analysed the reuse of Actions and automation practices for GitHub Actions workflows, concluding nearly all jobs make use of them. Saroar and Nayebi [44] qualitatively surveyed Actions developers in the goal of understanding how they use, create and search for Actions and identifying possible challenges they face while doing so. Khatami et al. [45] analysed 10,012 commits from the 83 most populars repositories using GitHub Actions looking for different kind of smells. They proposed an automatic smell detectors for each identified smell, and gave insights on how developers reacted to those smells, concluding 7 out of 22 smells are revelant to developers whereas 9 out of 22 had mixed reactions.

2.3. GitHub Actions Code Reusability Mechanisms

The integrated CI/CD mechanisms offered by GitHub provide a variety of methods for facilitating code reuse in workflows. This section describes the principal means of code reuse available in GitHub Actions.

2.3.1. Reusable Actions

GitHub Actions supports different type of reusable components directly referenceable inside its workflows. In the GitHub Actions ecosystem, reusable components are called *Actions.* An Action represents an individual task, combinable with others to create a job. They can either consist of a single Javascript

Listing 2: GitHub Action workflow file for building a Python package via a reusable workflow

```yaml
# previous keys (on, permissions, etc.)

jobs:
  deploy:
    uses: ./.github/workflows/pypi-reusable.yaml
    with:
      python-version: 3.12
      username: __token__
    secrets:
      password: ${{ secrets.PYPI_API_TOKEN }}
```

file being executed, be composed themselves of Actions or shell commands, or refer to a Docker container. In the latter case, the container is then used as environment for the job, thus allowing specific version and dependency management. The objective of Actions is thus to replace frequent sequences of commands following the principle of "don't repeat youself" (DRY).

The GitHub Marketplace[3] enables maintainers to search for and share Actions. Alternatively, an Action can be referenced by the repository hosting its code.

2.3.2. Reusable Workflows

Reusable workflows[4] represent another possibility for code reuse. As its name suggests, reusable workflows enables the maintainer to declare a workflow reusable by others. Such a reusable workflow is declared in a very similar manner as done in Listing 1. Once declared, this workflow can be referenced elsewhere, possibly in workflow files located in different repositories. Unlike Actions, a reusable workflow can contain multiple jobs, each composed of multiple steps.

By using this reusability mechanism, the workflow presented at the Listing 1 can be transformed to reference a reusable workflow. The resulting workflow, shown at the Listing 2, is a shorter workflow file, where the steps of the `deploy` job are replaced by a reference to a reusable workflow (located in the same repository in our example), which describes these steps.

2.3.3. Composite Actions

Composite Actions are a type of reusable Actions. A composite Action bundles a list of Actions or shell commands as a single Action, thus appearing as a single step in a workflow file.

Assuming a composite Action `example/build-publish-python@v1` is declared and bundles the last four steps of the workflow presented at the Listing 1, we could simplify our workflow by replacing these steps with a reference to `example/build-publish-python@v1`. An example of resulting workflow is presented at the Listing 3. Line 8 calls a composite Action, while lines 9-12 are giving needed variables to the Action (such as username and password).

2.3.4. Starter Workflows

GitHub freely provides examples and templates of GitHub Actions workflows, called *starter workflows*. Such starter workflows can be found freely in a GitHub repository[5] or can also be integrated automatically via the GitHub web interface. Maintainers can copy-paste the content into their own workflows and adapt them to their own use-case. Such reuse allows for a different starting point when creating a new workflow file, offering an alternative to a blank workflow file. Nevertheless, such reuse can also result in the creation of a significant amount of code clones across workflow files.

[3]https://github.com/marketplace?type=actions
[4]https://docs.github.com/en/actions/sharing-automations/reusing-workflows
[5]https://github.com/actions/starter-workflows

Listing 3: GitHub Action workflow file for building a Python package via a composite action

```
# previous keys (on, permissions, etc.)

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - uses: example/build-publish-python@v1
      with:
        python-version: 3.12
        username: __token__
        password: ${{ secrets.PYPI_API_TOKEN }}
```

As an example, the previously presented workflow in Listing 1 is one of the available starter workflow, where only the python version was changed from `3.x` to `3.12`.

3. Research Goals

Since configuration-as-code files essentially contain code, I hypothesise that they are affected by the majority of issues encountered in general-purpose code, even though the syntax for expressing configuration files can be quite different from the one of traditional programming languages.

In the specific case of GitHub Actions, Saroar and Nayebi [44] qualitatively concluded that writing a workflow based on a previously written workflow file and using a starter workflow as a basis are the two most popular methods used by consulted GitHub Actions maintainers when writing a workflow file. The majority of consulted maintainers thus appear to be basing themselves on other workflows, which may be indicative of the popularity of copy-paste or clone-and-own reuse across workflow files. Such reuse might lead to a substantial number of code clones in GitHub Actions workflow files.

I thus hypothesise that CI/CD configuration files also suffer from code cloning, due to copy-paste or clone-and-own reuse and the many observed reasons for code duplication in general purpose code (such as automatic code generation [17]). My doctoral research therefore aims to empirically study code clones in configuration as code files of CI/CD tools. As a first step (**G1-3**), I will study the GitHub Actions ecosystem, and in a final step (**G4**), I will extend this research to a wider range of tools, and also compare the practices and impact of code cloning across these tools.

To achieve these different goals, I plan on using a mixed-method empirical analysis as advocated by Creswell [46]. Such mixed-method empirical analysis entails two primary research methodologies: qualitative and quantitative. Quantitative studies and mining of software repositories are typically sufficient to inform decision-making. However, a qualitative approach may be necessary to better understand the underlying causes behind specific phenomena, which might prove difficult to study via a quantitative approach.

3.1. G1: Assessing Code Cloning Practices in the GitHub Actions Ecosystem

G1 will mainly focus on GitHub Actions workflow files. After a careful review of the literature, I found no previous research analysing code clones for GitHub Actions.

In the goal of quantifying and describing code clones in GitHub Actions, we will proceed to address the following research questions:

RQ1: What are code clones in workflows and how to identify them?

This question will study how code clones can be detected in GitHub Actions workflow files. The order and structure of the syntactic keys used in workflow files (such as ‘on:’, ‘jobs:’, ‘uses:’ or ‘run:’) lead to quite some repetition and syntactic similarity across workflow files. We therefore expect current clone detection algorithms to yield many false positives and lose precision when applied on configuration code, which can be syntactically quite different from general-purpose code. We plan to devise an algorithm taking these syntactic specificities into account in order to precisely detect code clones in workflow files. We will exclude Type IV clones from the analysis as they do not depend on the syntax.

RQ2: How prevalent are code clones in workflow files?

This question aims to quantify code clones across workflow files. We plan to empirically analyse code clone presence in GitHub Actions workflow files, thus confirming the presence of code clones as well as the extent of duplicated code in such files.

RQ3: What are the characteristics of code clones?

Clone characteristics such as length or position will be studied as well as social characteristics such as the author and maintainer of code clones. This question will also analyse the code clone contents, e.g., whether they are complete jobs, sequence of steps or events. How code clones differ or resemble one another will also be studied in this question. Such analysis will provide insights on which parts of workflow configurations maintainers tend to clone and why.

3.2. G2: Understanding code cloning practices

In order to gain a deeper comprehension of code clones in GitHub Actions workflow files, this second goal aims to further analyse code cloning practices of workflow files by studying clones provenance, evolution and impacts. More specifically, we will address the following three research questions:

RQ4: Where do code clones come from?

Assuming code cloning is a popular practice among GitHub Actions workflows maintainers, such code cloning could come from several sources. Maintainers could use their own workflows, their organisation workflows or GitHub starter workflows as a basis.

Based on the results of **G1**, we will construct and study the code clone evolution history, also known as *clone genealogy*, inspired by existing research [47, 48]. By leveraging an existing differencing tool for GitHub Actions workflow files developed at our lab [49], we will quantitatively study the origin of code clones.

Such a study will provide insights in how code clones are introduced in workflow files and knowledge in how workflow files are created.

In addition, we will also qualitatively study how developers tend to reuse code. The process of choosing what code to reuse and the practice chosen by developers to do so will also be a part of this study.

RQ5: How do code clones co-evolve?

By further studying the clone genealogy constructed in **RQ2**, this question aims to understand the co-evolution of code clones. Code clones can follow multiple evolution patterns such as consistent evolution (i.e., code fragments stays similar to a given similarity metric), independent evolution (i.e., code fragments become different to a given similarity metric), delayed or late propagation (i.e., code fragments changes consistently at different times) [50]. Which evolution pattern do code clones follow in the specific case of GitHub Action is still unknown.

RQ6: What is the impact of code clones?

This question seeks to understand how code clones affect workflow files and their creation or usage. In general purpose code, code clones can be beneficial in certain instances, yet exert a detrimental effect in others [22, 9, 18, 23, 8]. However, it is still unknown whether code clones impact positively or negatively workflow files. This question will thus study the different impacts of code clones in workflows, basing its answer on the previously studied aspects of clones (studied in **RQ4** and **RQ5**).

3.3. G3: Improving code reuse practices

The objective of this goal is to improve code reuse practices in GitHub Actions workflow files by assisting maintainers in the avoidance of code clones considered detrimental.

This goal is highly dependent on the conclusions of **G1** and **G2**. It is our expectations that previous goals will enable us to improve code reuse practices in GitHub Actions workflows. However, this goal will be adapted or removed depending on the conclusions of **G1** and **G2**.

RQ7: Are reusable components introduced in workflows?

This question seeks to understand whether code clones transform into reusable components in the context of GitHub Actions workflow files, and whether this change impacts their code clones. GitHub introduced multiple code reusability mechanisms (such as reusable workflows or composite Actions) that could limit code cloning. However, whether maintainers refactor their workflow files in order to use these mechanisms and thus, reduce code duplication, is still unknown. Similarly, the reasons to use (or avoid) such mechanisms and the difficulties encountered by maintainers in doing so are still unknown and will be qualitatively studied as part of this research question.

RQ8: How to help maintainers avoid code clones?

As it is already the case with general purpose code [9], we expect code clones in GitHub Actions workflow files to have positive and negative effects. (The veracity of these expectations will be established in **RQ6**). Though some workflows may not need nor benefit from code clones removal, we expect that refactoring often encountered code fragments will mitigate or remove adverse effects of code clones, thus helping maintainers. The objective of this research question is thus to identify and implement strategies in order to help maintainers avoid code clones, or at least mitigate the adverse effects of such clones.

Such strategies could include refactoring code clones into reusable components (such as Actions or reusable workflows). Such refactoring could use code clones in order to create reusable components and provide an automatic patch introducing such components. This automatic refactoring may be a response to possible difficulties encountered by maintainers as will be studied in **RQ4**.

Due to limitations of reusability mechanisms GitHub or specific needs in workflows, we expect to be unable to refactor some code clones. Another strategy thus involves automatically propagating changes of code clones in order to support their co-evolution studied in **RQ5**. Such a strategy will mitigate or remove bug propagation while also reducing maintenance costs, which are two of the adverse effects of code clones [9].

This research question will be divided into multiple research questions as my doctoral research progresses. As my doctoral research has only recently begun, I currently do not have enough hindsight related to this question to provide a detailed description.

3.4. G4: Generalisation to Other CI/CD Tools and Social Coding Platforms

Other CI/CD tools (such as Jenkins[6], CircleCI[7], Travis[8] or AppVeyor[9]) or social coding platforms (such as GitLab[10] or Gitea[11]) uses configuration-as-code practice in the scope of CI/CD configuration. The second goal aims to expand the scope of the study to other CI/CD ecosystems, allowing us to compare how code cloning and reuse varies between different CI/CD ecosystems using configuration-as-code practice.

The methods of this second goal is based on the same fundamental concepts of **G1-3** methods. A large-scale and representative dataset of configuration files will be required for each considered CI/CD.

[6]https://www.jenkins.io/
[7]https://circleci.com/
[8]https://www.travis-ci.com
[9]https://www.appveyor.com/
[10]https://docs.gitlab.com/ee/ci/
[11]https://about.gitea.com/

Large-scale archiving system (such as Software Heritage [51]) could be a first step in retrieving enough data for different CI/CD. Moreover, in the particular case of GitLab, such archiving system could also preserve self-hosted GitLab instances, allowing us to consider such instances. Concerning the detection of code clones, I plan to adapt the algorithm developed in Goal 1 to other configurations files, beginning with those based on YAML for simplicity. The depth of this goal and its feasibility will be dependent on the time left in my thesis.

The following research question will be addressed by this goal:

RQ9: What are the similarities between code clones of different CI/CD ecosystems?

This question aims to compare the different code clones between different CI/CD ecosystems providing knowledge about whether practices related to code clones differ per ecosystem or whether they are common. This question will also provide insights into how maintainers deal with code clones on different ecosystems.

One part of this question will focus on the code reusability practices used in different CI/CD tools. Understanding and comparing the practices used in different platforms will allow us to understand why a given practice is used in a single CI/CD, or to the contrary, common to all. In doing so, we expect to identify how code reusability systems are used in different CI/CD tools, and to formulate various advices helping developers of different platforms.

Moreover, the content and semantic behind detected clones will be analysed and compared. Such an analysis may result in recommendations, easing the task of creating and maintaining CI/CD configuration files.

As a first step, this question will focus on GitLab CI/CD. Similarly to GitHub Actions, GitLab CI/CD is directly integrated into GitLab via a YAML configuration files located inside the repository. Whereas the main ideas are the same, they are structured differently in a GitLab configuration file. Gitea is another social coding platform integrating a CI/CD. It is of particular interest in this question as its configuration files follow the same syntax of GitHub Actions, though some differences may occur.[12] Other CI/CD will be included in this research questions depending on the time left in my thesis.

This last research question will later be divided into multiple research questions. As my doctoral research has only just begun, I currently do not have enough hindsight related to provide details related to this question.

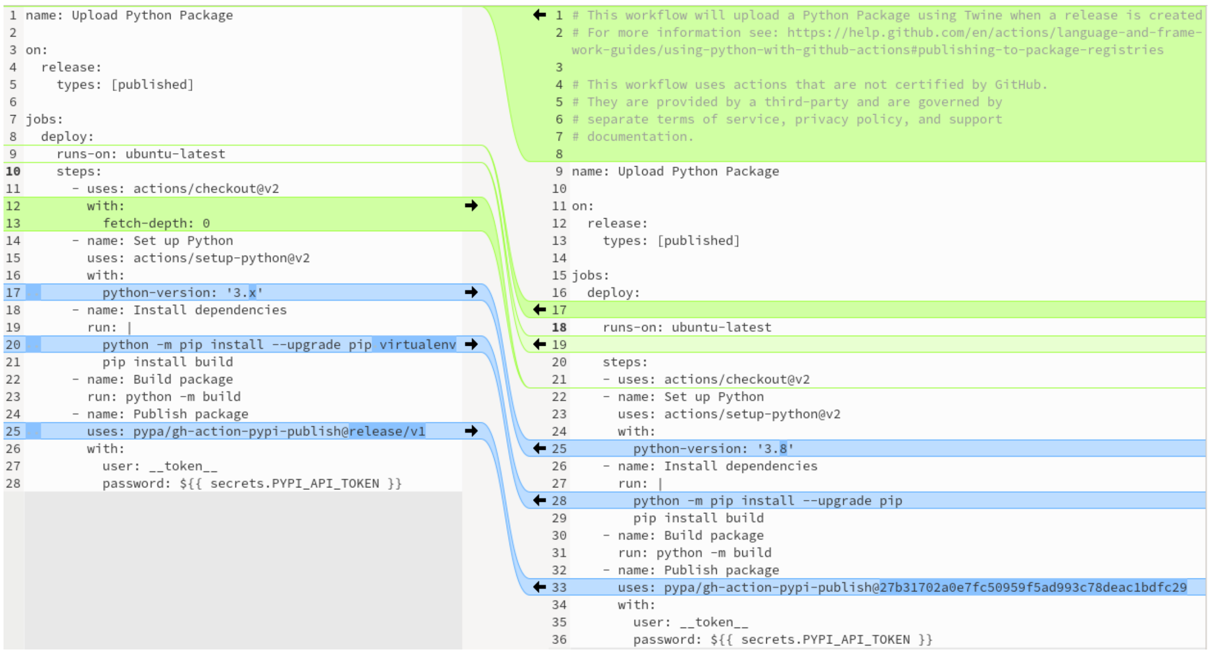
4. Progress and Future Work

This section explains my current progress for the different goals.

4.1. Dataset

A preliminary goal of my thesis is to gather a large historical dataset of GitHub workflow files, allowing us to quantitatively study the code duplication and reuse in the context of workflow files. To this end, I developed and published *gigawork* (which is an acronym for "**Gi**ve me **G**itHub **A**ctions **Work**flows"), an open-source extraction tool for extracting workflow files from a git repository.[13] *gigawork* traverses the git history tree beginning from a given git reference, git HEAD by default, following the first-parent rule.[14] For each commit touching a workflow file, it extracts the workflow file version as well as other metadata (such as the commit date, the author, ...) [52]. gigawork was applied on a list of 43K+ repositories, obtained via a query to SEART search engine [53] (where we excluded repositories not using GitHub Actions). In an updated version of the dataset [52], 2.5M+ workflow files, representing 219K+ workflow histories, are present. In addition, boolean flags indicating which workflow files of the dataset are valid YAML files and which respect the current GitHub Actions syntax were added by *gigawork*.

[12]https://docs.gitea.com/next/usage/actions/comparison
[13]https://github.com/cardoeng/gigawork
[14]This rule dictates that git only follows the first parent of each commit.



```yaml
name: Upload Python Package

on:
  release:
    types: [published]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip virtualenv
          pip install build
      - name: Build package
        run: python -m build
      - name: Publish package
        uses: pypa/gh-action-pypi-publish@release/v1
        with:
          user: __token__
          password: ${{ secrets.PYPI_API_TOKEN }}
```

```yaml
# This workflow will upload a Python Package using Twine when a release is created
# For more information see: https://help.github.com/en/actions/language-and-framework-guides/using-python-with-github-actions#publishing-to-package-registries

# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

name: Upload Python Package

on:
  release:
    types: [published]

jobs:
  deploy:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.8'
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install build
    - name: Build package
      run: python -m build
    - name: Publish package
      uses: pypa/gh-action-pypi-publish@27b31702a0e7fc50959f5ad993c78deac1bdfc29
      with:
        user: __token__
        password: ${{ secrets.PYPI_API_TOKEN }}
```

Figure 1: A visual diff of two workflows found in GitHub repositories.

Although the dataset does not contain any information about code clones, we can use it to detect such code clones. For instance, two very similar workflow files, extracted from the dataset, can be found at Figure 1.[15] Both workflows are very similar to the one presented at Listing 1, though they are coming from different repositories. Moreover, they also have the same goal. As seen on Figure 1, the main changes between these two workflows are comments and version changes. We can also note the addition of a new python dependency (virtualenv). Finally, a limit of depth for fetching the repository was added via fetch-depth variable.

4.2. Code clones detection

In the goal of answering **RQ1**, multiple ways of detecting code clones in GitHub Actions workflow files were already studied and experimented. However, no definitive algorithm was found yet. I am thus currently focussing on **RQ1**, continuing experimenting different approaches in the goal of efficiently detecting code clones, while keeping in mind this algorithm might be extended to other YAML files.

References

[1] GitHub, Github actions now supports ci/cd, free for public repositories, https://github.blog/2019-08-08-github-actions-now-supports-ci-cd/, 2019. [Online; accessed 21 June 2024].

[2] GitHub, Octoverse: The state of open source and rise of ai in 2023, https://github.blog/2023-11-08-the-state-of-open-source-and-ai/, 2023. [Online; accessed 13 May 2024].

[3] GitLab, Gitlab 8.0 released with new looks and integrated ci!, https://about.gitlab.com/releases/2015/09/22/gitlab-8-0-released/, 2015. [Online; accessed 21 June 2024].

[4] GitLab, The most-comprehensive ai-powered devsecops platform, https://about.gitlab.com/, 2024. [Online; accessed 22 June 2024].

[5] M. L. Gupta, R. Puppala, V. V. Vadapalli, H. Gundu, C. Karthikeyan, Continuous integration,

[15] Left side corresponds to https://github.com/instrumentkit/InstrumentKit/blob/main/.github/workflows/deploy.yml, right side corresponds to https://github.com/ASUS-AICS/LibMultiLabel/blob/master/.github/workflows/python-publish.yml. Both represents the latest version available in the dataset.

delivery and deployment: A systematic review of approaches, tools, challenges and practices, in: International Conference on Recent Trends in AI Enabled Technologies, Springer, 2024, pp. 76–89.

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

[7] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas, A. Glover, J. Holman, J. Micco, B. Murphy, T. Savor, et al., The top 10 adages in continuous deployment, IEEE Software 34 (2017) 86–95.

[8] M. Fowler, Refactoring: improving the design of existing code, Addison-Wesley Professional, 2018.

[9] N. Saini, S. Singh, et al., Code clones: Detection and management, Procedia computer science 132 (2018) 718–727.

[10] M. A. Oumaziz, J.-R. Falleri, X. Blanc, T. F. Bissyandé, J. Klein, Handling duplicates in dockerfiles families: Learning from experts, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2019, pp. 524–535.

[11] M. A. Oumaziz, Cloning beyond source code: a study of the practices in API documentation and infrastructure as code., Ph.D. thesis, Bordeaux, 2020.

[12] S. McIntosh, M. Poehlmann, E. Juergens, A. Mockus, B. Adams, A. E. Hassan, B. Haupt, C. Wagner, Collecting and leveraging a benchmark of build system clones to aid in quality assessments, in: International Conference on Software Engineering, 2014, pp. 145–154.

[13] G. Cardoen, Une analyse empirique de la duplication de code dans les CI/CD workflows sur GitHub, Master's thesis, University of Mons, 2023.

[14] C. W. Krueger, Software reuse, ACM Computing Surveys (CSUR) 24 (1992) 131–183.

[15] S. Haefliger, G. Von Krogh, S. Spaeth, Code reuse in open source software, Management science 54 (2008) 180–193.

[16] J. Krüger, T. Berger, An empirical analysis of the costs of clone-and platform-oriented software reuse, in: ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, 2020, pp. 432–444.

[17] M. Zakeri-Nasrabadi, S. Parsa, M. Ramezani, C. Roy, M. Ekhtiarzadeh, A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges, Journal of Systems and Software (2023) 111796.

[18] C. K. Roy, J. R. Cordy, A survey on software clone detection research, Queen's School of computing TR 541 (2007) 64–68.

[19] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, B. Maqbool, A systematic review on code clone detection 7 (2019) 86121–86144.

[20] M. Lei, H. Li, J. Li, N. Aundhkar, D.-K. Kim, Deep learning application on code clone detection: A review of current knowledge, Journal of Systems and Software 184 (2022) 111141.

[21] A. Sheneamer, J. Kalita, A survey of software clone detection techniques, International Journal of Computer Applications 137 (2016) 1–21.

[22] C. J. Kapser, M. W. Godfrey, "cloning considered harmful" considered harmful: patterns of cloning in software, Empirical Software Engineering 13 (2008) 645–692.

[23] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner, Do code clones matter?, in: International Conference on Software Engineering, IEEE, 2009, pp. 485–495.

[24] C. K. Roy, M. F. Zibran, R. Koschke, The vision of software clone management: Past, present, and future (keynote paper), in: Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), IEEE, 2014, pp. 18–33.

[25] C. K. Roy, J. R. Cordy, Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization, in: International Conference on Program Comprehension, IEEE, 2008, pp. 172–181.

[26] T. Kamiya, S. Kusumoto, K. Inoue, CCFinder: A multilinguistic token-based code clone detection system for large scale source code, IEEE Transactions on Software Engineering 28 (2002) 654–670.

[27] Y. Semura, N. Yoshida, E. Choi, K. Inoue, CCFinderSW: Clone detection tool with flexible multilingual tokenization, in: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2017, pp. 654–659.

[28] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, C. V. Lopes, SourcererCC: Scaling code clone detection to big-code, in: Proceedings of the 38th international conference on software engineering, 2016, pp. 1157–1168.

[29] L. Jiang, G. Misherghi, Z. Su, S. Glondu, Deckard: Scalable and accurate tree-based detection of code clones, in: 29th International Conference on Software Engineering (ICSE'07), IEEE, 2007, pp. 96–105.

[30] M. Wang, P. Wang, Y. Xu, Ccsharp: An efficient three-phase code clone detector using modified pdgs, in: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2017, pp. 100–109.

[31] F. Arcelli Fontana, M. Zanoni, F. Zanoni, A duplicated code refactoring advisor, in: Agile Processes in Software Engineering and Extreme Programming, Springer, 2015, pp. 3–14.

[32] V. Gayoso Martínez, F. Hernández Álvarez, L. Hernández Encinas, State of the art in similarity preserving hashing functions (2014).

[33] L. Lulu, B. Belkhouche, S. Harous, Overview of fingerprinting methods for local text reuse detection, in: 2016 12th International Conference on Innovations in Information Technology (IIT), IEEE, 2016, pp. 1–6.

[34] J. Martinez-Gil, Source code clone detection using unsupervised similarity measures, in: International Conference on Software Quality, Springer, 2024, pp. 21–37.

[35] J. R. Pate, R. Tairas, N. A. Kraft, Clone evolution: A systematic review, Journal of software: Evolution and Process 25 (2013) 261–283.

[36] M. Mondal, B. Roy, C. K. Roy, K. A. Schneider, An empirical study on bug propagation through code cloning, Journal of Systems and Software 158 (2019) 110407.

[37] P. Estefó, R. Robbes, J. Fabry, Code duplication in ROS launchfiles, in: International Conference of the Chilean Computer Science Society (SCCC), IEEE, 2015, pp. 1–6.

[38] GitHub, Octoverse 2022: The state of open source software, https://octoverse.github.com/2022/developer-community, 2022. [Online; accessed 13 May 2024].

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

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

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

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

[43] P. R. Mazrae, A. Decan, T. Mens, M. Wessel, A Preliminary Study of GitHub Actions Workflow Changes, in: CEUR Workshop Proceedings, 2023.

[44] S. G. Saroar, M. Nayebi, Developers' Perception of GitHub Actions: A Survey Analysis, in: International Conference on Evaluation and Assessment in Software Engineering (EASE), ACM, 2023.

[45] A. Khatami, C. Willekens, A. Zaidman, Catching Smells in the Act: A GitHub Actions Workflow Investigation, in: International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2024.

[46] J. W. Creswell, Mixed-method research: Introduction and application, in: Handbook of educational policy, Elsevier, 1999, pp. 455–472.

[47] M. Kim, V. Sazawal, D. Notkin, G. Murphy, An empirical study of code clone genealogies, in: European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005, pp. 187–196.

[48] J. Krinke, A study of consistent and inconsistent changes to code clones, in: Working Conference on Reverse Engineering (WCRE), IEEE, 2007, pp. 170–178.

[49] P. R. Mazrae, A. Decan, T. Mens, gawd: A differencing tool for GitHub Actions workflows, in: International Conference on Mining Software Repositories, ACM, 2024.

[50] S. Thummalapenta, L. Cerulo, L. Aversano, M. Di Penta, An empirical study on the maintenance of source code clones, Empirical Software Engineering 15 (2010) 1–34.
[51] R. Di Cosmo, S. Zacchiroli, Software heritage: Why and how to preserve software source code, in: International Conference on Digital Preservation, 2017, pp. 1–10.
[52] G. Cardoen, T. Mens, A. Decan, A dataset of GitHub Actions workflow histories, in: International Conference on Mining Software Repositories, ACM, 2024.
[53] O. Dabic, E. Aghajani, G. Bavota, Sampling projects in GitHub for MSR studies, in: International Conference on Mining Software Repositories (MSR), IEEE, 2021, pp. 560–564.