Agile Processes

Group Project

**JDPM\_BOYS**

Daniel - dodriscoll4@mycit.ie - R00221504

Patryk - Patryk.Malinowski@mycit.ie - R00210173

Jamie - jamie.roche1@mycit.ie - R00151829

Mohammed - mohammed.al-hajri@mycit.ie - R00208964

A logo of a person running on a keyboard

Description automatically generated

Table of Contents

[Introduction 2](#_Toc152312608)

[Setup 3](#_Toc152312609)

[Configuration 4](#_Toc152312610)

[GitHub Actions Workflow 4](#_Toc152312611)

[Initial Code Review and Task Breakdown 6](#_Toc152312612)

[Testing Functionality and Issue Identification 6](#_Toc152312613)

[Issue Resolution and Implementation of Fixes 7](#_Toc152312614)

[Test File Creation 7](#_Toc152312615)

[Experience 8](#_Toc152312616)

[Usefulness 8](#_Toc152312617)

[Ease of Use 9](#_Toc152312618)

[Challenges 9](#_Toc152312619)

[Conclusion 10](#_Toc152312620)

# Introduction

The objective of this group project was to evaluate the effectiveness of GitHub Actions, a continuous integration (CI) tool provided by GitHub, in facilitating collaboration among software developers. GitHub serves as a central hub for version control and collaboration, allowing teams to manage and track changes to their codebase efficiently.

GitHub Actions is a CI (Continuous Integration) service integrated into GitHub, automating the software development lifecycle. It enables developers to define workflows using YAML files, automating tasks such as building, testing, and deploying code. Continuous integration involves regularly integrating code changes into a shared repository. GitHub Actions plays a crucial role in automating this process, ensuring that code changes are continuously tested and validated, leading to early bug detection and enhanced code reliability.

In the context of our project, we focused on creating a continuous integration workflow for a Python project. This workflow not only showcased the capabilities of GitHub Actions but also demonstrated the integration of agile principles, emphasising adaptability and collaboration, work visualisation for better code management, and continuous integration to produce a cohesive, accurate, and robust software artefact. The goal was to instil a high level of confidence in the development process, emphasising the importance of automated testing and collaborative practices.

GitHub Actions System Diagram


Figure 1 GitHub Actions System Diagram.

# Setup

To establish a collaborative environment, we initiated a shared repository on GitHub for our Python project. This involved creating a new repository, configuring settings, navigating to the collaborator’s tab, and inviting team members. Each member then set up their local repository on their machine, enabling them to contribute to the Python project via push and pull requests.

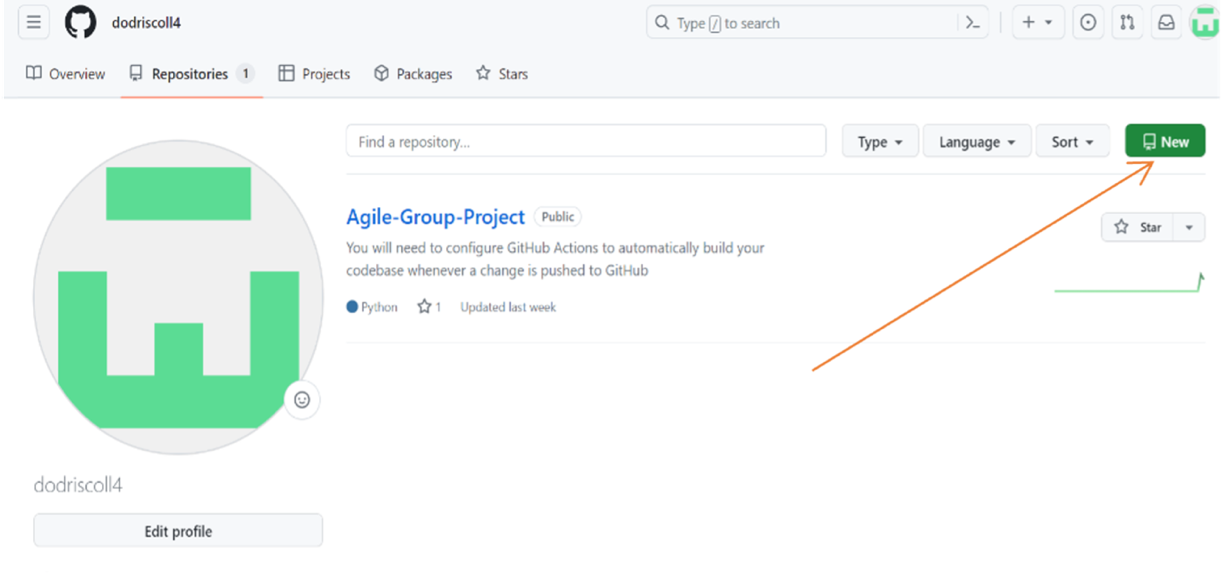


Figure 2 Screenshot of creating repository.

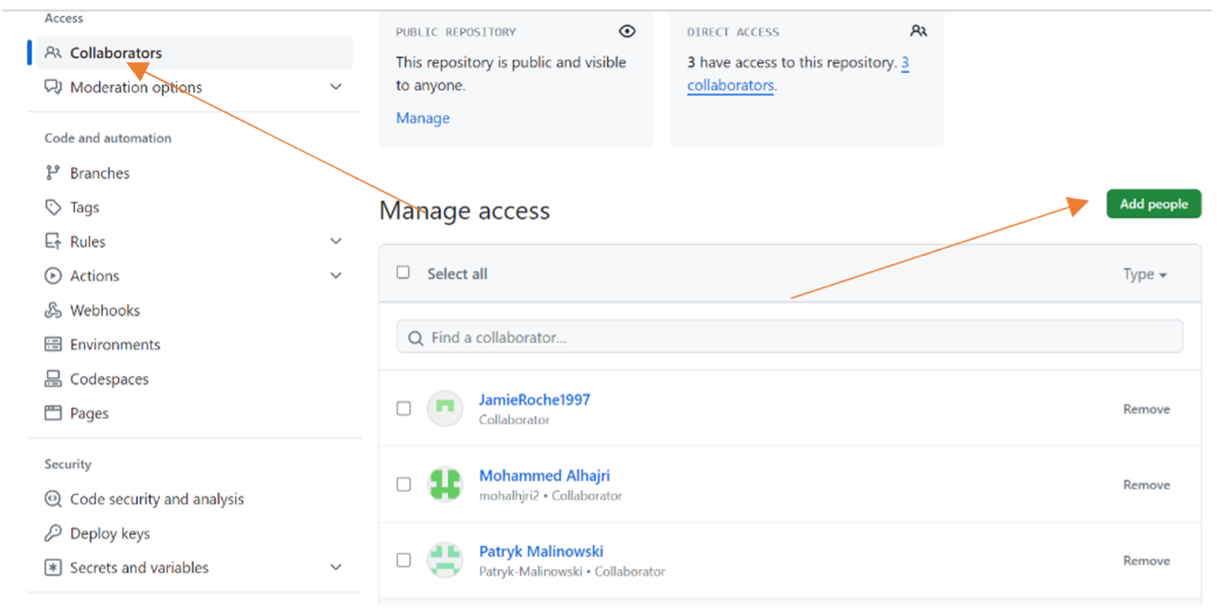


Figure 3 Adding team members to repository.

Additionally, the team adopted a version control strategy, utilising branches and pull requests to manage code changes efficiently. This approach allowed for streamlined collaboration and effective version control.

# Configuration

## GitHub Actions Workflow

Next, we implemented a GitHub Actions workflow for our repository, leveraging the pre-built Python application workflow available on GitHub. This workflow included pre-configured settings for building, running, and testing our Python code. This was done by clicking actions, creating a new workflow, and finding the Python application workflow that is pre-built for GitHub Actions. This means that Python dependencies are already included allowing us to build, run and test our Python code through GitHub Actions:

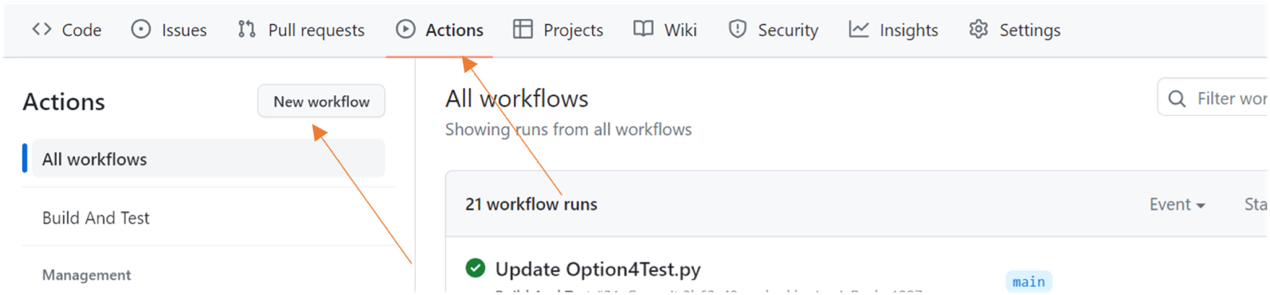


Figure 4 Creating GitHub Actions workflow.

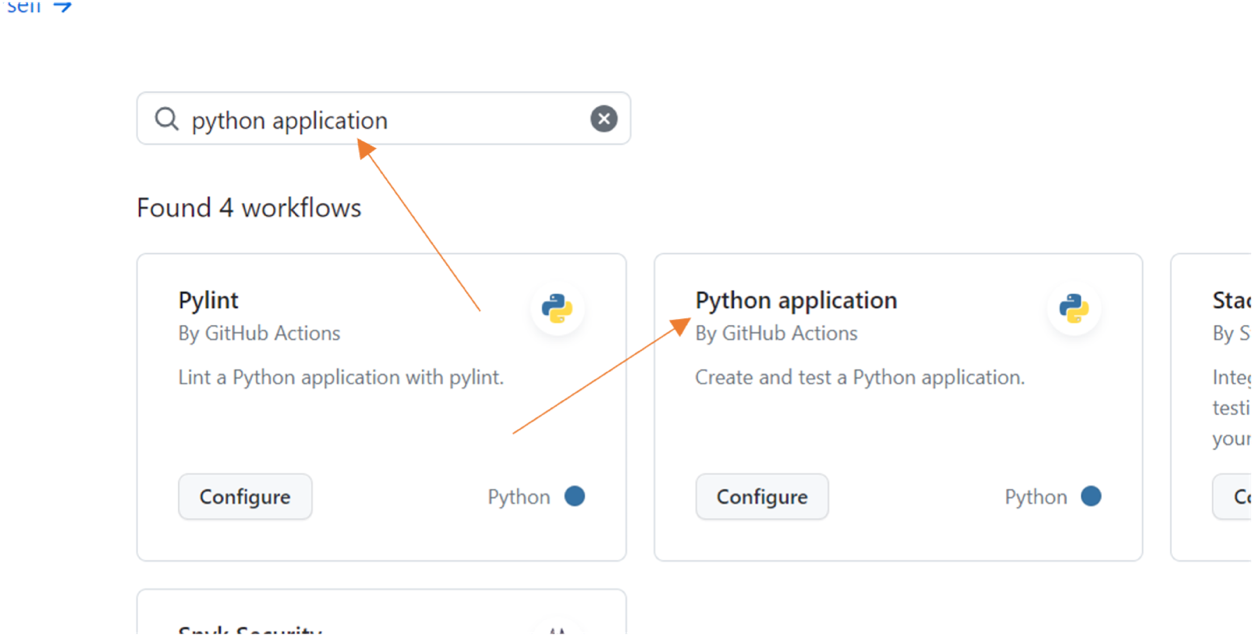


Figure 5 Selecting pre-built Python application workflow.

YAML language was utilised to customize the workflow within the python-app.yml file, specifying the Python test files to be run. This meant that the workflow created would automatically run tests and linting operations with each push to the repository. We can then further configure the workflow using the YAML language on the python-app.yml file:

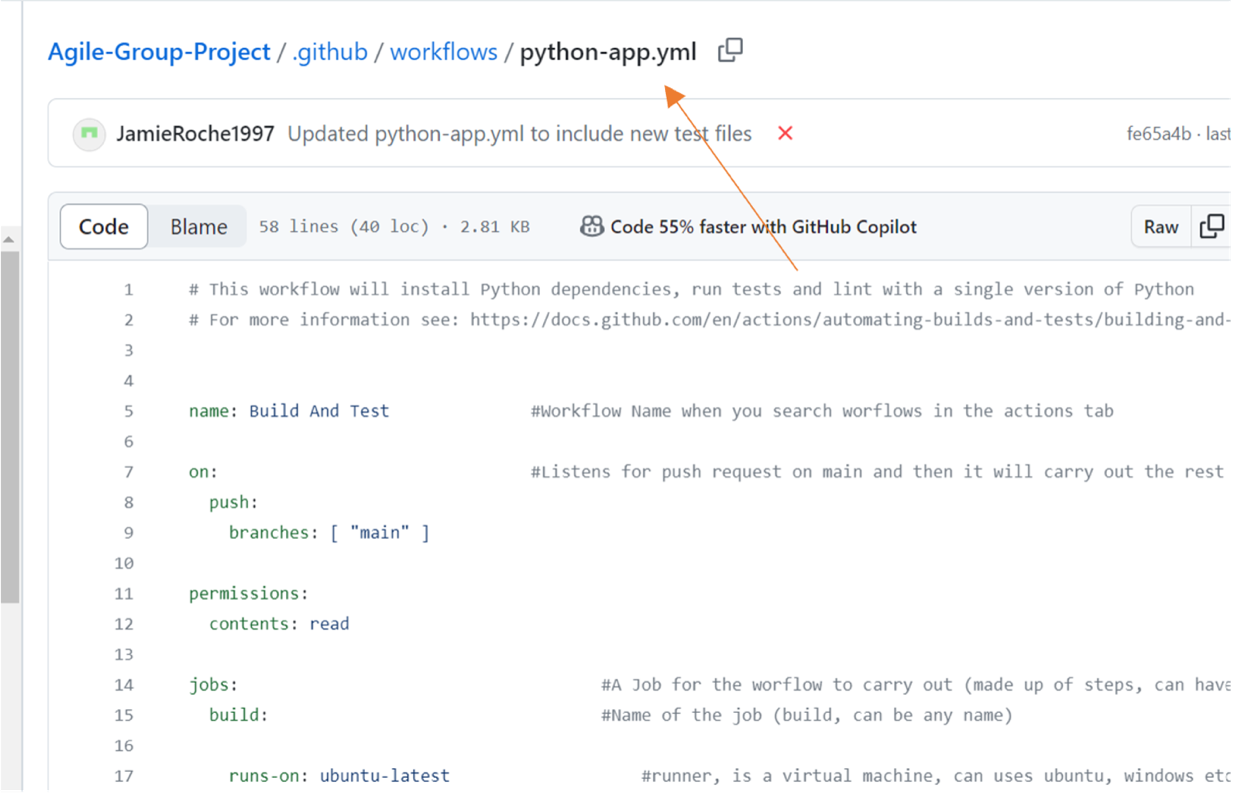


Figure 6 Showing python-app.yml file.

## Initial Code Review and Task Breakdown

Upon downloading the necessary text files and the Python codebase, the team engaged in a collaborative effort to address initial syntax errors and ensure code compilation. Each team member actively participated in fixing these errors, fostering a shared understanding of the codebase and its structure.

Following the initial code review, the team decided to break down the menu options into separate tasks for comprehensive testing and issue identification. This approach allowed for a focused examination of each functionality, aiding in the systematic identification of potential issues.

## Testing Functionality and Issue Identification

Team members initially tested the functionality of each menu option, noting any discrepancies or unexpected behaviour. This phase aimed to identify issues or bugs present in the codebase, which were tracked in the PartialCodeFaults document. The collaborative nature of this process facilitated a thorough examination of the codebase.

## Issue Resolution and Implementation of Fixes

Once issues were identified, the team collectively brainstormed and proposed solutions to address the challenges encountered. This collaborative problem-solving approach ensured that each team member contributed insights and suggestions.

Once we came up with solutions, the team implemented fixes for the identified issues. The process involved modifying the codebase to align with the desired functionality and resolving any underlying problems. Continuous communication and collaboration were paramount during this phase to ensure a consistent and coherent approach to issue resolution.

## Test File Creation

Upon successful implementation of fixes for each menu option, the team created test files to validate the functionality and robustness of the code. This phase presented challenges that extended the expected timeline for task completion. The creation of the test files demanded attention to detail and thorough testing scenarios to accurately test the expected functionality of each menu option.

After completing an assigned task, the team updated the python-app.yml file and pushed the changes to GitHub. This triggered the automatic build and testing of the Python codebase, providing immediate feedback on the success or failure of the push:

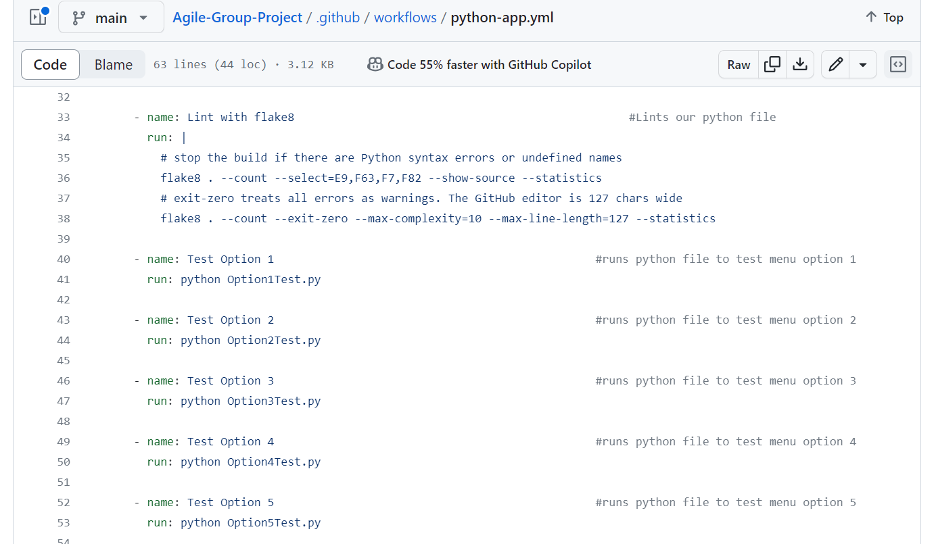


Figure 7 Showing code used to run automated tests.

For each following push, the previous tests (eg. Option1Test.py) were run on each Python build. GitHub Actions tells the user if a successful or unsuccessful push has been made as seen below.

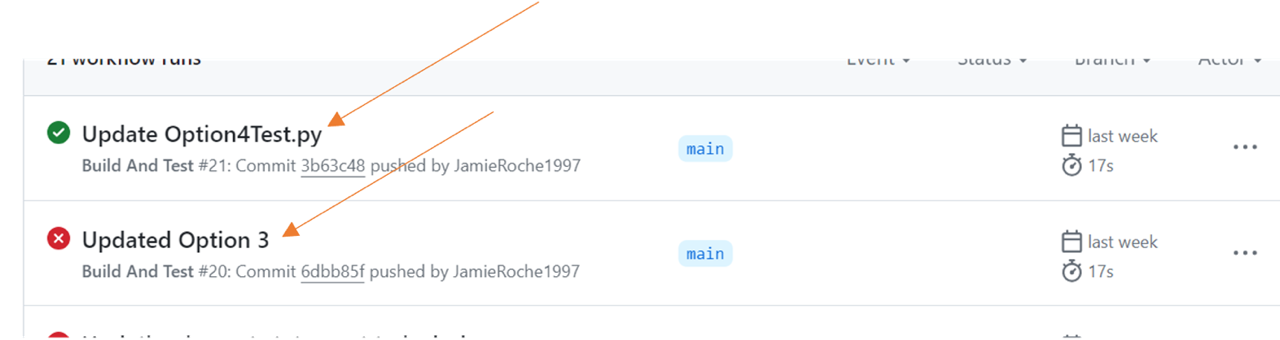


Figure 8 Showing a successful and failed test.

# Experience

## Usefulness

GitHub Actions proved highly beneficial for our team, offering:

**Automated Testing Advantage:** Swift assessment of code changes, ensuring consistent quality. Immediate feedback on the pull requests or commits aided in catching issues early, reducing bugs and enhancing code reliability. This continuous testing loop maintains code health and stability throughout the development lifecycle.

**Workflow Flexibility:** YAML-based workflows provided flexibility in defining complex processes. Trigger mechanisms, such as push events or scheduled runs, ensured workflows kicked off precisely when needed, streamlining development and deployment cycles.

**Pre-built Actions and Integrations:** The extensive library of pre-built actions and integrations reduced development time and effort. Features like secrets management, detailed logs, and artefact retention contributed to a secure and transparent development environment.

## Ease of Use

Following the initial configuration of GitHub Actions, the team seamlessly integrated Python tests into the workflow. The process of pushing a test file to GitHub triggered automatic build and testing of the entire codebase, simplifying the continuous integration process. The familiar GitHub push workflow remained intact, with the only deviation occurring when a test failed, requiring resolution before initiating another push.

## Challenges

Despite the benefits, the team encountered challenges:

* **YAML Syntax Learning Curve:** Learning the syntax of YAML proved challenging initially. Understanding how to integrate automated test files to run on pushing code to the GitHub repository required additional effort.
* **Updating YAML File for Tests:** Initially, the assumption that running "pytest" would automatically find and run all test files proved incorrect. The team had to update the YAML file to specify each test file that needed to run with every push.
* **Test File Creation Challenges:** The creation of test files presented unexpected challenges. This phase demanded considerable time and effort, sometimes surpassing the time invested in fixing the code itself. These challenges showed the importance of thorough testing practices and the complexities involved in creating effective test files to ensure the reliability of the codebase.

# Conclusion

In conclusion, this project provided valuable insights into the integration of GitHub Actions into our development workflow. Despite initial challenges, the benefits of automated testing, workflow flexibility, and pre-built actions significantly contributed to our collaborative coding efforts. GitHub Actions not only enhanced our code quality but also streamlined our development processes, demonstrating its effectiveness in a team environment. The experience gained from this project has reinforced the importance of continuous integration in ensuring code health and stability throughout the development lifecycle.