Israel Coleman Jr   
CMPSC 487W

Nate Wagenhoffer

29 October 2023

Testing and CI

Test plan:

**Step 1: Setting Up the Repository**

1. **Create a New Repository**: If you haven't already, create a repository for your project on GitHub.
2. **Branches**: Utilize branches to manage different versions of your test plan (e.g., **main**, **develop**, **test-plans**).

**Step 2: Test Plan Document**

1. **Create a Test Plan Document**: In your repository, create a new markdown file (**test\_plan.md** or similar) in a dedicated folder (e.g., **/docs** or **/test**).
2. **Structure of the Test Plan Document**: Outline the structure of your test plan document. For example:
   * Introduction
   * Objectives
   * Scope
   * Test Strategy
   * Test Phases (e.g., Unit Testing, Integration Testing, User Acceptance Testing)
   * Test Cases
   * Test Environment
   * Responsibilities
   * Risks and Contingencies
   * Metrics and Reporting
   * Approval and Sign-Off

**Step 3: Writing the Test Plan**

1. **Introduction**: We will Provide a brief overview of the project and the purpose of the test plan.
2. **Objectives**: Define the goals and objectives of the testing process.
3. **Scope**: Specify what will be tested and what will not be covered.
4. **Test Strategy**: Explain the overall approach to testing (e.g., automated testing, manual testing, tools to be used).
5. **Test Phases**: Detail the different testing phases and their objectives.
6. **Test Cases**: List down specific test cases, scenarios, or user stories to be tested.
7. **Test Environment**: Describe the hardware, software, tools, and configurations required for testing.
8. **Responsibilities**: Clearly define roles and responsibilities of team members involved in testing.
9. **Risks and Contingencies**: Identify potential risks and how they will be mitigated.
10. **Metrics and Reporting**: Specify the metrics to be collected and how test results will be reported.
11. **Approval and Sign-Off**: Outline the process for approval and sign-off of the test plan.

**Step 4: Collaborating and Version Control**

1. **Collaboration**: Team members are to review and provide feedback on the test plan document.
2. **Version Control**: Use GitHub's version control features to track changes and revisions to the test plan document.

**Step 5: Publishing the Test Plan**

1. **Commit and Push**: Once finalized, we will commit the changes to the repository and push the test plan document to GitHub.
2. **Readme or Wiki**: Optionally, we can link the test plan document in the repository's README file or create a wiki page for easy access.

**Step 6: Review and Updates**

1. **Regular Updates**: We will Ensure the test plan is regularly updated to reflect any changes in the project or testing approach.
2. **Feedback and Improvements**: Incorporate feedback and improvements from testing activities into the test plan.

**Test automation and CI:**

Test automation infrastructure:

PyTest

**Justification:**

Our entire project yet is using python such as our VS (conda), and our continuous integration is using Python packages with conda, and inside of our git actions workflow, we are using PyTest, just a good way to keep our workflow linear.

**New test code:**

To add new test code to our base, we already created a testing branch, and with CI workflow setup using a push trigger, it will get added automatically to our base when we push the test code to the base.

**CI Service and repository connection:**

Our CI service is through GitHub actions, setup using a workflow, were using Python packages with conda. As seen below.

A screenshot of a computer

Description automatically generated

**Justification:**

We decided to use this again, just to keep everything linear so we do not have to jump between different languages and programs, everything can just stay in python. Also, we decided to using GitHub actions, just so it will be easier, because its automatically embedded inside of our repo.

**Pros/Cons matrix:**

There were two other CI services we investigated being Travis and MS Azure pipelines.

Travis

**Pros:**

1. **Ease of Use:** Travis CI is relatively easy to set up and configure for continuous integration workflows. Its integration with GitHub makes the setup process seamless.
2. **GitHub Integration:** It seamlessly integrates with GitHub repositories, allowing developers to trigger builds and tests automatically upon code commits or pull requests.
3. **Diverse Language Support:** Travis CI supports a wide range of programming languages and environments, making it versatile for different projects.
4. **Free Tier:** It offers a free tier for open-source projects, enabling developers to utilize continuous integration without incurring costs for these types of projects.
5. **Build Matrix:** Allows the configuration of multiple build environments simultaneously, enabling tests across various versions of dependencies or different operating systems.
6. **Customization:** It provides extensive configuration options, enabling customization of build steps, environments, and deployment procedures.
7. **Community Support:** As a widely used CI tool, it has a large and active community, providing ample resources, documentation, and support.

**Cons:**

1. **Limited Concurrency:** The free tier may have limitations on concurrency and build capacity, causing potential queueing or delays in executing builds for larger projects or during high-traffic periods.
2. **Complex Configurations:** Advanced configurations might be complex, requiring a deeper understanding of its configuration language or documentation.
3. **Limited Build Time:** Free-tier builds might have a time limit, which can be restrictive for larger projects or tests requiring longer execution times.
4. **Private Repositories Cost:** While open-source projects can use Travis CI for free, private repositories require a paid subscription, which might not be cost-effective for small or individual projects.
5. **Dependency on External Services:** As it heavily relies on GitHub integration, any issues with GitHub's availability might affect Travis CI's functionality.

MS Azure pipelines

**Pros:**

1. **Integration:** Seamless integration with Azure DevOps services and other popular development tools like GitHub, Bitbucket, etc.
2. **Flexibility:** Supports multiple languages, platforms, and deployment targets (Windows, Linux, macOS).
3. **Scalability:** Scales well for projects of any size, from small teams to enterprise-level applications.
4. **CI/CD:** Comprehensive CI/CD capabilities for automating builds, testing, and deployment processes.
5. **YAML Support:** Configuration as code using YAML for defining pipelines, making it version-controlled and easily reproducible.
6. **Extensive Marketplace:** A wide range of extensions and integrations available in the Azure DevOps marketplace for additional functionality.
7. **Security:** Built-in security features and compliance with industry standards, offering secure pipelines and data protection.

**Cons:**

1. **Learning Curve:** Steep learning curve for beginners, especially while setting up and configuring pipelines with YAML.
2. **Complexity:** Managing complex pipelines can become challenging, especially when dealing with multiple stages, dependencies, or conditional workflows.
3. **Cost:** Costs may escalate for large-scale usage or resource-intensive builds, as Azure Pipelines charges based on parallel jobs and usage.
4. **Limited UI Features:** The UI can sometimes lack certain advanced features available through YAML configuration.
5. **Platform Restrictions:** Although it supports multiple platforms, some functionalities might be limited or work differently across different OS environments.
6. **Dependency on Azure Services:** Heavily tied to the Azure ecosystem, which might be limiting if you prefer using other cloud services.

**CI build Test:**

1. Unit test
2. Performance test

**CI Triggers:**

Push, will be the only type of trigger we will use.