Ronan Smyth - STUDENT

L00176857  Computer Science

Devops Lab Folio – Part 1

Contents

[1. Introduction 3](#_Toc185420634)

[1.1. Aims 4](#_Toc185420635)

[2. Git 5](#_Toc185420637)

[3. GitHub 6](#_Toc185420638)

[3.1. Merging 6](#_Toc185420641)

[3.2. Tags and Milestones 6](#_Toc185420642)

[3.3. Issues 7](#_Toc185420648)

[3.4. Boards 7](#_Toc185420650)

[4. Software Development Lifecycle 9](#_Toc185420651)

[4.1. Stages 9](#_Toc185420652)

[4.2. Agile 9](#_Toc185420653)

[4.3. Stand-ups, Reviews and Retrospectives 10](#_Toc185420654)

[5. Autodocumentation 11](#_Toc185420655)

[5.1. JavaDoc 12](#_Toc185420656)

[5.2. Junit Testing with Maven 12](#_Toc185420657)

[6. Performance 14](#_Toc185420658)

[7. Testing 14](#_Toc185420659)

[8. Cloud 15](#_Toc185420660)

[9. Security 16](#_Toc185420661)

[10. Conclusion 17](#_Toc185420662)

[Appendix A 19](#_Toc185420663)

[Appendix B 21](#_Toc185420666)

# Introduction

The DevOps methodology, which emerged around 2007, combines practices, tools, and cultural philosophies to streamline and automate the processes between software development and IT operations teams, hence the name, DevOps. The term DevOps merges the concepts of development and operations, signifying the unification of these disciplines into a seamless, continuous process. The focus of DevOps is on empowering teams, facilitating communication between professionals of different fields through collaboration and using technology to automate tasks.

In the project at hand, an element of the Zelda game for Nintendo, a real estate agent for the Kakarico Village, was developed via the creation of a DevOps pipeline. Figure 1 displays the GitHub account where the pipeline has been implemented, which can be found at this URL: <https://github.com/l00176857/KakaricoVillageRealEstateAgent>.

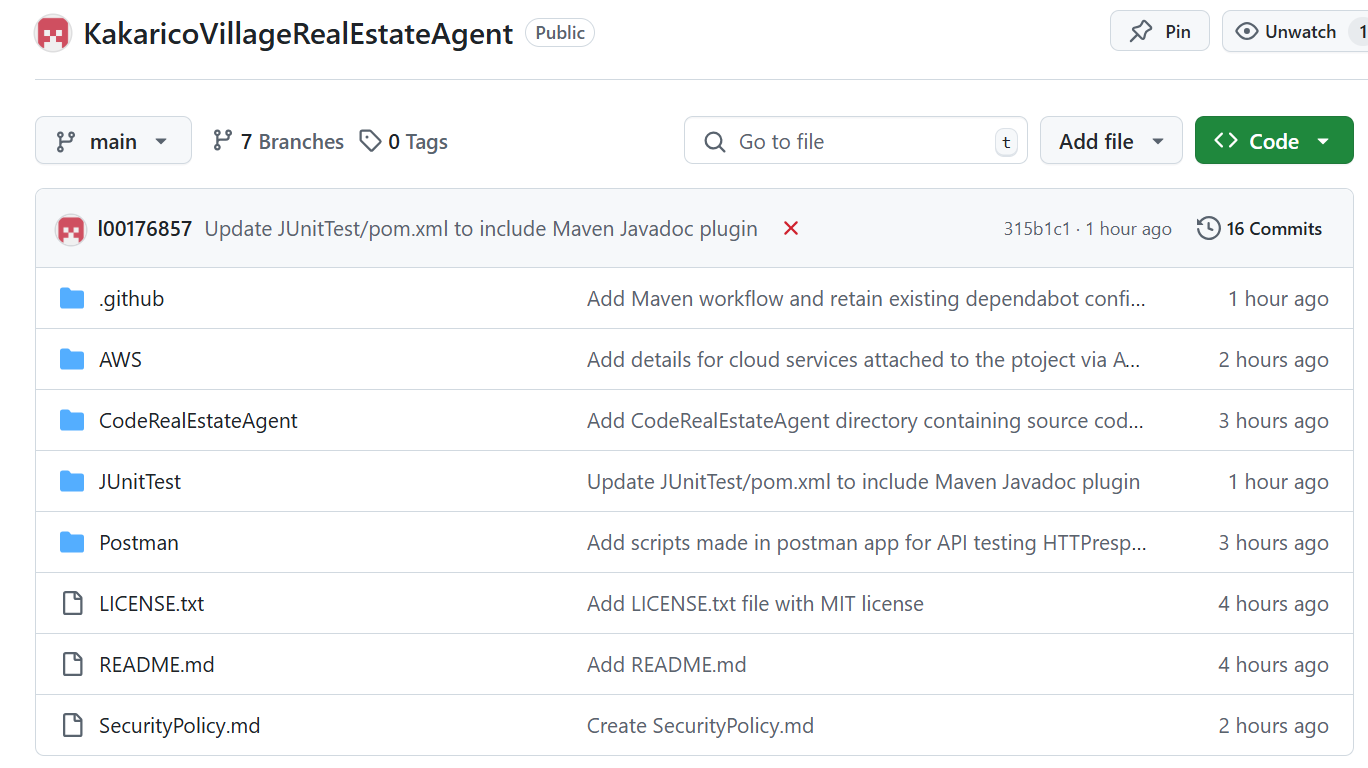


Figure 1‑1: GitHub repository where the project can be found.

## Aims

The following aims were considered in completion of the project.

1. Further investigation into the functionality of Git for management of a local repository for code and documentation.
2. Further experimentation with the functionality of GitHub and interconnectivity features for Git.
3. Research into API testing through the Postman tool.
4. Further research into JUnit testing using Apache Maven.
5. Investigation into how Junit testing and JavaDoc documentation can be automated by integrating Maven with GitHub.
6. Study of the AWS platform for cloud services.

# Git

A computer screen shot of white text

Description automatically generated

Figure 2‑1: Initialise local Git repository and remotely connect to GitHub.

In figure 2-1, by following early labs, a local Git repository was created and linked to the GitHub Repository (repo) for the project. A README.md file was created and committed to the repo.

A computer screen shot of a computer program

Description automatically generated

Figure 2‑2: Git status and logs.

Git status is used to show any staged files waiting to be committed to the repo. Git log then shows a record of files that have been committed to the repo. In addition, various commands can be added to status and log to provide more detailed or specific information.

A black screen with white text

Description automatically generated

Figure 2‑3: Git branch command demonstrates branching strategy.

The git branch command demonstrates the branching strategy which will be discussed in closer detail in the following chapter.

# GitHub

GitHub is the distributed version of Git, which is local, which allows teams to collaborate from different locations. While Git pulls branches from Github so that developers can work on projects on their own machine, the product can then be pushed back up to GitHub and merged to the base data enabling integration of new work with existing work. This allows developers to add new structures to the package or change what is already there.



## Merging

The system allows for different types of merging when developers add or edit code including:

1. Merge Commit: Combining all commits from a branch into the main branch, this approach preserves the complete history of changes.
2. Squash and Merge: This condenses all commits from the branch into a single commit before merging into the base branch resulting in a reduced Git history but can be clearer.
3. Rebase and Merge: Rebasing the commits from the branch onto the base branch without creating a merge commit leads to a linear project history. However, rewriting the commit history may resulting less clarity.

In this project,

## Tags and Milestones

Tags are used to mark specific points in the Git history of issues and pull requests, often for release versions. Milestones, in GitHub, are more significant than tags representing a grouping of related work which helps teams prioritise and work towards a common objective.

In this project, neither tags nor milestones have been applies due to the ad hoc nature of the information making up the project. For this reason, no version of the project ever existed past version one and no milestone was foreseeable by the researcher.



## Issues

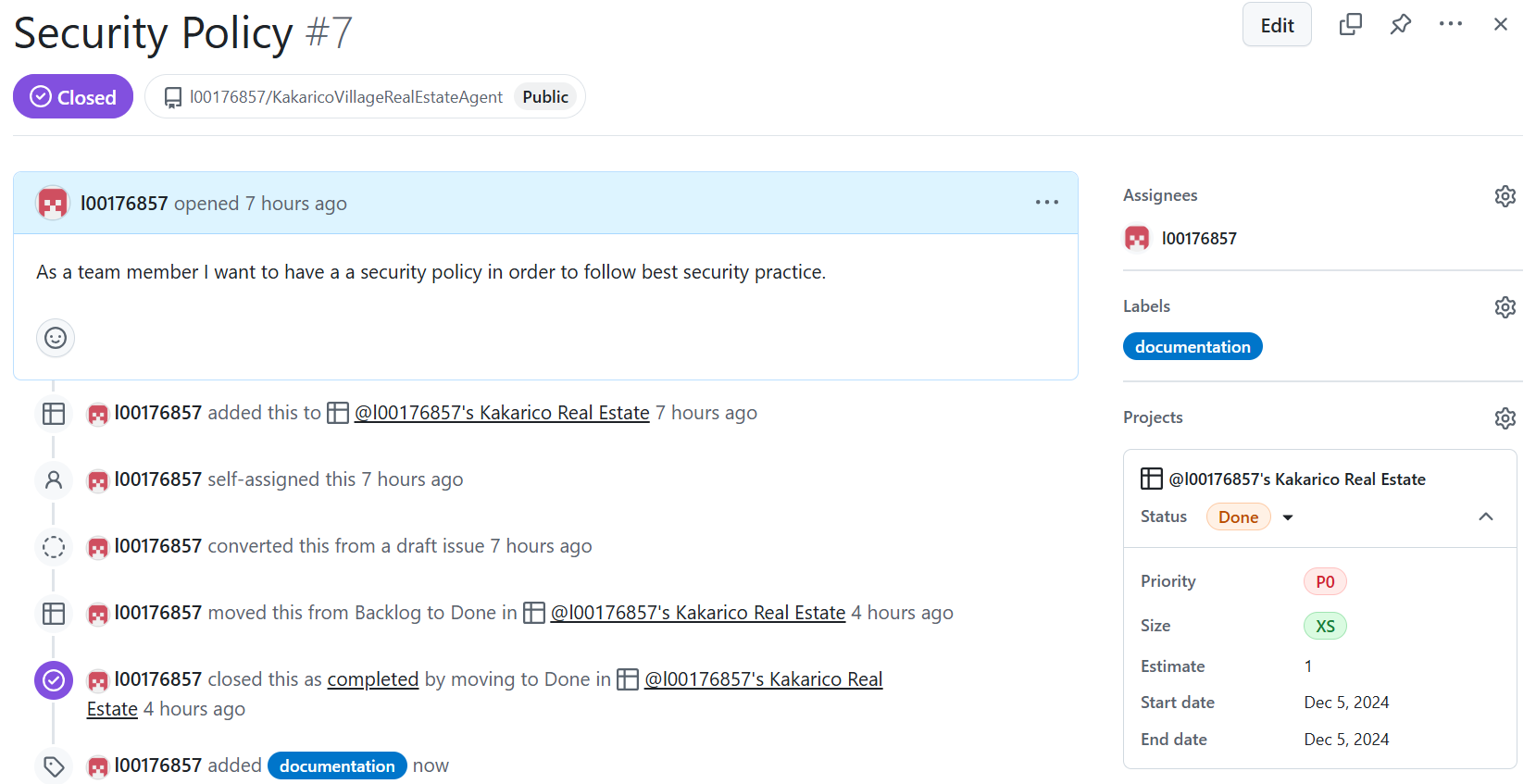


Figure 3‑1: An issue created to communicate the creation of a security policy.

Issues, as in figure 3-1, are made from user stories and represent a formal task in agile teams. User stories are collected during requirements gathering, which are statements about what the system must, should and could do when the related task is completed. The user stories can be collected from agile team members, product owners and users of the system and represent those parties’ expectations of the system.

User stories are transformed into issues by the agile team and on doing so, a list of tasks to complete is created. The completion of all of the tasks on that list marks the completion of the project.

Issues are assigned team members to work on them. Moreover, they will be prioritised and contain labels and time estimations indicative of their function and difficulty, respectively.



## Boards

A screenshot of a chat

Description automatically generated

Figure 3‑2: A Kanban boards containing issues for the project.

Figure 3-2 shows the Kanban board for this project. The Kanban board is broken into what are known as Swim-Lanes, of which there are typically five though it is at the \_\_\_\_\_\_ of the agile team. In the above board, the Backlog and Ready swim-lanes cannot be seen and there are currently no issues in those lanes.

The backlog stores the total number of issues to be addressed while those prioritised are moved to the ready lane first to be selected by team members. Issues progress to the In-Progress Lane, visible in figure 3-2, when the task has been adopted by a team member or members. After tasks are completed, they move to the Review Lane where work done should be reviewed by as many other team members as time allows, though at least one is required. When the work has been reviewed it will either by moved to the Done Lane, and considered done, or sent back to the backlog if the task is not complete.

The above project shows the current state of tasks where three are in progress, actively being worked on, three are in review, awaiting a reviewer and five have been completed satisfactorily.

Kanban boards are an excellent way to communicate the progress of a project visually and are equally effective for agile team members in person or geographically dispersed, as they are for stakeholders outside the development team.

Branching Strategy

Being a small, individually-manned project, the branching strategy chosen is a feature-based branching strategy, where branches are created for specific tasks and features. The ad hoc nature of the course which led to the project also contributed to this decision.

The branch structure is made up of five branches described below:

1. Main: The main production branch.
2. Testing: A branch for testing purposes used for integrating and validating features before merging them into main.
3. Code-review: A branch for code reviews or peer collaboration before merging to main or other branches. This branch was chosen as theoretically work in an agile team should always be reviewed although there being only one developer on the team, it was never intended to be used.
4. Api-Test: Abranch created for testing API-specific functionality. The testing branch was to be used for JUnit testing and the automation thereof. Since a lab was completed on Postman and API testing, which was quite unrelated to the rest of the project, a decision to use a separate branch for this.
5. Cloud: Afeature branch dedicated to cloud-related work or configurations. Similar to the api-Test branch, work was done in AWS which did not relate to other aspects of the project and so a stand-alone branch was chosen to represent this work.

With this branching strategy, once the work is complete, branches are merged into a main branch owing to the fact that each branch serves an isolated purpose.

# Software Development Lifecycle

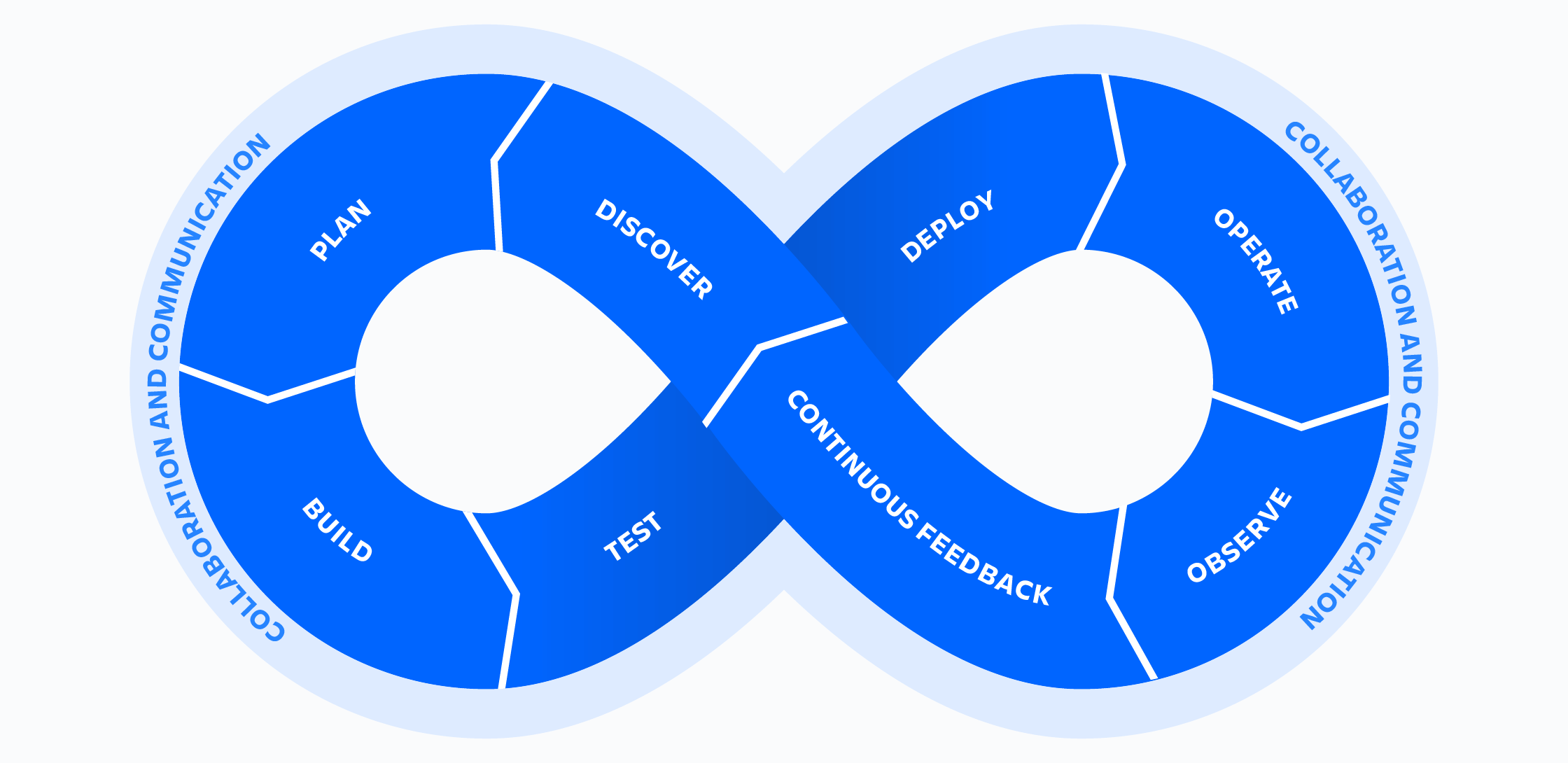
The DevOps lifecycle encompasses eight phases in which teams work together communicating to ensure alignment, maintain momentum, and uphold quality standards.

Figure 4‑1: DevOps Lifecycle (Atlassian, 2024).

## Stages

The eight stages of the DevOps lifecycle are broken into four on the left side of figure 4-1, which represents the part of the lifecycle for which the development team are responsible, and four stages on the right, for which the users, or operators, of the system are responsible.

DevOps is the agile, and so iterative and incremental, collaboration of a development team with the operators of a system. The infinity symbol is used to symbolise the continuous nature of DevOps whereby developers discover requirements and user stories before planning, building and testing new systems and improvements. Users then deploy, operate, observe pros and cons and provide feedback in order for developers to improve the system. The infinity symbol signifies the never-ending nature of this process which in theory continues until the system’s End-Of-Life (EOL). Knowledge gained through continuous observation and feedback during the DevOps lifecycle then benefits the creation of the replacement system.

## Agile

DevOps is an agile methodology and so work is done in two-to-four-week sprints by small teams of knowledgeable and motivated individuals who are, ideally, collocated. It is unclear whether this can be applied to this particular research project due to the inherent constraints.

A DevOps team includes developers and IT operations working collaboratively throughout the product lifecycle until EOL. This is done to increase the delivery speed and the quality of software. It’s a so-called cultural shift, that has significantly improved the experience of teams and the organizations they work for.

DevOps teams use tools to automate and accelerate processes, which helps to increase reliability. A DevOps toolchain helps teams tackle important DevOps fundamentals including continuous integration, continuous delivery, automation, and collaboration.

## Stand-ups, Reviews and Retrospectives

As communication is so fundamental to agile, short meetings, known as stand-ups, occur daily for teams to touch base. During this project, this was done once in person. Team members, in turn, address what they had done, what will do next and any difficulties they are having. Furthermore, the stand-up format was practiced once weekly on the messaging platform slack in the form of a tool called Polly, which address the same information in the context of a messaging application. The stand up was practiced on November 13th.

Reviews are meetings attended by the product owner and the whole development team which demonstrate the user stories that have been completed during a sprint. It is suggested that these meetings are held when 20% of development is complete. At these meetings, the product owner will present the release goal, the sprint goal, and any new features at a high level. Each member of the development team is allocated 5 minutes to present what it has completed during the sprint. The Review and retrospective took place with head of department on December 4th.

Retrospectives are meetings that provide a safe space for a team to reflect on what works well and what does not so that improvements can be made. They can be structured around various themes or involve images as can be seen in figure 4-2.



Figure 4‑2: Visual aid used for retrospectives.

The review and retrospective for this project took place at the same time, which would not be an ideal way to approach these types of team events. Due to requirements being and remaining quite unclear, both the parts of the product owner and team members were largely for show. In addition, teams for the review and retrospective were formed just before the meetings took place and weren’t actually working on any project as a team, very much in line with the ad hoc progression of the course itself. Consequently, there was little possibility of cohesion between team members in terms of communication and as such explanations were weak by and large and some elements were impossible like final kudos and creating a theme-based retrospective.

That said, the proceedings did serve to give some first-hand experience into the format of the meeting.

# Autodocumentation

A screenshot of a computer

Description automatically generated

Figure 5‑1: .GitHub folder containing workflows for automation.

Apache Maven, among other things, can be used to achieve automated Junit testing of code pushed or pulled into one, some or all of the branches in GitHub. When Maven is installed a .yml file is produced to determine its behaviour. This file can be integrated with instructions for automatic testing of code and/or automation of documentation via JavaDoc. Figure 5-1 shows implementation of the .tml file in the .github/workflows directory in the GitHub repository whereas 5-2 demonstrates the pom.xml file for this instance of Maven.

A screen shot of a computer program

Description automatically generated

Figure 5‑2: pom.xml in the Maven repository.

Lines three through eleven in 5-2 show rules implementing automation on creating push or pull requests in the branches main and testing, which is where the main body of code would traverse.

## JavaDoc



Figure 5‑3: JavaDoc plug-in in the maven.yml file.

Research online seemed to indicate that inclusion of these lines of code would be the only necessary step in applying JavaDocs to automatically. The applied implementation has not been tested due to lack of understanding and time constraints

* 1. Junit Testing with Maven

A screenshot of a computer program

Description automatically generated

Figure 5‑4: .gitignore file added to Maven enabled repo so as to avoid pushing unwanted files.

A screen shot of a computer program

Description automatically generated

Figure 5‑5: Unit testing in the implemented in maven.yml file.

A computer screen with text

Description automatically generated

Figure 5‑6: Successful command line execution of Junit test using Maven.

The system conducted a series of tests on the KakarikoRealEstateAgent class to verify its functionality. First, the listProperty() method was tested to ensure that the agent could successfully list a property, such as displaying a message about listing "Cozy Cottage" for 500 rupees. Next, the sellProperty() method was evaluated by simulating the sale of a property to a buyer (e.g., "Link" purchasing for 450 rupees) and confirming that the agent’s total earnings were updated correctly. The offerDiscount() method was also tested by simulating a discount of 50 rupees on listed properties and checking for the appropriate message. Lastly, the system verified the agent’s details, such as name, agency name, and specialty, to ensure they were correctly set and retrievable. Overall, these tests confirmed that the agent can list and sell properties, offer discounts, and maintain accurate internal attributes like total earnings and personal details. The printed messages served as validation of expected behaviour, and the absence of errors indicates that the methods are functioning correctly.

# Performance

A screenshot of a computer

Description automatically generated

Figure 15: Codewind Profiler Tool.

Attempts were made to install Codewind profiler tool through VS code and JConsole through the command line.

In both cases technical issues and time constraints prohibited uptake of the performance tools.

A screen shot of a computer screen

Description automatically generated

Figure 16: Attempt to implement JConsole through the command line.

# Testing

Firstly, the Postman tool is being used to test API responsiveness as evidenced in figure 6-1. Two scripts were written in postman and pushed to the GitHub repo where they can now be found.

A screenshot of a computer

Description automatically generated

Figure 17: Postman script to check valid response..

A screenshot of a computer

Description automatically generated

Figure 18: Postman script to check response time.

# Cloud

AWS has been employed to give cloud functionality to the theoretical team as evidenced in figure 6-2. Several ec2instances and cloud formations were implemented and the .yaml files for each can be found in GitHub.

A computer screen shot of a program

Description automatically generated

Figure 8‑1: Cloud formations .JSON file.

A screenshot of a computer

Description automatically generated

Figure 8‑2: EC2Instance available in main branch in GitHub.

# Security

GitHub integrates several security features. Figure 7-1 and 7-2 show implementation of the Dependabot which checks dependencies, that is libraries and API used in the code, for out of date or vulnerable components.

A screenshot of a computer

Description automatically generated

Figure 9‑1: Dependabot .yml governing schedule and scope.

A screenshot of a computer

Description automatically generated

Figure 9‑2: Dependabot configured in GitHub.

A screenshot of a computer

Description automatically generated

Figure 9‑3:Secret scanning for hardcoded secrets in the code.

# Conclusion

It has been learned that in the creation of a DevOps pipeline, developers and users of a system continuously collaborate in the improvement that system until its EOL. The eight stages in the DevOps lifecycle run continuously with the feedback stage from the previous agile iteration feeding the planning stage in the next.

Git and GitHub make up two sides of a coin whereby Git is used locally and GitHub allows for distributed teams sharing and changing code. The version control and review functionality are invaluable in distributed environments as is the ability to implement branching strategies and the merge options which go with that. Altogether, an enormous amount of control is provided to the developer who knows how to harness it well.

In addition to version control features, GitHub also facilitates communication and project management tools such as the Kanban board and issues. This is especially useful when working in distributed teams as all team members can be constantly up to date with team requirements and workflow. It is, however, also very useful for communicating with product owners and other stakeholders.

Not only can unit testing be integrated into GitHub, but it can also be automated which is a powerful tool for those whose work involves repetitive tests of a similar nature. Testing makes up a great deal of the total development process, occupying an entire swim-lane in the Kanban board as testament of such. Automating testing saves developer time and organisation money. Maven was used in this project as the Junit testing tool and screenshots are seen throughout this document indicating the automation thereof.

It is not just testing that can be automated, another seriously time-consuming and often costly software development activity can also be automated, namely, documentation. JavaDoc was used in conjunction with Maven on this project to automate the production of documentation associated with push or pull requests around the main and testing folders, where code would be passing.

Lastly, performance enhancing tools can be used with DevOps projects to facilitate sharing, testing and communication. One such tools is the AWS cloud service, which has been employed to open a number of EC2Instances for the project as well as a number of cloud formations which can be used. Another tool used during production is the Postman tool where scripts have been written which may lead to the automation of API testing.

On the whole, the development of a DevOps pipeline provides developers with the version control and information sharing systems to work on distributed teams in an efficient way. It aims to automate as many of the repetitive tasks as possible through the pipeline. It also aims to facilitate communication between development teams and operations teams for seamless collaboration.

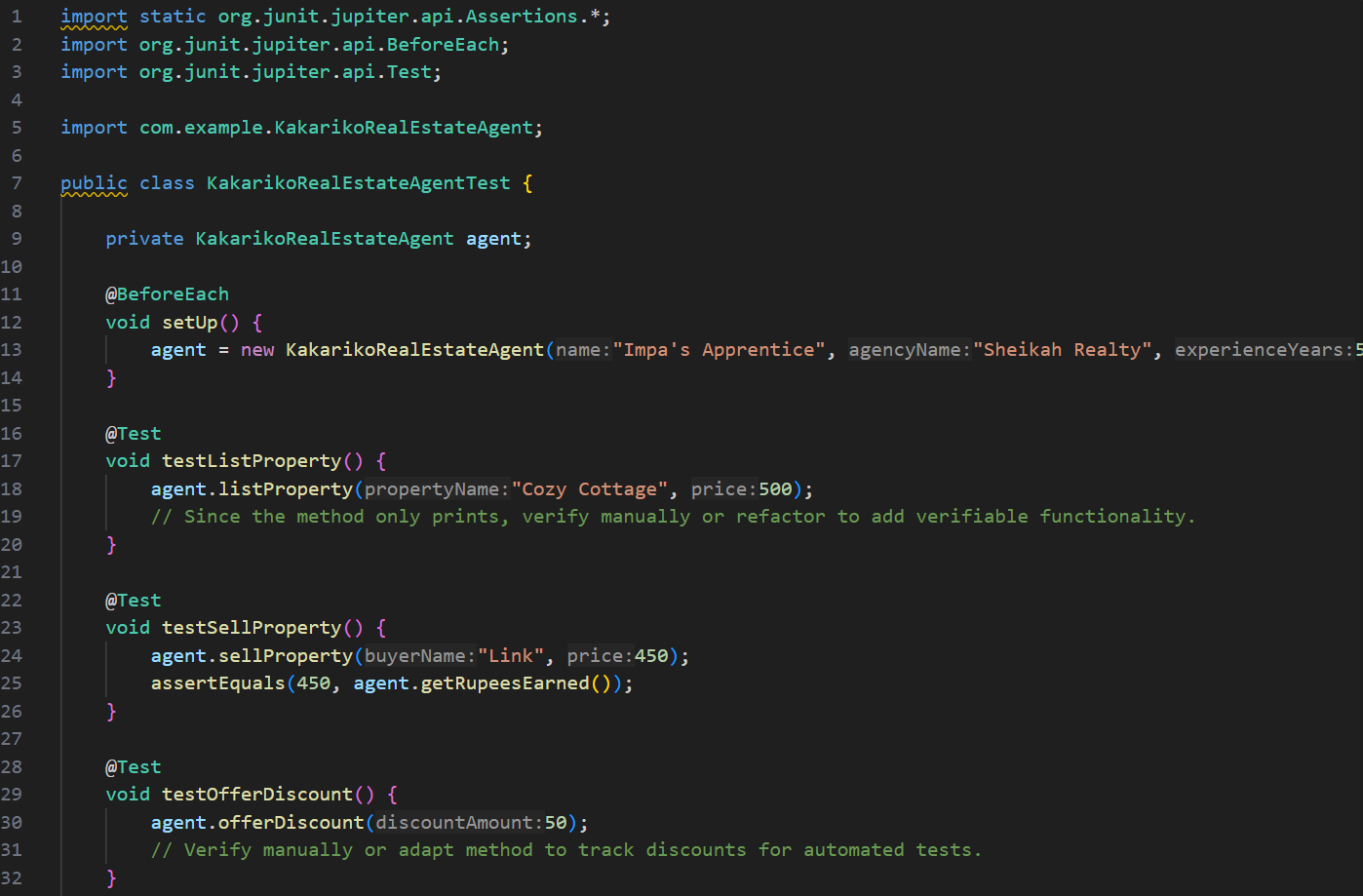
# Appendix A

# A screen shot of a computer program Description automatically generatedA screen shot of a computer program Description automatically generated

# A screen shot of a computer code Description automatically generated

Figure 0‑1: Kakarico real estate agent Java class.

# Appendix B

A computer code on a black background

Description automatically generated

Figure 0‑1: Unit test for Kakarico real estate agent class.