Software Design Specification

1. Introduction

1.1. Overview - Purpose of system

The purpose of building a CI/CD pipeline is to support **DevOps** [1] — the amalgamation of cultural philosophies, practices, and tools that speeds up an organization's ability to deliver applications and services at high velocity. This allows a team to develop and improve products quicker than organizations using traditional software development and infrastructure management processes.

Working together with **IBM**, we plan to build a GitOps pipeline to build and deploy an application to OpenShift. This automation will include performing static analysis, unit testing, code coverage, packaging in the form of a container using **Podman** [2] and deploying onto **Red Hat OpenShift** [3].

1.2. Scope

In Scope	• Automated building • Automated testing & static program analysis • Automated containerisation • Automated deployment to Red Hat OpenShift • Creating a web application to test the pipeline of the code.
Out of Scope	• Setting up a messaging system

1.3. Definitions, abbreviations

- **CI/CD:** Continuous Integration/Continuous Delivery.
- **CI/CD pipeline:** This helps us to automate the steps in the software design process. The pipeline builds code, runs tests (CI), and safely deploys a new version of the application(CD). Automated pipelines remove manual errors, provide standardized feedback loops to developers, and enable fast product integration.
- **Standardisation:** This is a common format of a document or file that is used by one or more software developers while working on the same computer program. Software standards allow for the interoperability between different programs created by different developers.
- **Containerisation:** The process of packaging an application along with its required libraries, frameworks, and configuration files together so that it can be run in various computing environments efficiently.
- **Deployment:** This refers to the process of running an application on a server or device.
- **Image:** an immutable file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

Software Design Specification

2. System Design

2.1. Design Overview

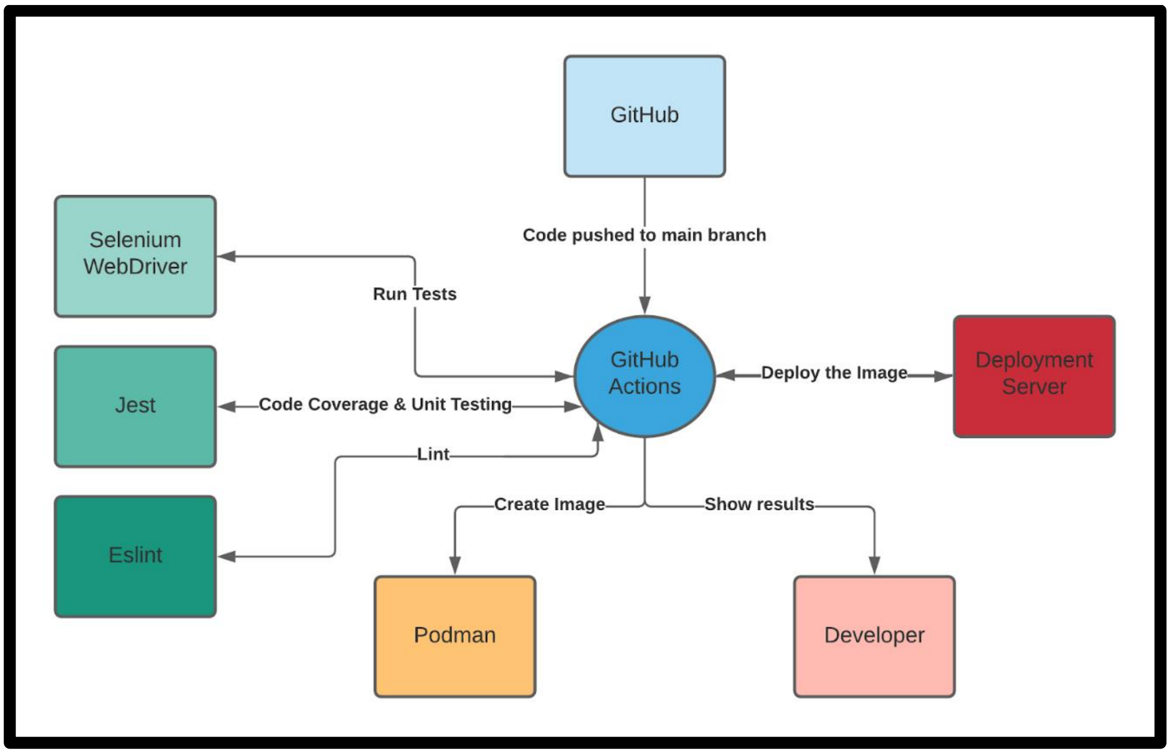
Our design is for a fully functional **CI/CD pipeline** and a **simple application** to demonstrate the use of this pipeline. This will allow developers to write, commit, and push code into a repository, and the code will continuously be built and tested.

2.1.1. High-level overview of how the system is implemented, what tools, frameworks and languages are used etc.

The application we will use to demonstrate out pipeline will be a simple application showing **Covid-19** [4] **statistics on a dashboard**. The version-control will be handled with **Git** [5] to enable collaboration of work and track modifications to the files and we will use **GitHub Actions** [6] to perform Continuous Integration. For testing, we will be using **Selenium Webdriver** [7]. For code coverage and static unit testing, we will be using **Jest** [8]. For linting, we will use **Eslint** [9]**.** For containerization, we will use **Podman.** There will not be a database, it will be replaced with an **external API - Rapid API** [10] for demonstration. Finally, the application will be deployed onto **Openshift.** This simple application will be a Webapp made with **Vue.js** [11] for the front-end, and **Node.js** [12] for the back-end.

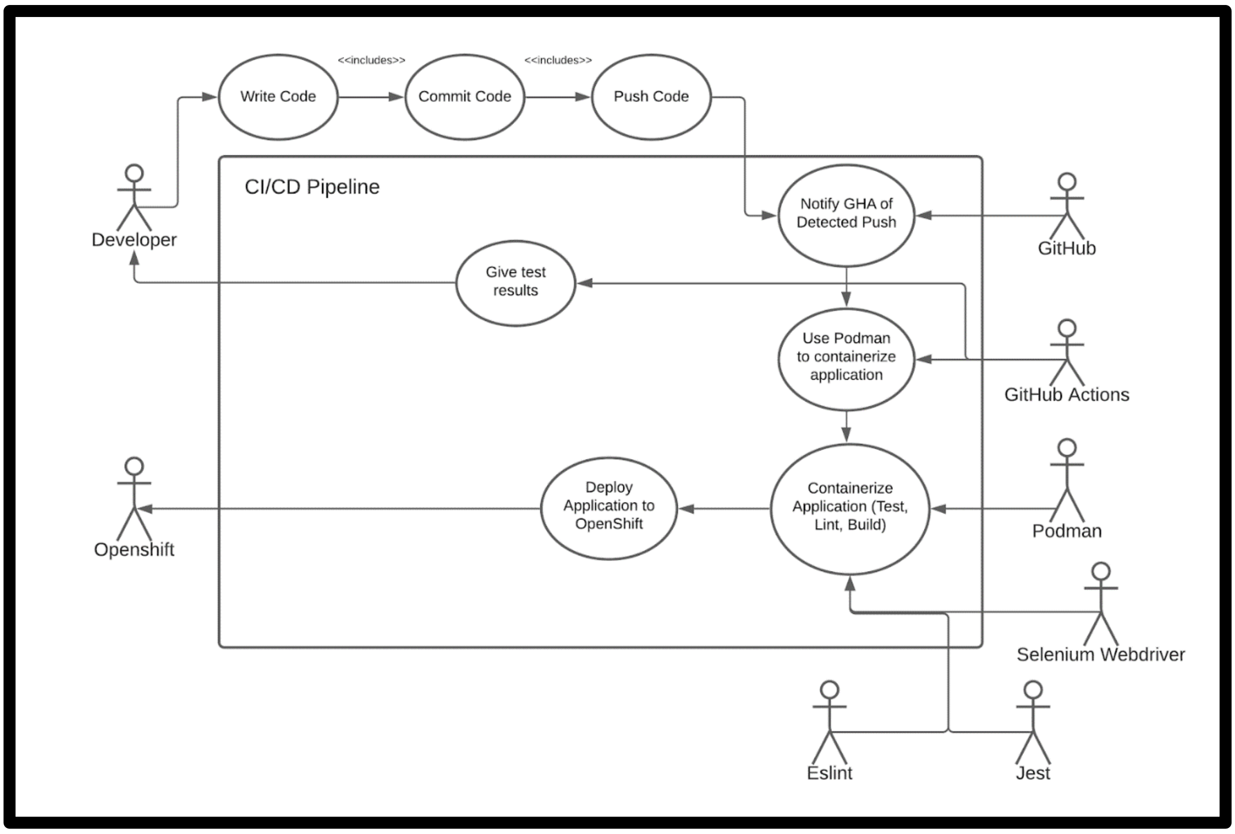
2.2. System Design Models

2.2.1. System Context



Software Design Specification

2.2.2. Use cases (from Requirements)



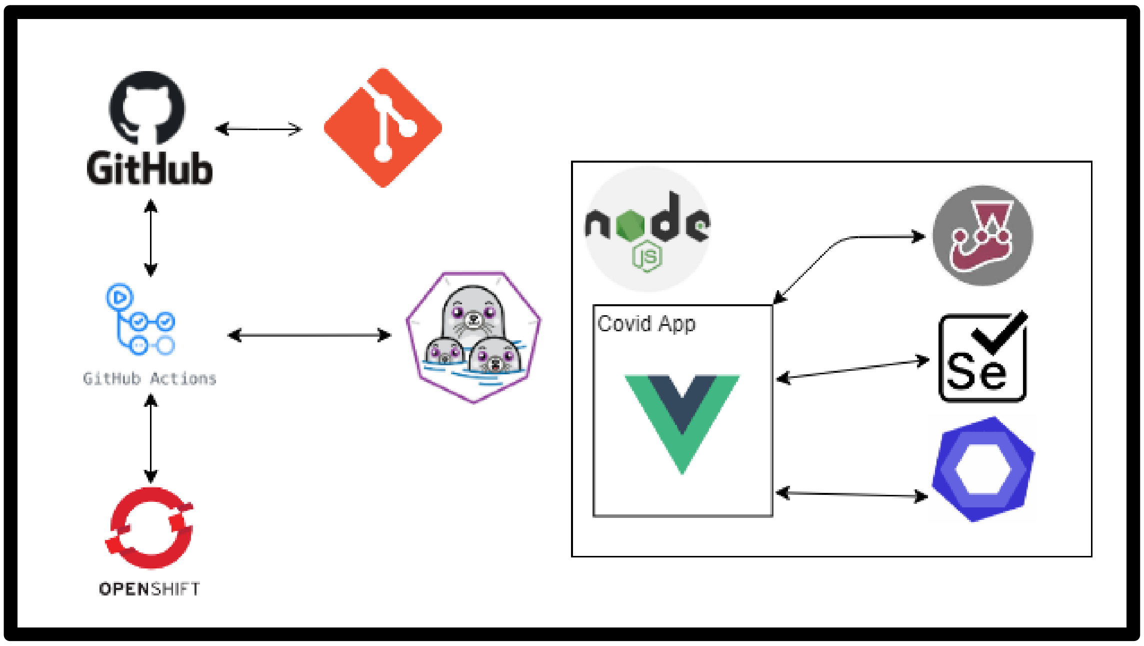
Name: Write, Commit, Push Code **Participating Actor:** Developer **Entry Condition:** The developer must change the code on their local machine. **Exit Condition:** The developer has successfully written, committed the code into a git repository, triggering the CI/CD pipeline. **Normal Scenario:** 1. The developer writes their code to the best of their ability. 2. The developer commits their code that they are satisfied with. 3. The developer pushes their code onto the code repository. **Error Scenario:**	**Name:** Notify GitHub Actions of Detected Push **Participating Actor:** GitHub **Entry Condition:** The developer must have pushed code to the code repository for GitHub to notify GitHub Actions. **Exit Condition:** Github Actions has been successfully notified, to kick off the CI build. **Normal Scenario:** 1. The developer pushes their code onto the code repository. 2. GitHub recognizes this push and notifies GitHub Actions. 3. GitHub Actions receives a notification from GitHub and starts the CI build. **Error Scenario:**

Software Design Specification

• The developer tries to push without a commit beforehand.	• GitHub notifying GitHub Actions when there was no push to the code repository.
Name: Use Podman to Containerize Application **Participating Actor:** GitHub Actions **Entry Condition:** GitHub has successfully notified GitHub Actions of a code push. **Exit Condition:** Podman will start to containerize the application. **Normal Scenario:** 1. Github Actions receives a notification from Github. 2. Github Actions notifies Podman to containerize the application. **Error Scenario:** • Trying to notify Podman without a notification of a code push.	**Name:** Containerize Application (Test, Lint, Build) **Participating Actor:** Podman, Selenium Webdriver, Jest, Eslint. **Entry Condition:** Podman was informed to containerize the application. **Exit Condition:** The application will be containerized, and tasks such as testing, linting, and building will be completed. **Normal Scenario:** 1. Podman containerises the application. 2. Testing is done with Selenium Webdriver and code coverage with Jest. 3. Linting is done with Eslint. 4. The application is built. **Error Scenario:** • Containerizing the application without code coverage from Jest.
Name: Deploy Application to Openshift **Participating Actor:** Openshift **Entry Condition:** The application has been containerized. **Exit Condition:** Openshift will manage the containers. **Normal Scenario:** 1. The application has been fully containerized and finished static unit testing. 2. The container will be handled by Openshift. **Error Scenario:**	**Name:** Give test results **Participating Actor:** Github Actions **Entry Condition:** The application has gone through testing, linting, and building. **Exit Condition:** The developer can see if the application has passed all tests and if the application has been built. **Normal Scenario:** 1. The CI/CD pipeline tests the application, and builds the application. 2. The developer gets the results from the tests and from building the application.

• Trying to deploy an application that is not containerized yet.	**Error Scenario:** • Giving test results when no tests and building has occured.

2.2.3. System Architecture



Our system architecture is displayed above and shows how the different components of our project work cohesively as part of a system. We use a git repository on GitHub to store our source code, which is mainly written using Javascript, using Vue.js and Node.js. GitHub Actions recognizes any push to this repository and uses Podman for containerization. After the application is containerized, carry out steps such as building the code, testing the code, and linting it. This is with the help of Selenium Webdriver, Jest, and Eslint.

2.2.3.1 Architectural Decisions

Architectural Decision	Git
Problem Statement	Multiple version control solutions to be chosen from.
Motivations	To enable collaboration of work and track modifications to the files
Alternatives	Subversion
Justification	Distributed, no assets large enough to validate the use of subversion
References	https://git-scm.com/

Software Design Specification

Architectural Decision	Web application (JS, Node.js)
Problem Statement	Multiple applications to be chosen from.
Motivations	To develop a dummy application used in testing the effectiveness of pipelines.
Alternatives	Ruby on Rails, React, Typescript
Justification	Application is not within scope to validate the use of additional frameworks (React/Typescript), could also simulate the agile environment with unplanned changes.
References	https://nodejs.org/en/

Architectural Decision	No database
Problem Statement	Inclusion of database to be chosen from.
Motivations	To decide application structure
Alternatives	Postgres, MySql, MongoDB, GraphQL
Justification	Not within requirements and unnecessarily added complexity, could easily be replaced with Rapid API for demonstration.
References	https://rapidapi.com/marketplace

Software Design Specification

Architectural Decision	Selenium Webdriver
Problem Statement	Multiple testing frameworks to be chosen from.
Motivations	To test code quality
Alternatives	Jest, Puppeteer
Justification	It is extensive (and less limiting), open-sourced, and Neil has extensive experience with it.
References	https://www.selenium.dev/documentation/en/webdriver/

Architectural Decision	GitHub Actions
Problem Statement	Multiple pipelining solutions to be chosen from.
Motivations	To support CI/CD use cases of application development.
Alternatives	Travis, Jenkins, DevOps
Justification	Built-in support by GitHub with documentations covering most of our use cases.
References	https://github.com/features/actions

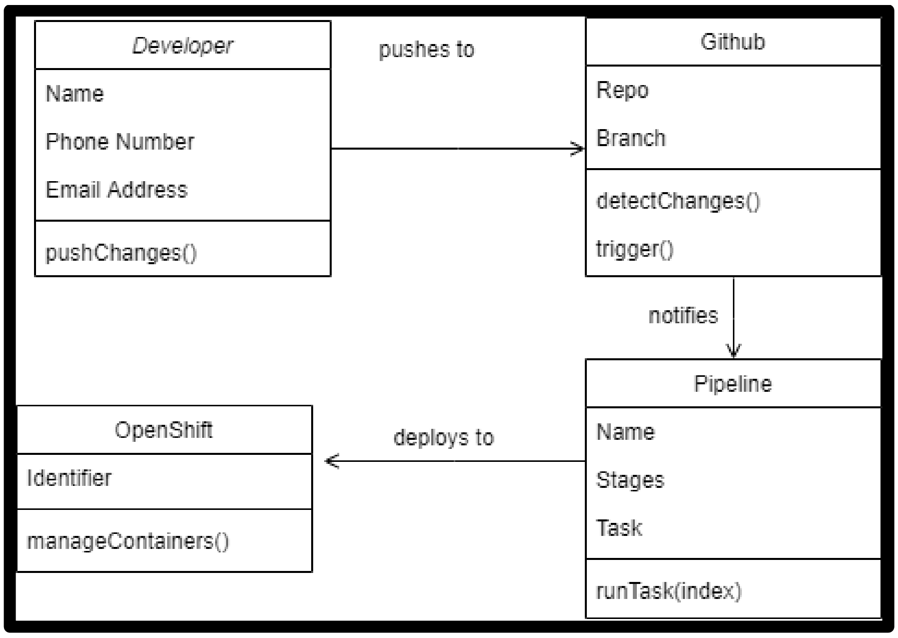
Software Design Specification

Architectural Decision	Vue.js
Problem Statement	Multiple frontend framework to be chosen from.
Motivations	To integrate the frontend framework for easier development.
Alternatives	React, Next.js
Justification	Vue is the middle group from Vanilla JS and React JS, while still providing many benefits and out-of-the-box tooling.
References	https://vuejs.org/

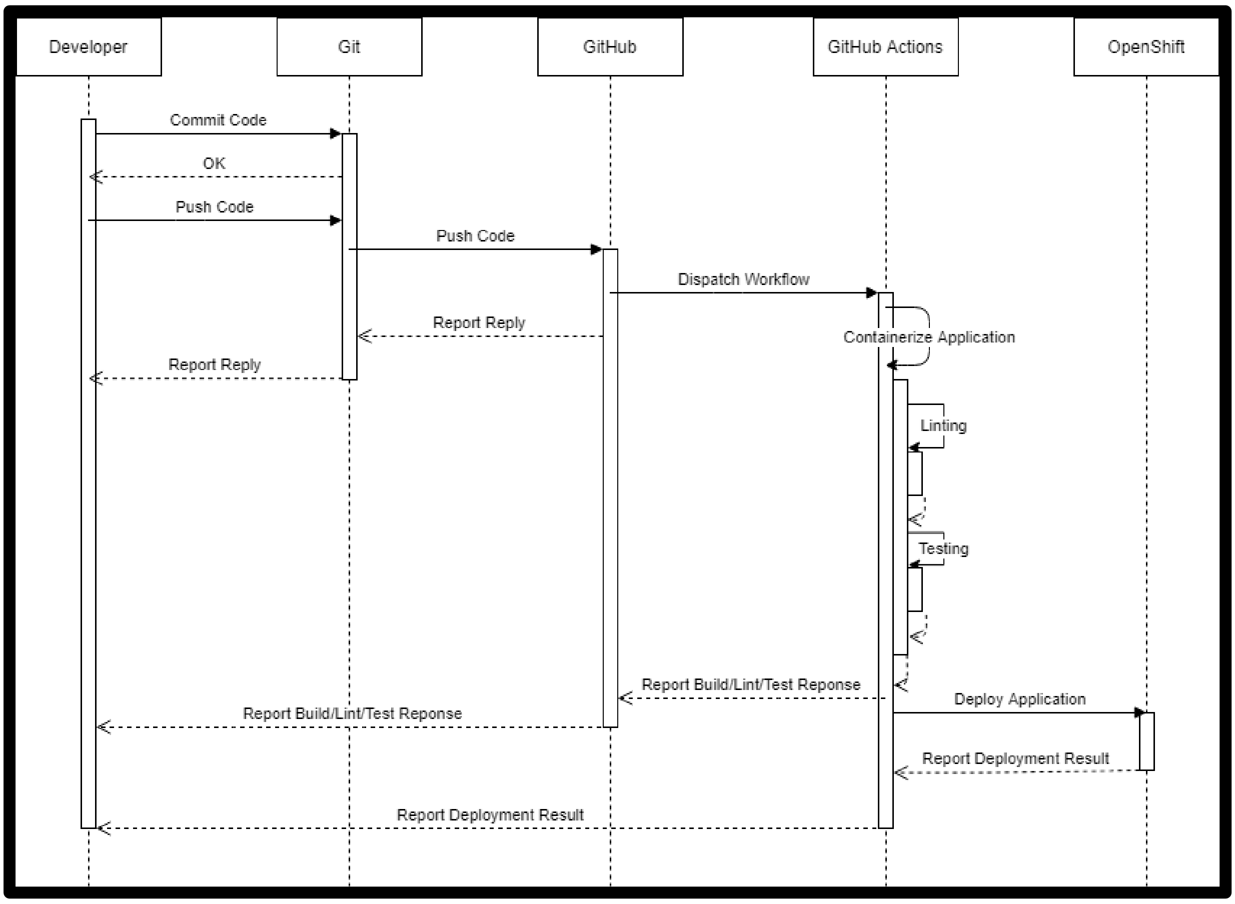
Architectural Decision	Jest
Problem Statement	Webdriver is not suitable for code coverage.
Motivations	Code coverage test.
Alternatives	Cypress, Mocha, Chai
Justification	Jest has the lowest technology overhead and highest capability
References	https://jestjs.io/

Software Design Specification

2.2.4. Class Diagrams

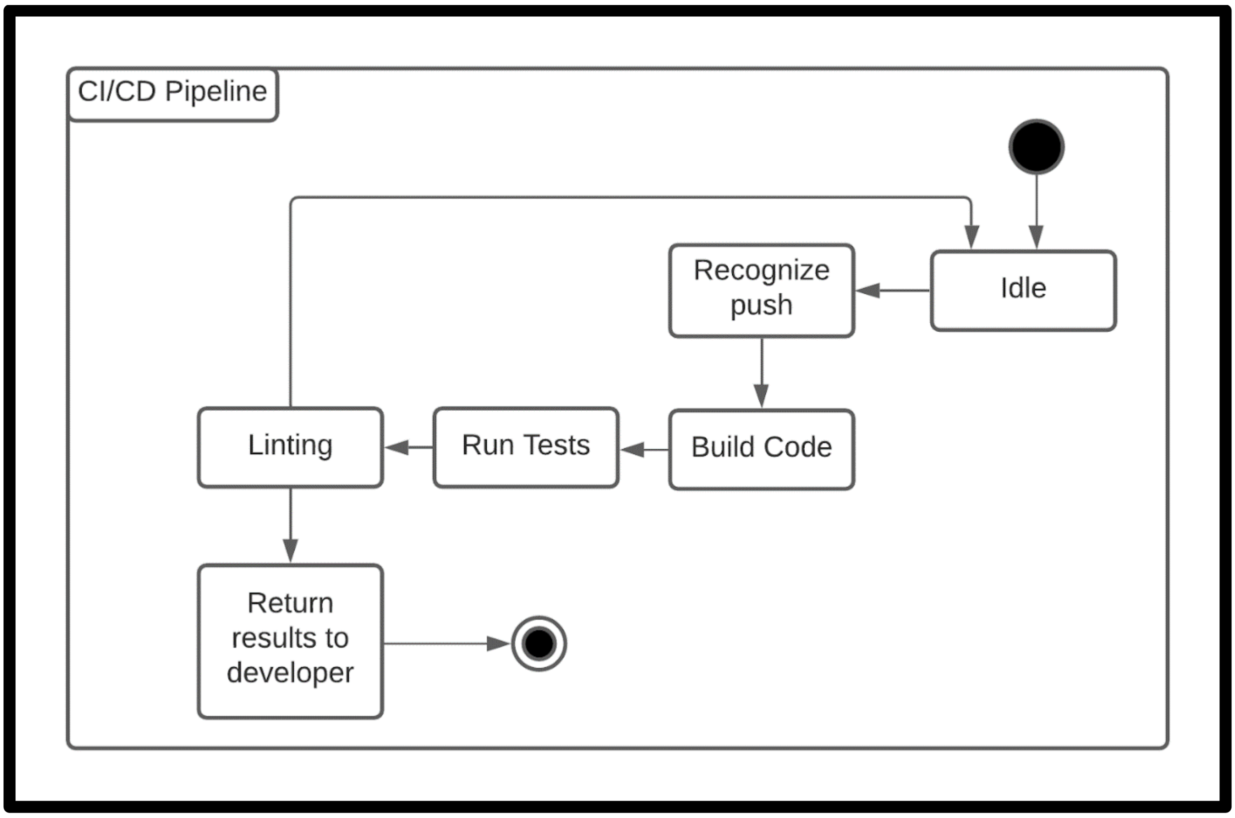


2.2.5. Sequence Diagrams



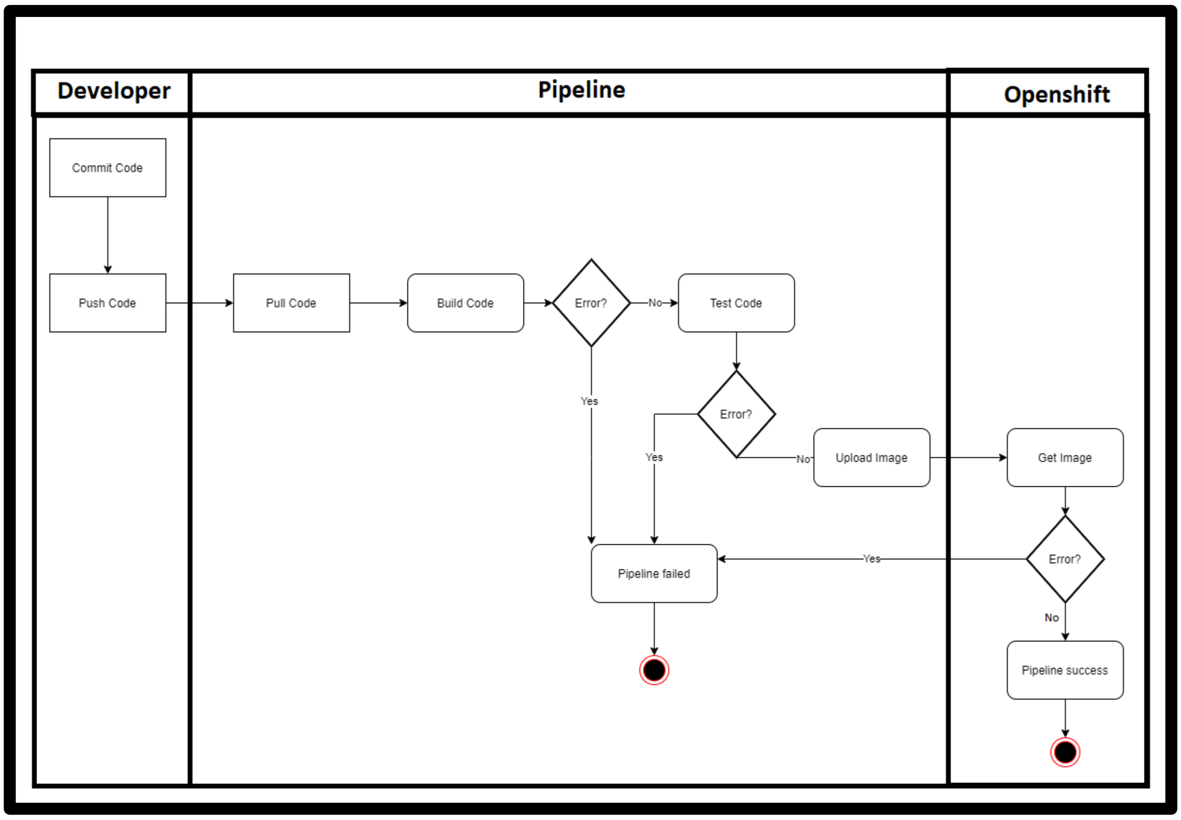
Software Design Specification

2.2.6. State Diagrams



2.2.7. Other relevant models

Dynamic Model



Software Design Specification

References

1. What is DevOps? - Amazon AWS
2. Podman
3. Red Hat OpenShift
4. COVID-19
5. Git
6. GitHub Actions
7. Selenium Webdriver
8. Jest
9. Eslint
10. Rapid API
11. Vue.js
12. Node.js

Other useful resources

- https://learn.openshift.com
- https://www.openshift.com/blog/deploying-to-openshift-using-github-actions
- https://redhatspain.com/openshift-pipelines/
- https://developers.redhat.com/blog/2019/02/21/podman-and-buildah-for-docker-users/
- https://developers.redhat.com/developer-sandbox