Quang Hoang Nguyen

CONTINUOUS INTEGRATION FOR EMBEDDED ENVIRONMENT

Technology and Communication
2022

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Quang Hoang Nguyen
Title	Continuous Integration for Embedded Environment
Year	2022
Language	English
Pages	59 + 2 Appendices
Name of Supervisor	Jukka Matila

The purpose of this thesis was to create a CI pipeline for Embedded Environment with Azure DevOps to perform an integration between tools and service, such as Azure Boards, Git Repository and Visual studio code editor. During the implementation, this project emphasised the automation of continuous integration.

The material was collected mostly on the Internet and services supplier's documentation.

Azure DevOps was used to create an automated environment. Developers commit the code will automatically trigger the CI pipeline which building, testing, and merging the code into the Azure repository according to Azure Boards tasks.

Keywords DevOps, CI/CD pipeline, Scrum, and Embedded

CONTENTS

CONTENTS of FIGURES

LIST OF APPENDICES

1 INTRODUCTION

In modern technology world, a software project tends to evolve complicatedly and uncontrollably. The process requires many good, productive tools and other factors as philosophies. To survive and grow in this information age, a company needs continuous stable loops to compete on the market. Having a fast-paced software development life cycle is a minimum requirement a company/business should have. The life cycle should not only be fast-paced but also adaptive to rapid changes from customers and also include quality product application.

This project thesis aimed to create a CI pipeline in practical part. A pipeline is the process of accumulating instructions from the processor through a pipeline. It allows storing and executing instructions in an orderly process /1/. It generates an automation of merging local developers' code to the entire project after building a Real-time operating system and testing code with the Google test. This pipeline supports none-human interventions which emphasizes the automation strength of tools in application development reliably and faster.

Chapter 2 researches around the DevOps philosophy, Agile methodology, Scrum framework and CI/CD pipeline. This theorical part shows how these practices could help a company to stay well against the speed of technological changes and how these terms could satisfy the customers but still keep the quality of the potential application. In addition, Chapter 2 also makes a comparison between those philosophies and their pros and .

2 DEVOPS BACKGROUND

This section summarises the definitions related to DevOps: Waterfall, Agile, Scrum, DevOps, and Continuous Integration/ Continuous Delivery/ Continuous Development. All of these definitions have a mutual purpose that increases the software development life cycle.

Software Development Life Cycle (SDLC) is a process that produces of high-quality, low-cost software in the shortest possible production time. The goal of SDLC is to provide a well-structured flow of phases that meets and exceeds all customer expectations and requirements. A detailed plan is tailored to stages or phases that enhances development pace and minimize risks and costs. /2/

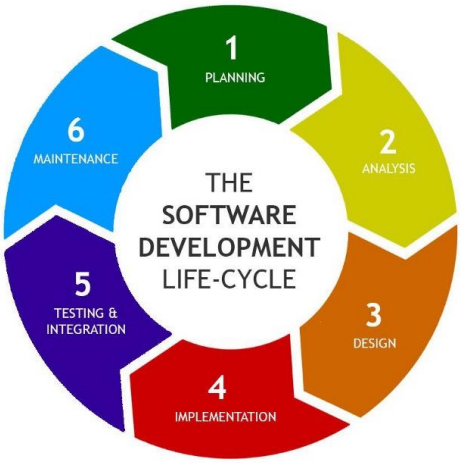


Figure 1. Software Development Life-Cycle /3/

There are four factors that help to generally define an epitome SDLC:

- **Process** must enable a faster feedback cycle and create a culture for learning
- **Architecture** should be agile for change and resilient to failure
- **Technology** is imperative for innovation, increasing productivity and maximizing impact

- A robust **infrastructure** is the foundation for innovative platforms

2.1 Waterfall Model

Prior to Agile, the Waterfall model was the first model to be introduced in SDLC. Waterfall is a linear approach; the next phase is begun only after the previous phase is complete.

As its name, the process flow of this architecture starts from high ground and 'falls' to the final stage. It apportions into small stages to control the process individually and easily. The outcome of one phase represents as the input for the next phase. This approach is also called a sequential flow.

2.1.1 Waterfall Model Workflow

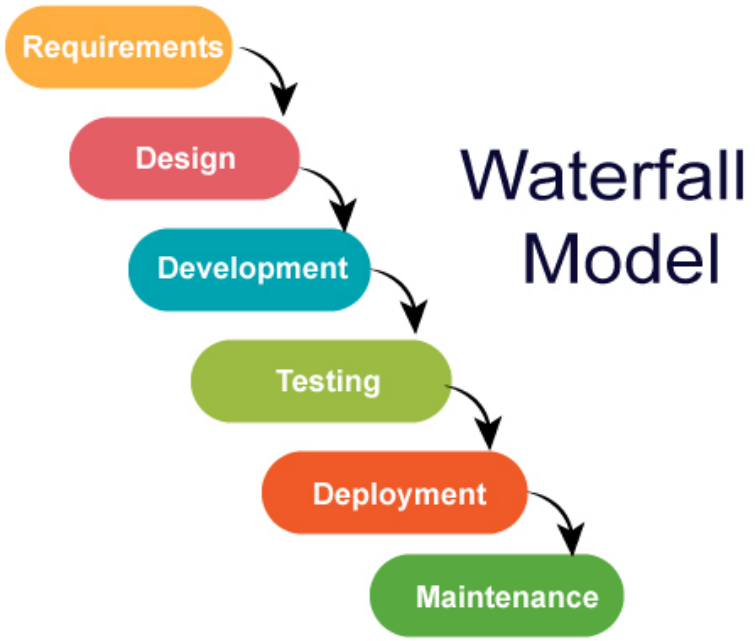


Figure 2. Waterfall model /4/

As Figure 2 shows, the requirements are first received from the customers or stakeholders. The next phase is the planning and design phase before transferring

the ideas to become discernible software by developing the code. Whenever application prodcut is about to be released or deployed, it must be tested carefully before publishing. The final work is to maintain the software. This model runs through these consecutive phases in a definite order and starts over again at the top.

2.1.2 Advantaged and Disadvantages of Waterfall Model

The waterfall model was a pioneer for determining the success of the SDLC. In the early stages, it has many advantages. It is very easy to follow and understand to use, not only for developers but also the customers. The phases are indicated explicitly and completed one at a time. The tasks do not overlap and it is convenient to check or arrange tasks to perform a good phase. Everything is well documented because the work is obvious.

The model asks the team to keep to the requirements and scope determined at the first stage of the project. Therefore, the critical drawback is that it is difficult to make changes or additions. If customers need to change anything in the middle of the process, they have to wait for the next loop to update a new feature. Moreover, the code may not work after all and it leads to a bottleneck for the whole process. Because the team cannot move to the next stage with a malfunctioning code. /5/

In a long term, this model cannot work effectively because of the incoming updates and time consuming. Keeping the testing phase on the last half of the model causes uncertain software. The testing phase is always a significant step to prove the software is functional. . In addition, there is not any working product until the final phase, which the entire process takes a lot of time. /5/

With waterfall methodology, there is more insecurity and inefficiency than advantages. This model only works well with simple project because it is time consuming and the work flow is rigid.

2.2 Agile

Agile is an epitome SDLC philosophy. It is a set of values and principles which comprises many methodologies, processes, and tools to speed up software development. It is also an iterative development approach utilizing collaborative effort through self-organizing teams originating in software development. /6/

Development teams have focused on speeding up the process to get new products and features faster. A dynamic process requires the ability to adapt to new conditions on the fly which resulted in the creation of Agile.

Figure 3. Agile Manifesto /7/

In 2001, there were 17 independent developers created Agile Manifesto by agreeing common traits of software development improvement /7/.

Addition to the values of the Manifesto, there are 12 principles that support an agile software development culture. In general, Agile promotes iterative feed backs cycle, cross-functional collaboration, self-organization teams that uses tools and process to tailor to particular tasks.

2.2.1 Agile Model Workflow

The Agile workflow is almost similar to that of Waterfall as Figure 4 shows below. There are total of six stages in this model: planning then design, code development, test, deployment, reviewing and launching. The key item in this model is working in periods of iteration. The Waterfall model proceeds without looking back, but Agile has an iterative cycle to plan, test, review and develop the code again.

It divines the application into small features and tasks for each iteration. After an amount of time the specific tasks are completed in a priority order. In this way, it keeps the small features and the whole project clear and easy to follow without forcing to multitask all features at once as in Waterfall.

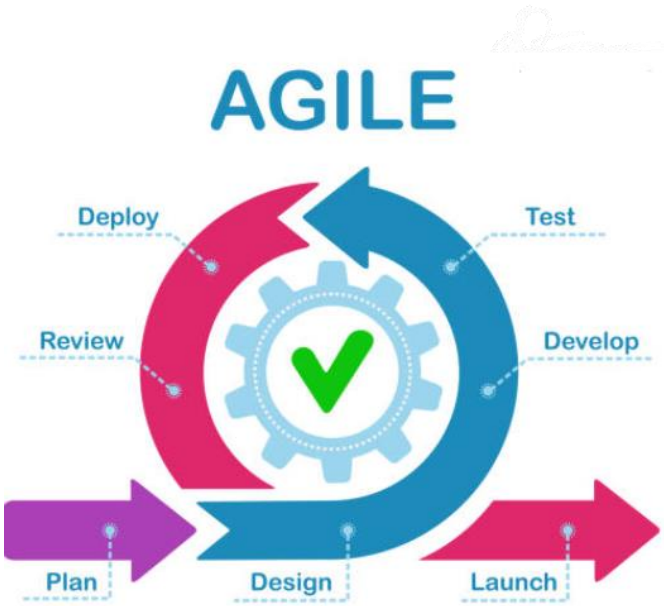


Figure 4. Agile model /8/

In other words, each iteration is like consecutive, individual wheels of a train. When combined together, it makes a train which makes the software development smoothly and agile. The real utilisation of Agile gives a common foundation for making decisions about the good way to develop software.

There are a lot of frameworks of Agile such as Scrum, Kanban, and Extreme Programming.

2.2.2 Scrum

In 1986, Hirotaka Takeuchi and Ikujiro Nonaka first used the software development term scrum in a paper titled “The New Product Development Game”. The paper was published in the January 1986 issue of Harvard Business Review. The term is borrowed from rugby, where a scrum is a formation of players. /9/

According to the Adevait report, Scrum is the highest used method among the Agile methodologies with 56%. With Scrum the development process can be speed up efficiently. /10/

Figure 5. Scrum Framework portion percentage compares to other frameworks /10/

Scrum is one of the most popular lightweight frameworks by advancing from Agile because it can be applied to any kinds of teamwork through principles and lessons. Scrum concentrates on communication between team members, self-organized, cross-functional cohesive working teams, and faster delivery potential shippable product. There are three new terms in Scrum: /10/

- Product owner
- Scrum master
- Sprint

The Product Owner is part of the Scrum team. This person has responsibilities to transfer user stories and create a product backlog. They are also acting on behalf of the customers to identify specified requirements for the development team. /11/

The Scrum master is a master of scrum who facilitates Scrum by managing and helping the team to ensure the Scrum frameworks is followed. The responsibilities are: /11/

- Clearing Obstacles
- Establishing an environment where the team can be effective
- Addressing team dynamics
- Ensuring a good relationship between the team and product owner as well as others outside the team
- Protecting the team from outside interruptions and distractions

2.2.3 Scrum Framework Workflow

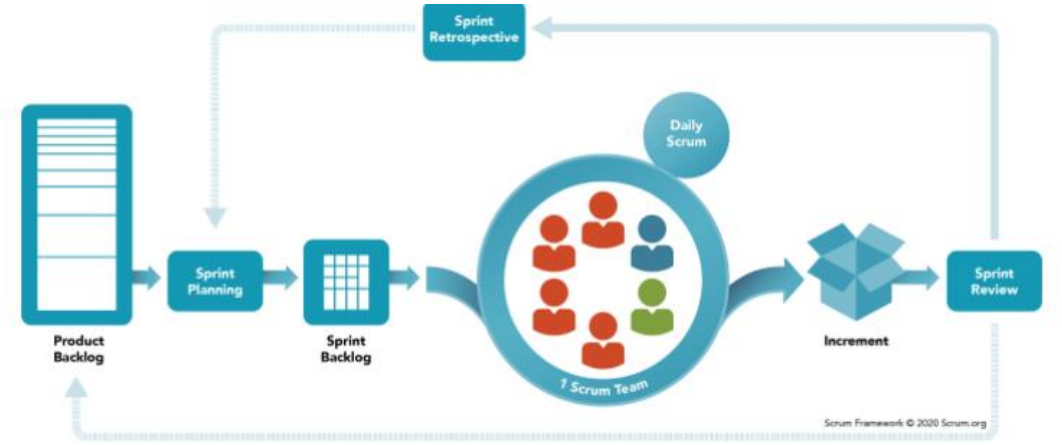


Figure 6. Scrum framework /12/

The workflow of Scrum is demonstrated in Figure 6. Scrum begins by filling in the detailed list Product Backlog, just as in planning. The end-product requirements of the stakeholders are defined as Product Backlog. The iteration of Scrum is called Sprint which lasts between 1 and 4 weeks depends on the team culture. During each Sprint, top user stories of Product Backlog are prioritized as Sprint Backlog which means breaking down each requirement into smaller pieces that have to be done on each Sprint. The team works on the defined Sprint Backlog. Everyday there is a short 15-minute meeting event called Daily Scrum. The aim of the event is to give a clear vision of work that the Scrum team has carried out, denote obstacle and solution for that, and to adjust the decisions. After every Sprint, the team gives the customers the increment which is a minimum viable product. At the end of a Sprint, the team meets to present the work they have achieved and review it together in the Sprint review. This focus on how to improve the product outcome. The last event in the Sprint is the Sprint retrospective. The retrospective is designated to build a habit of continuous process improvement and to Review the last Sprint workflow issues, difficulties, and how to replicate successes in the next iteration /13/.

2.2.4 Advantaged and Disadvantages of Scrum

The advantages of Scrum are the same as with Agile. The main advantage is to boost the development process faster. Scrum helps the team to complete project deliverables quickly and efficiently. A large project can be controlled by diving into easily manageable Sprints which lead to reduce time consumption. In addition, Scrum requires that any team member should be able to do what is required for the progress of the project. Also, when each team member can perform every job, it increases understanding and bonding between them.

According to customers, short Sprints enable feedback and changes. It typically results in more satisfied customers. The Scrum framework makes it more likely that the employees involved in a project are motivated and satisfied.

Scrum asks for cooperative spirit in each member, which also a negative point. If individuals are not very committed or collaborative the Scrum framework might fail. Therefore, applying Scrum to a large team is a difficulty.

Because each Sprint takes place in very short time in a matter of few weeks, it may force members to work under intense stress to attain the Sprint goals. The team must to have experience and skills to perform their own tasks quickly and successfully.

Scrum makes application development rapid through empirical process control, self-organization, collaboration, value-based prioritization, time-boxing, and iterative development.

2.3 DevOps

DevOps is a concept/mindset that fosters a culture of collaboration between the development team and the operation team, which is an evolution from Agile. The priority is on the maintenance of a shared culture throughout different sectors of the production team. This concept eliminates any specific siloed barrier that may

impede the process to improve collaborative and productivity by automating infrastructure, automating workflows, and continuously measuring application performance.

DevOps is a philosophy that combines software development with operations. The intent is to enable communication between both teams so that they can build, test, and release software more quickly with greater efficiency and speed /14/

The main characteristic of the DevOps movement is to advocate automation strongly and monitor at all steps of software construction, from integration, testing, releasing to deployment and infrastructure management. To obtain this trait, implementing continuous integration and continuous delivery (CI/CD) allows organizations to see a tremendous improvement in deployment frequency, lead time, detection of cybersecurity vulnerabilities and flaws, mean time to repair and mean time to recovery.

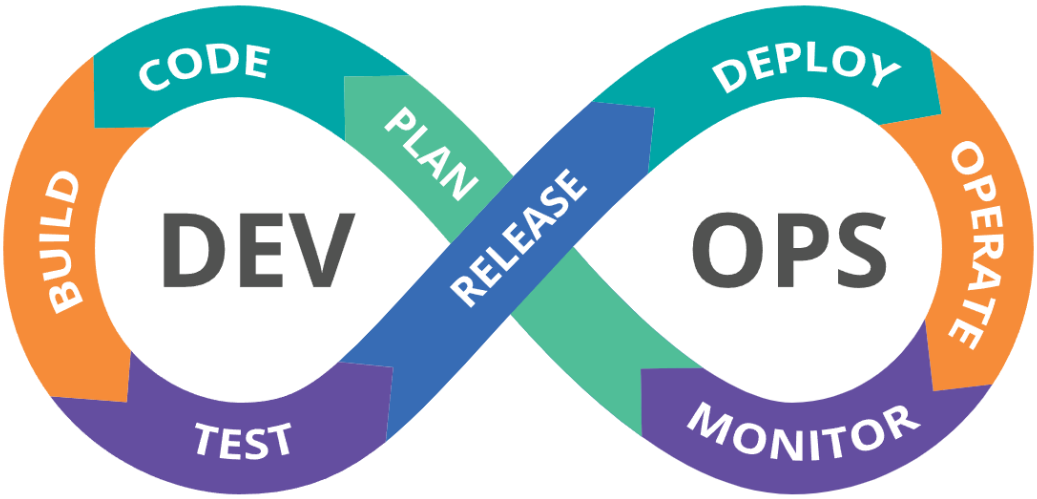


Figure 7. DevOps logo implies for an eternity of development process /15/

The goal of DevOps is also to deploy a code to reliable production faster and in an automated way. Instead of the old way, developers write the application and then give to the operation team who deploys and manages the infrastructure server to maintain the deployment, now both teams merge into one cross-functional team

for the entire process. Developers are also in charge of deployment software and vice versa. This makes both teams understand the essential issues of others and not doing the blame game. It ensures the effective of collaborative, cross-functional teams. /15/

2.3.1 DevOps or Agile

Agile and DevOps have the same purpose on the streamline development process. However, the aim focus of both cultures has several differences.

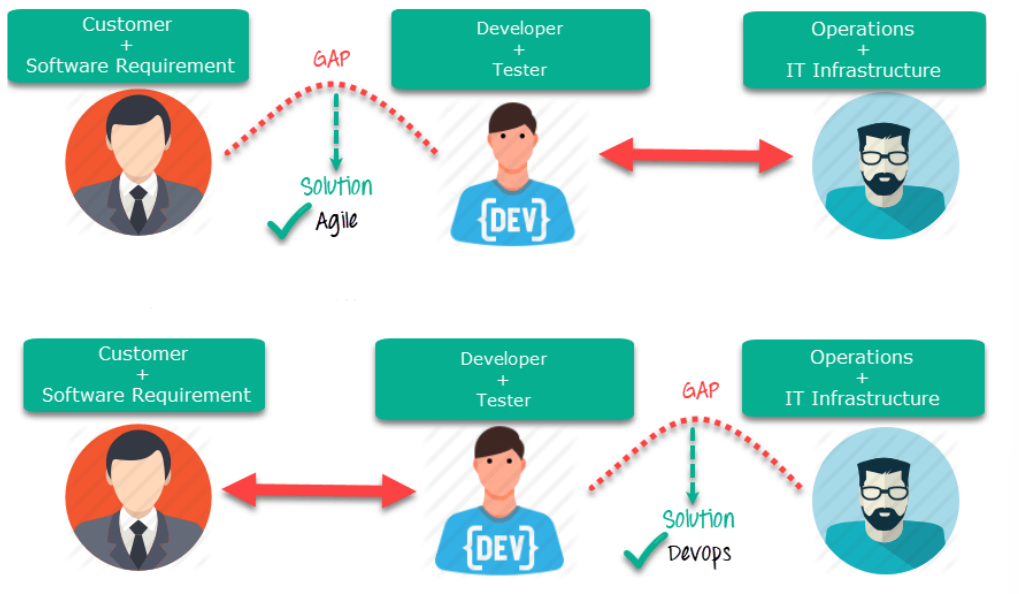


Figure 8 Solution gap of Agile and DevOps /16/

As shown in Figure 8, DevOps addresses the gap between development and operation. /16/

- DevOps unifies Dev and Ops team. Agile tries to collaborate each member in a smaller team so it can react quickly to customer changing.
- Agile deploys depends upon the Sprint. DevOps focuses on hyper-releases starting with several per day.

- DevOps focuses on testing and delivery product while Agile focuses on constant changes.
- Target area of DevOps is to end-to-end business solutions and fast delivery whereas Agile is Software development.
- Agile does not emphasise automation while DevOps does.
- Agile has a shorter development cycle and improved defect detection. DevOps supports Agile's release cycle which is an integral part of DevOps.

2.3.2 DevOps and Agile

Despite DevOps and Agile represent different solutions/approach in the software development. To create a perfect, powerful development process, the productive way is to combine these two practices. Both still have the same purpose, so the question is not which one to choose but how to practice both.

Michael Mazyar, CTO at Cary NC. based Samanage, further elaborated on the "mutual benefits" of both DevOps and Agile. "When applied in tandem, Agile and DevOps can enable organizations to develop and implement technology with significantly greater speed. [Plus], there's an emphasis on putting customer needs at the forefront of whatever technology it is you are developing, [along with an] understanding [of] how the software is being used, and [how it should be improved]." /14/

The strength of each practice is in the different phase of cycle. Combining two methodologies makes them fulfill each other defects to even more to streamline the process and quickly deploy the product to customers.

2.4 CI/CD Pipeline

Continuous Integration (CI) and Continuous Delivery (CD) are key parts of the DevOps pipeline. CI/CD are actually separate but linked concepts, based upon a continuous practice whereby work is integrated frequently. These two terms are

an automation of activities in application development stages which deliver frequently software to customers. Together, it creates a collection of activities that must be performed in order to deliver a new valuable version of software via CI/CD pipeline. Pipeline introduces automation and continuous to improve the SDLC. Before to the pipeline, people have to accomplish an application life cycle manually in every step.

Figure 9. CI/CD pipeline /17/

2.4.1 Continuous Integration

Continuous Integration (CI) participates in three simple steps as shown in Figure 9. CI emphasizes test automation to check the code is not broken when new commits are integrated.

In a company, there are many developers who work in the same project. When they complete their own work and then merge it into the shared repository, this action is manual, tedious and time consuming. This might take a long time to finish. In addition, it might cause a merge conflict with everyone pushing their code simultaneously. The repository receives lots of changes concurrently and the bugs/issues occur because each developer has customized their own local Integrated Development Environment (IDE) rather than on one cloud-based IDE. /17/

To avoid this, CI comes to help developers to merge their code smoothly to the shared bank code frequently and flawlessly. CI provides a validation code by automatically building the application and running different levels of automated testing such as unit and integration tests /17/.

To ensure the individual code before merging to the repository, the code is being built again and tested in the mutual CI sever. Developers finish developing their code in local machine which includes the source code and the self-test code before pushing their code into the repository. Before actually being a part of the main project code, the code is verified by compiler (GNU, javac, PyCham, etc). Then all test source codes are compiled. After that, tests are executed, such as unit tests, and integration tests. One more checking step before the final merging is to run a code health check – static analysis code (cppcheck, Checkstyle, test coverage, etc). Normally, compiling a code generates packages/artifacts like jar, war, zip to be deployed later. After passing all the build process, the code is verified to be able to be merged into the mainline.

If a code fails in some phases in the build, this is reported back to the owner, so they know they have to adjust the code to be validated. Meanwhile, the CI pipeline still works normally without stopping or waiting for the fault code. Thanks to CI, coders can now be more confident to push their self-code to the main source repository more often but still keep the verified quality by unit tests.

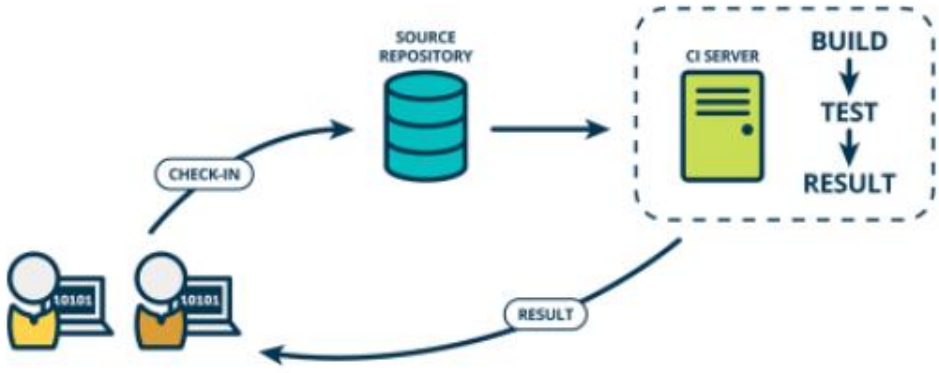


Figure 10. Continuous Integration process /18/

The benefits of CI are:

- Avoiding merge issues
- Testable build
- Delivery faster as the code is tested
- Deployment more often

2.4.2 Continuous Delivery and Continuous Development

CD stands for Continuous Delivery or sometimes Continuous Deployment. Continuous Delivery is a consequent process after CI that the mainline code is going to be tested again via many environments before turning into a product because is it the nearest process to production.

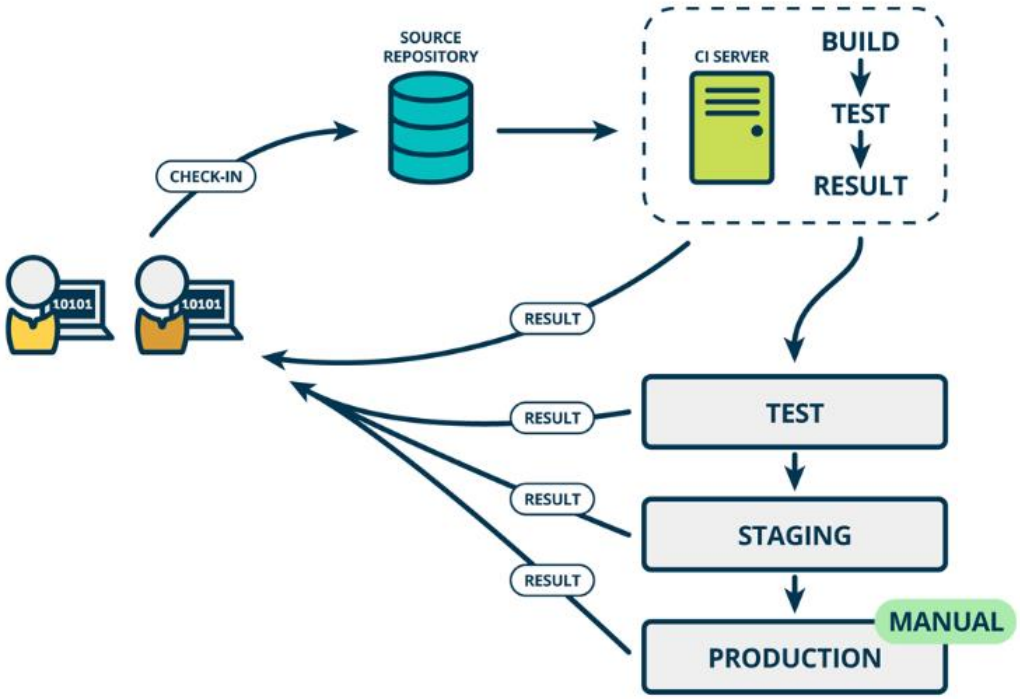


Figure 11. Continuous Delivery process /18/

The three environments are test-staging-production. CD entails deploying the application code from testing and developement into a staging environment, which is a replicate of the production stack, and running more functional requirement

and non-functional requirements tests. This environment is generally called Continuous Testing. For testing functional requirements (functional testing), Continuous Testing often involves unit tests, API testing, integration testing, and system testing. For testing non-functional requirements (non-functional testing - to determine if the application meets expectations around performance, security, compliance, and such.), it involves practices such as static code analysis, security testing, performance testing. /19/

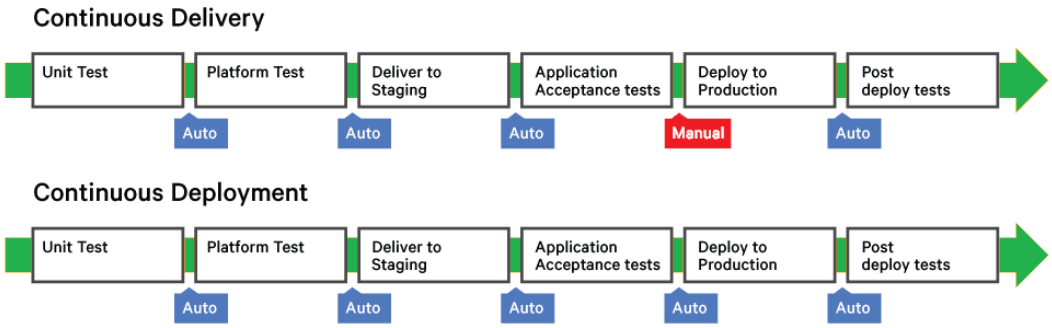


Figure 12. Comparison between Continuous Delivery and Continuous Deployment /21/

The staging environment could be a static environment premade for testing, or a provision and configuration of a dynamic environment with committed infrastructure and configuration code for testing and deploying the application code. After the staging environment is built using infrastructure as a code (IaC), a production environment can be built very quickly in the same way. /20/

All these all tests ensure

- Software can be released reliably whenever needed
- Automated deployment
- Deploy takes place often and quickly

If the code has passed all the tests, then it will transfer to the deploy state. In continuous delivery, there is a small manual approval step. The person in charge

will decide if the software is released. Even if it passes all tests, it still needs to wait for a approval.

To compare continuous delivery with continuous deployment, the only difference is there is no manual step approval. The process of C/Deployment is straight from the coding phase up to Production without any human intervention.

3 WORK DESIGN & IMPLEMENTATION

3.1 Work Design

In this section, the resource and technology used is listed in technology stack. The pipeline creation was carried out in design and implementation.

3.1.1 Technology Stack

Azure DevOps provides the developer with services for allowing teams to plan work, collaborate on code development, and build and deploy applications. Azure DevOps is a Software as a service (SaaS) platform from Microsoft that provides an end-to-end DevOps toolchain for developing and deploying software. These listed services provided by Azure DevOps are all used to implement this project /22/

- Azure Boards
- Azure Pipelines
- Azure Repos

Azure Boards provides software development teams with the interactive and customizable tools they need to manage their software projects. It provides a rich set of capabilities including native support for Agile, Scrum, and Kanban processes, calendar views, configurable dashboards, and integrated reporting. These tools scale as the business grows. The tools enable quick and easy tracking of work, issues, and code defects associated with the project. The Kanban board, shown in the following image, is just one of several tools that allows adding, updating , and filtering of user stories, bugs, features, and epics. /23/

Azure Pipelines automatically builds and tests code projects to make them available to others. It works with almost any language or project type. Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to test and build code and ship it to any target. /24/

PlatformIO is a cross-platform, cross-architecture, multiple frameworks, professional tool for Embedded systems engineers and for software developers who write applications for Embedded products. PlatformIO's unique philosophy in the embedded market provides developers with a modern integrated development environment. It supports built-in project generator for the most popular Embedded boards and frameworks. The key things to use PlatformIO is its framework that already supports Real-Time Operating System (RTOS) which is mainly used in Embedded systems. Moreover, this extension includes also debugging, unit testing, automated code analysis and remote management. It can maximize flexibility and choice by developers by using PlatformIO Core Command Line Interface (CLI). The project to build with this extension is implemented by "platformio.ini" – a project configuration file. /25/

Visual Studio code is a lightweight powerful code editor for this project. It can be installed easily and can be run on any operating system. With many extensions, it supports almost all languages just by simply integrating the extensions. Moreover, it has a built-in source control management git with not only Run and debug feature but also the deploy code with Microsoft Azure which hosts, stores, and queries relational and document-based data /26/. Thanks to those features, Visual Studio Code becomes the most popular source code editor.

The repository for this project is both GitHub and Azure Git Repository.

3.1.2 Workflow Process

The idea of this thesis project is to create an environment which integrates Azure Boards, Visual Studio Code and trigger CI pipeline in Azure. This environment is done empirically on Scrum Framework, so the flow is similar:

- Developer will be assigned for a task (feature, bug, …) in Azure Boards
- Developer develops the code through an assigned ticket task using Visual Studio Code

- Submit the code with a task ID to share the public repository
- Tasks/tickets automatically move to done state in a Sprint
- Trigger CI pipeline which tests and builds the code
- Send notification to the build master

The first stage is Branch. This stage is used to validate if the branches existed or not. The idea of this development was to have three branches concurrently

- Main/Master
- Development
- Test

As many usual other projects, Main is used to wait for merging the work from other branches to become the official mainline of the development.

The test branch was the main working branch to update or adjust any current development circumstance of the software. Development was the just the replication of the Test branch. When the code in Test was ready to run and passed all test, the code would be automatically merged into the Development branch. After merging, there would be a person who has the responsibility to pull the request from Development to Master branch such as a build master.

The next stage was Test. The purpose was to test the code with the unit Google test.

The Build code was the final stage. This aimed to build the RTOS uC/OS – III code by using platformIO and use Cppcheck to the static analysis code. If the process went successfully throughout the pipeline, the code was merged into Development and a notification was sent about a successfully built pipeline to the build master.

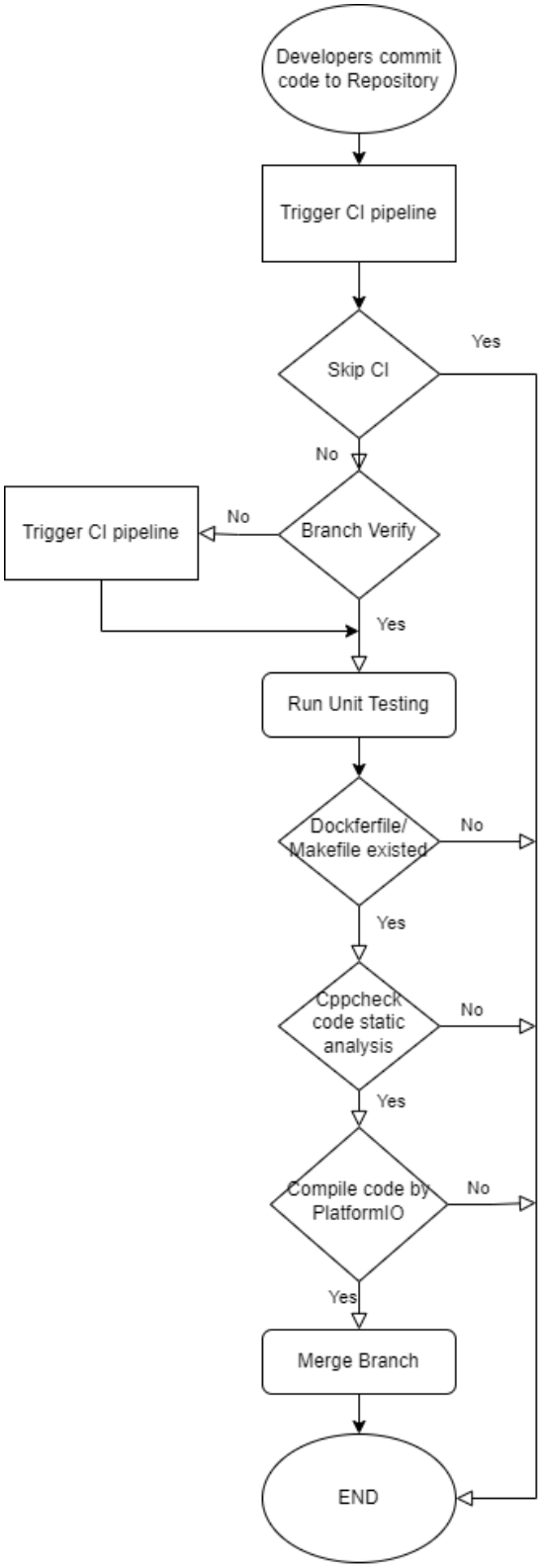

Developers commit code to Repository
Trigger CI pipeline
Skip CI
Yes
No
Trigger CI pipeline
No
Branch Verify
Yes
Run Unit Testing
Dockferfile/ Makefile existed
No
Yes
Cppcheck code static analysis
No
Yes
Compile code by PlatformIO
No
Yes
Merge Branch
END

Figure 13. Pipeline workflow

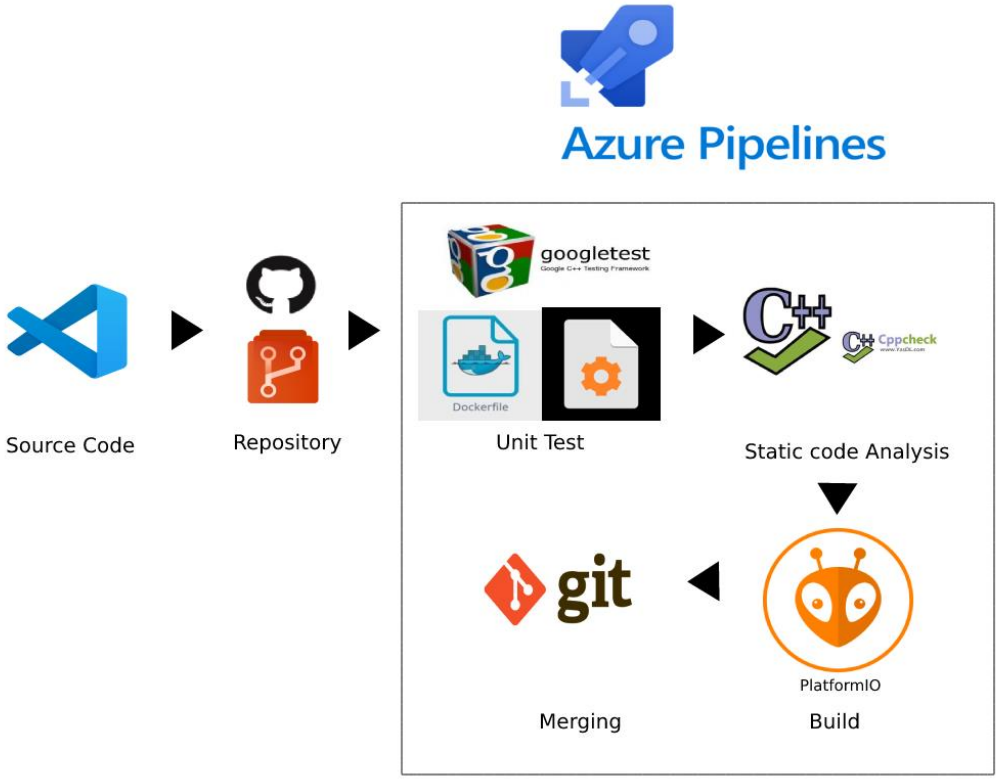


Figure 14. Visualize CI process' tools used

3.2 Pipeline Implementation

3.2.1 Project setup

To perform this project, project must be created first inside an organization.

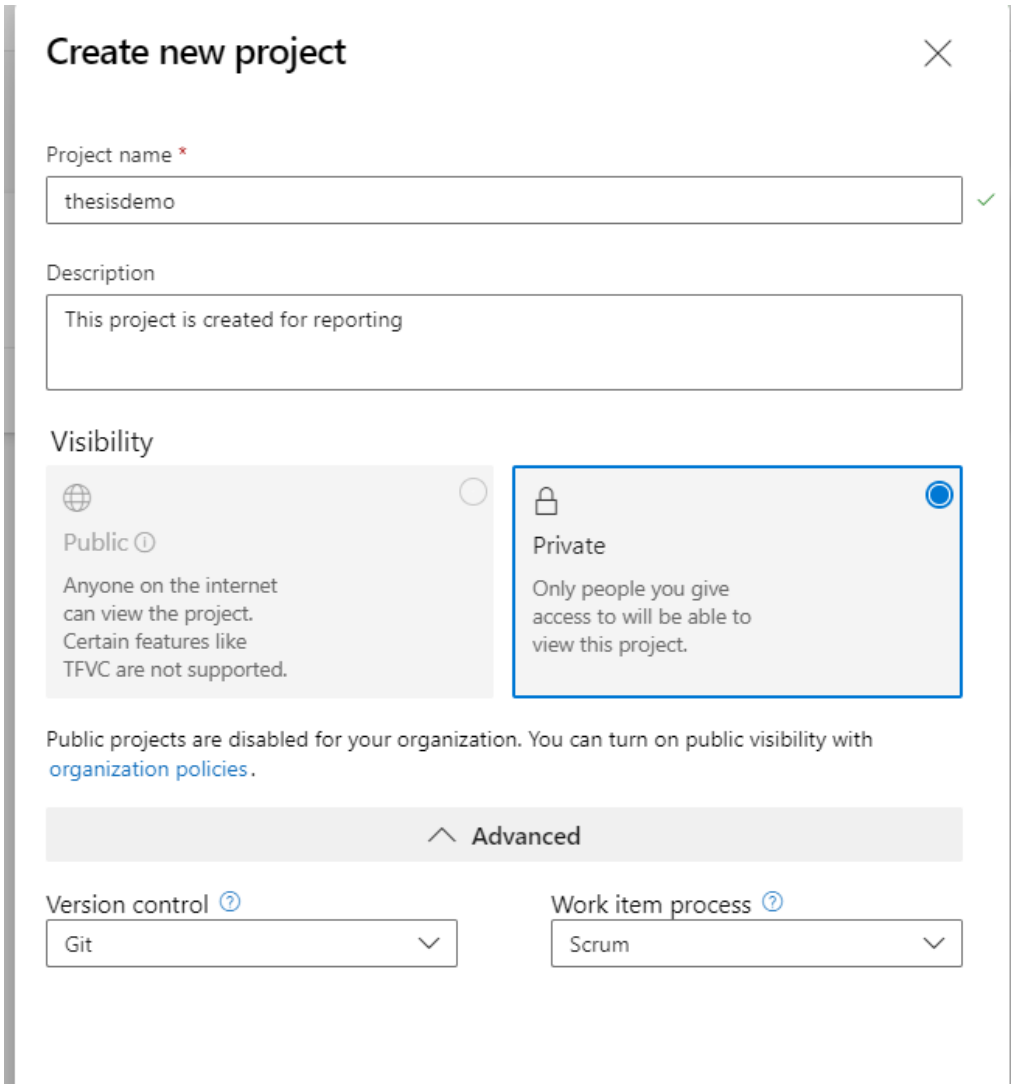


Figure 15. Create a project in Azure DevOps

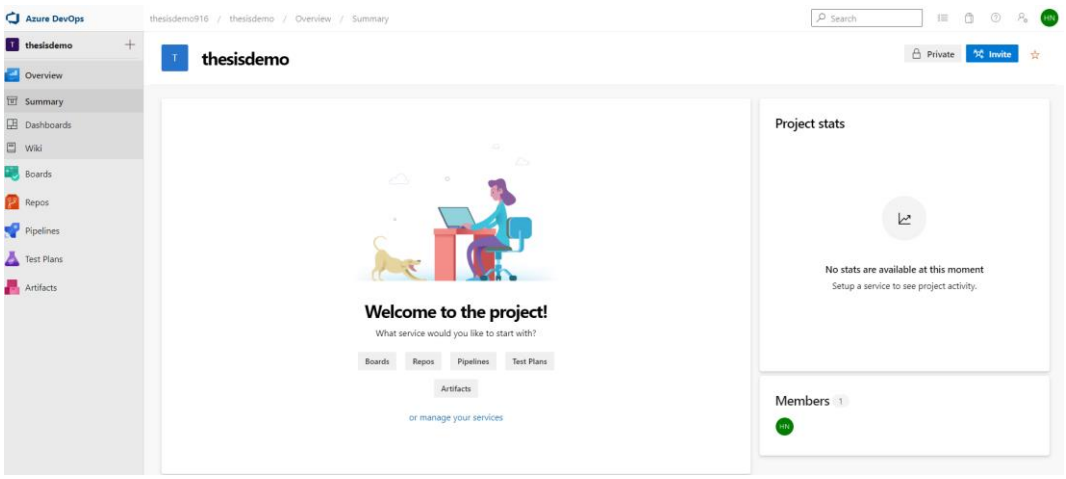


Figure 16. Project created successfully

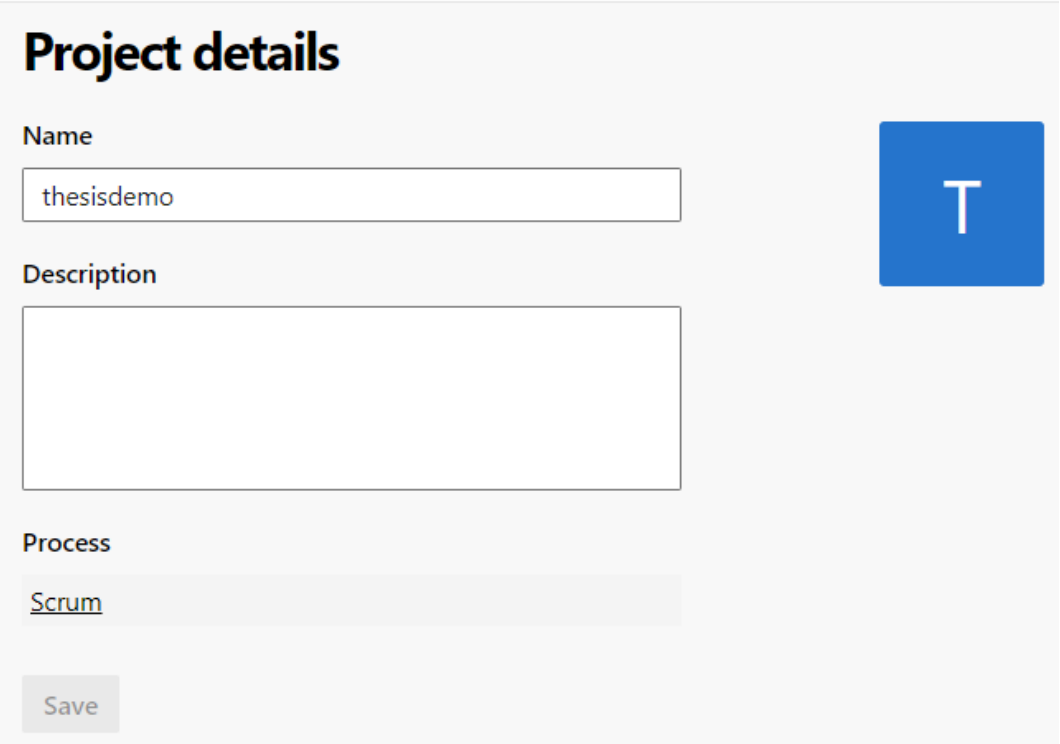


Figure 17. Scrum was set for project

3.2.2 Azure Boards Configuration

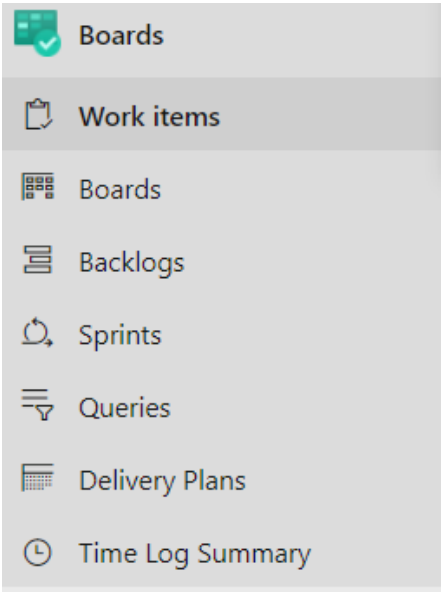


Figure 18. Azure Board' features first glance

- **Work items:** to create work items or to access list of work items with selected criteria.
- **Board**: to view work items as cards/tickets and perform quick status updates through drag-and-drop.
- **Backlogs**: to view, plan, order, and organize work items.
- **Sprints**: to access team's filtered view of work items based on a specific iteration Sprint.

As scrum explanation, there will be a product Backlog represent the requirements from customers. And task is created follow by backlog.

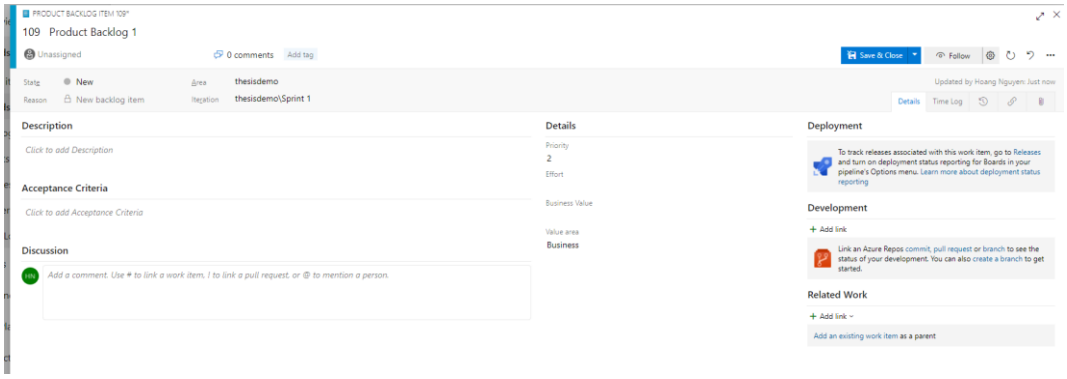


Figure 19. Product Backlog created

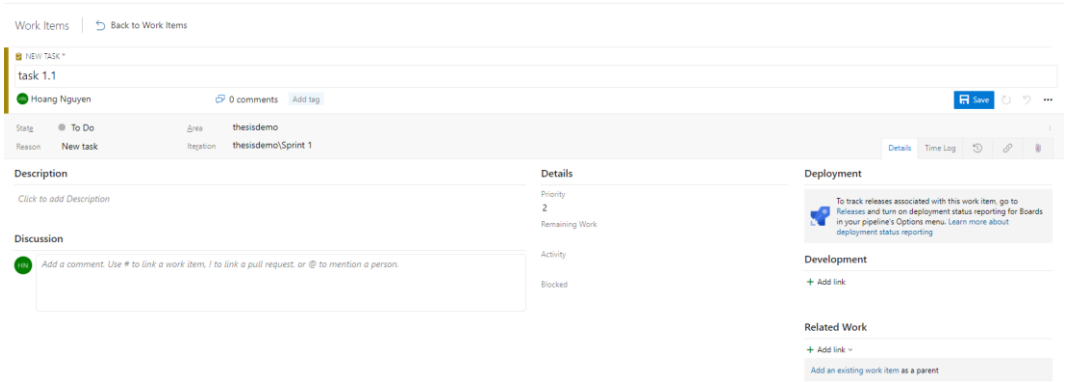


Figure 20. Task created

The work items always have it unique ID number on the left corner of each ticket.

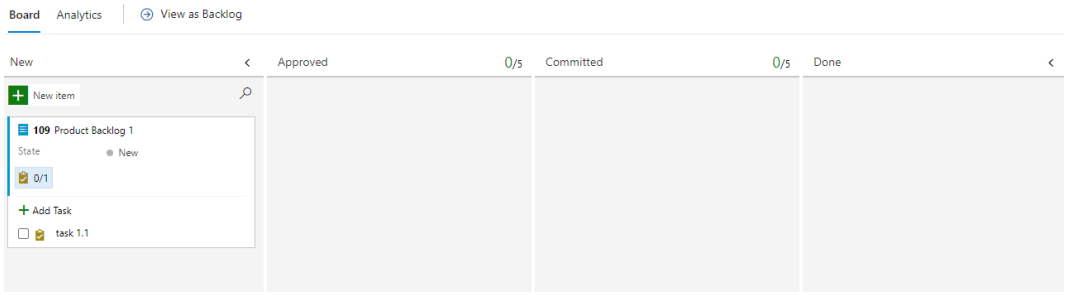


Figure 21. Product backlog viewed as Boards

Figure 22. Product Backlog item workflow /27/

Assigned a first Sprint duration as figure 23 below in **Sprints** section.

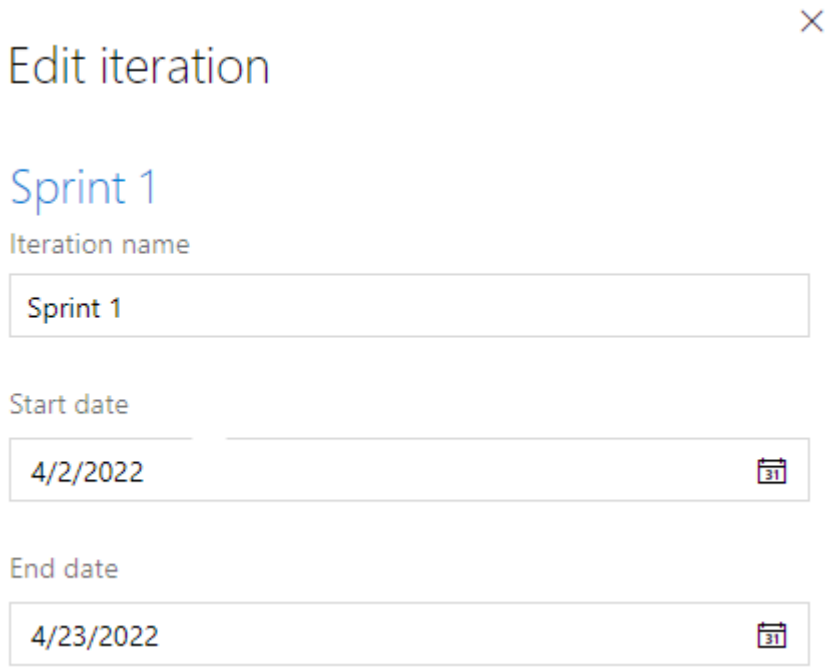


Figure 23. Edit Sprint 1 iteration

The Product Backlog will be breakdown into Sprint backlog. The figure 24 displayed the product backlog created before was address in Sprint 1 backlog as Scrum process – prioritize product backlog into Sprint backlog.

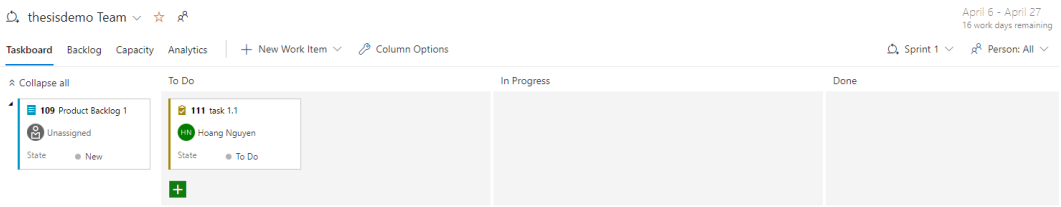


Figure 24. Sprint backlog view

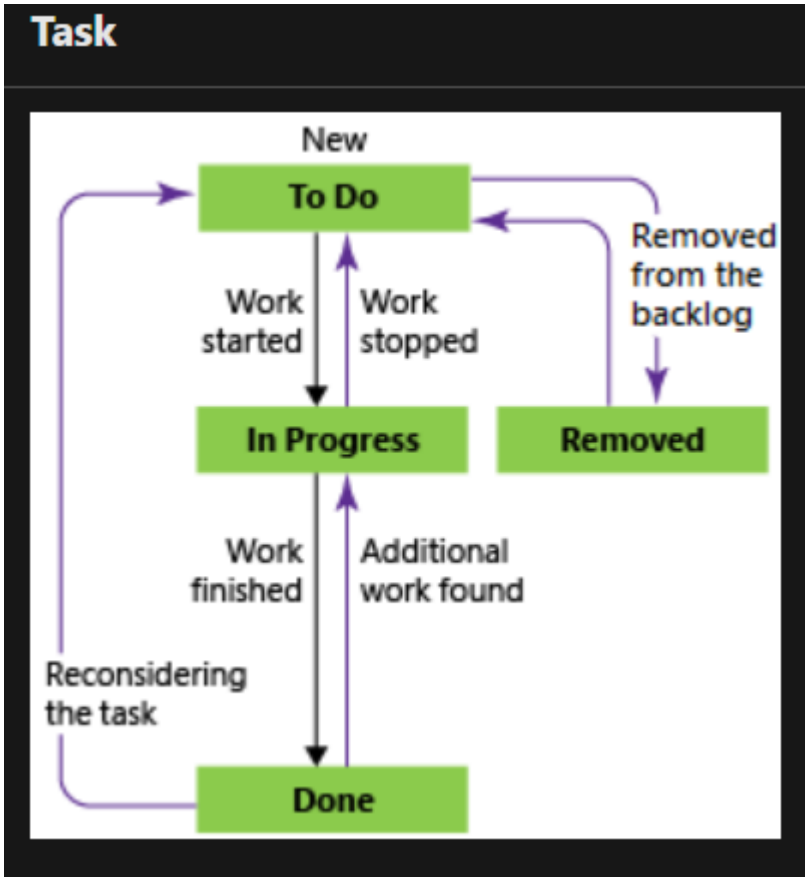


Figure 25. Task workflow /27/

After task and Sprint backlog was defined, developers job began.

3.2.3 Software development with integration environment

Before write some line of codes, developers have to choose repository for storing their code. This project code stored mainly at Azure Git. The Azure repository works identically in compare with GitHub by using Git version control system.

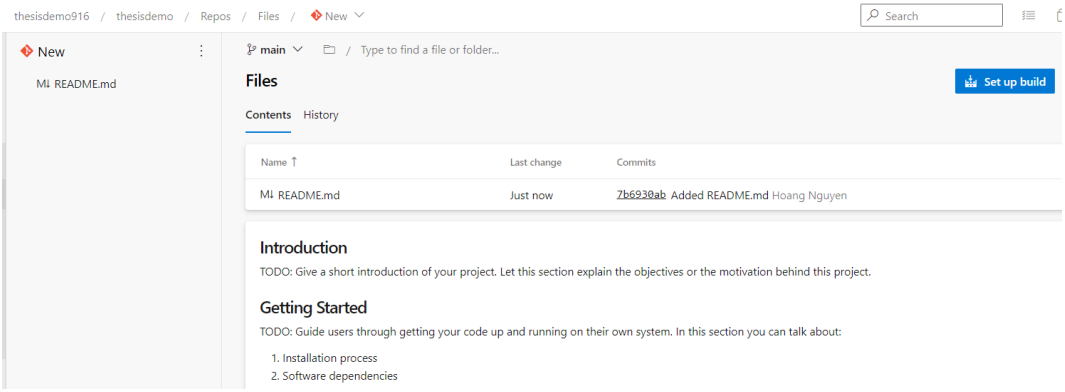


Figure 26. Azure Repository Git UI

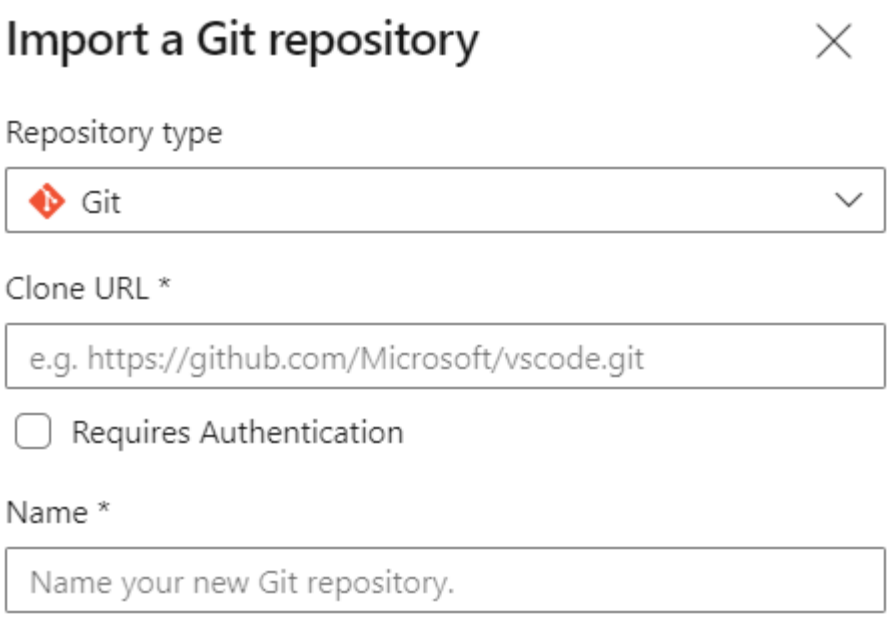


Figure 27. Azure repository can import another Git repository

To create an integration between Azure repository and boards, the repository had to be configured as figure 28 below. Otherwise, repository and boards worked as two independence entities and only be linked by manual works.

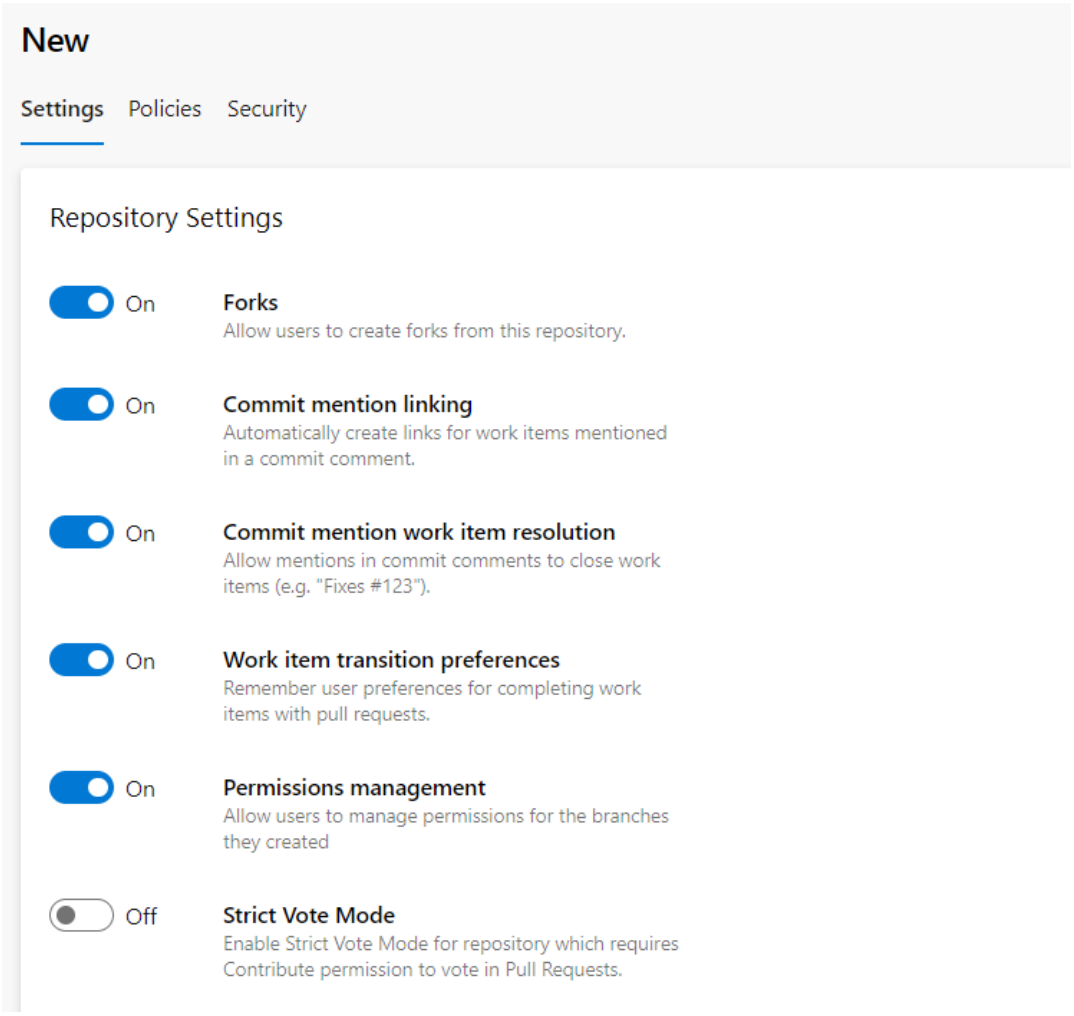


Figure 28. Commit mention repository setting

Regarding to GitHub, to embed GitHub repository to Azure boards, the project had to add GitHub connection in project setting in Boards section. After signing in the GitHub account, Azure Boards service asked to be approved and installed on chosen repository on GitHub as figure 30.

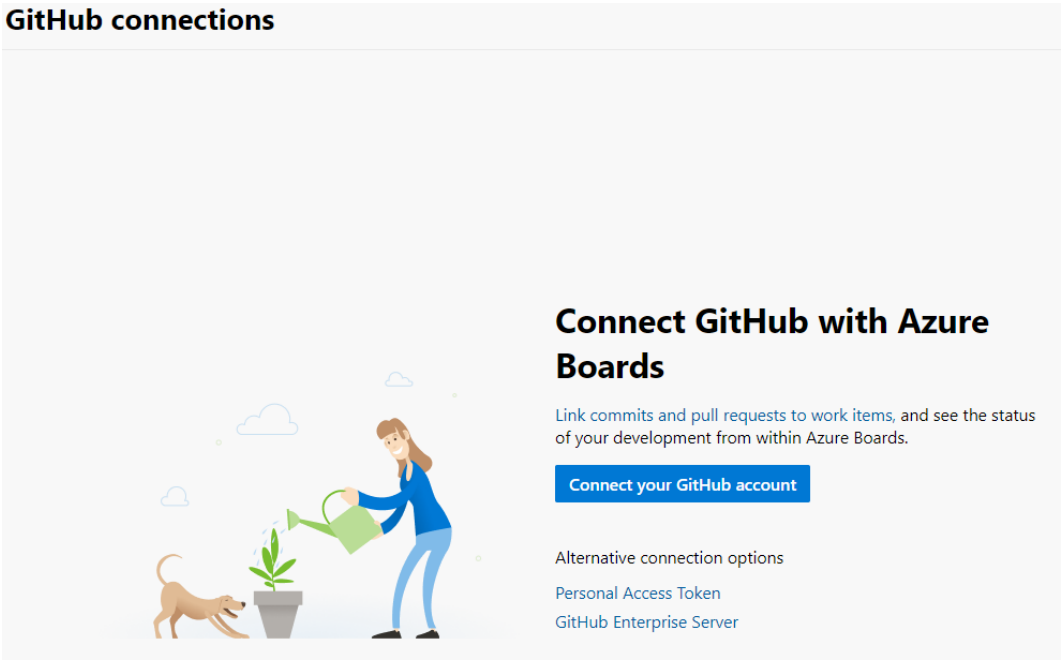


Figure 29. GitHub connections with Azure Boards

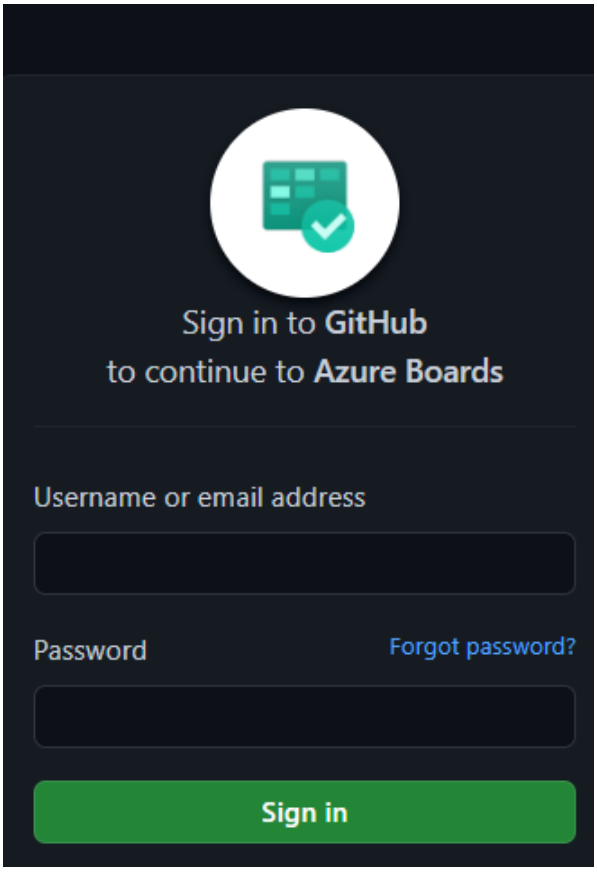


Figure 30. Sign in to authorize Azure Boards in GitHub

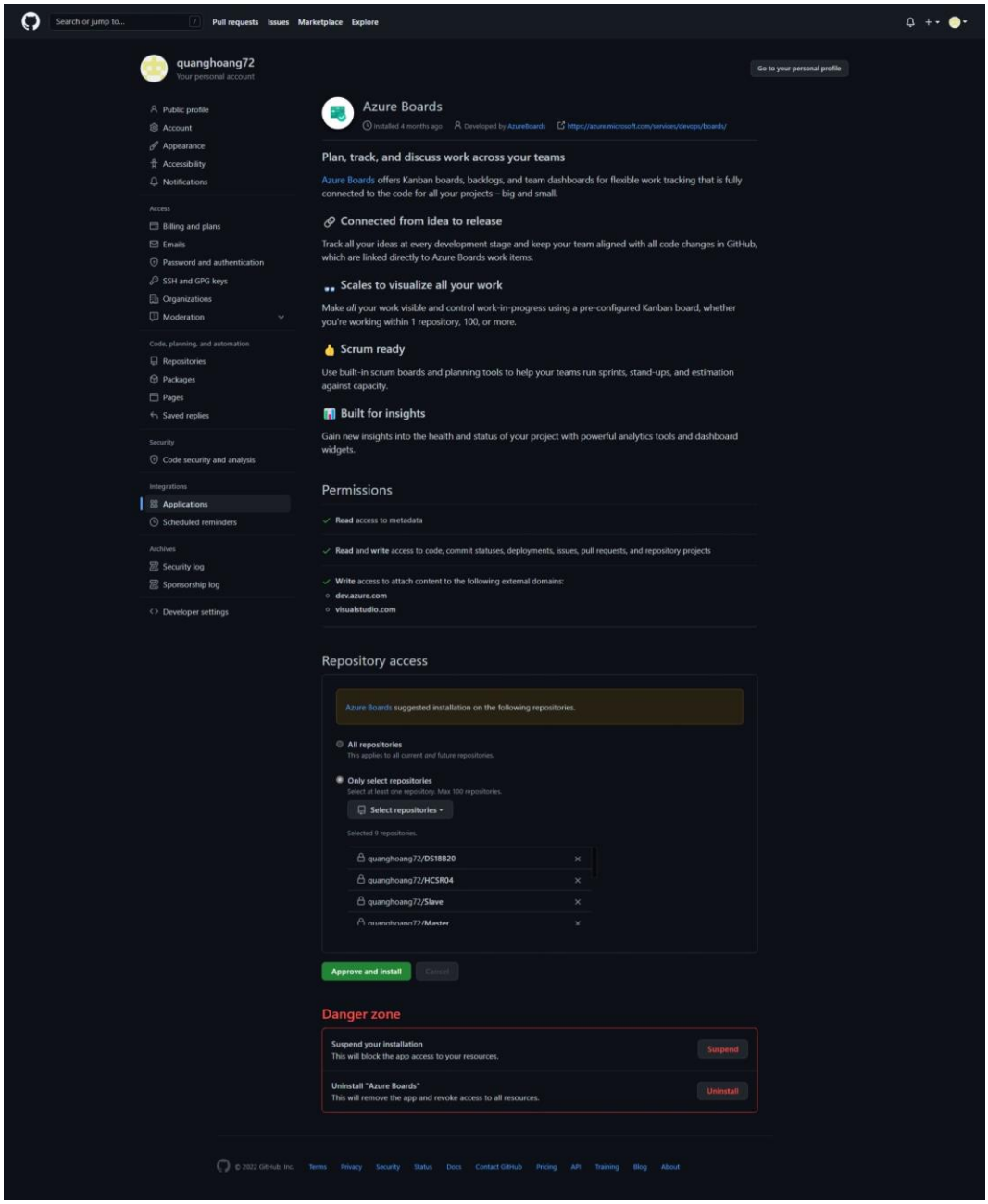


Figure 31. Azure Boards installation with GitHub

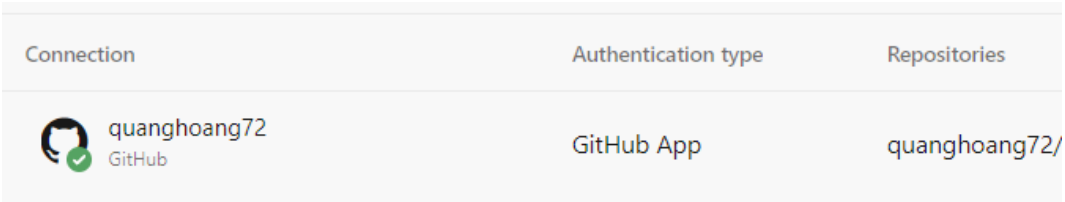


Figure 32. Successful GitHub connection to Azure Boards

The link between board and azure was done, the next connection is Visual studio code (VSCode) editor to the repository. This was performed by initializing the repository in the **Source Control** tab in VSCode.

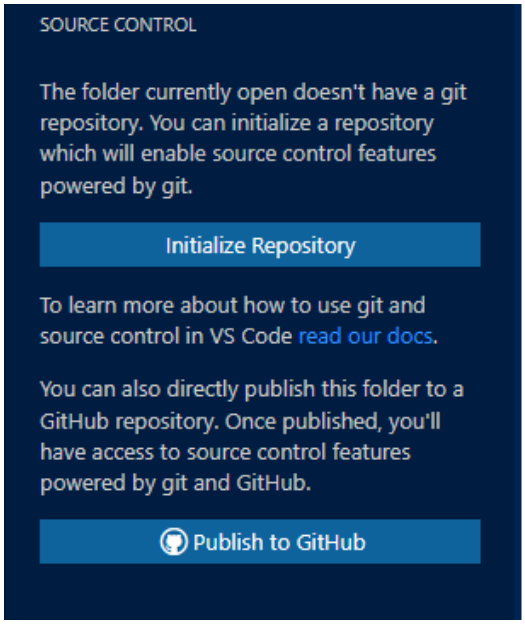


Figure 33. VSCode initialize repository

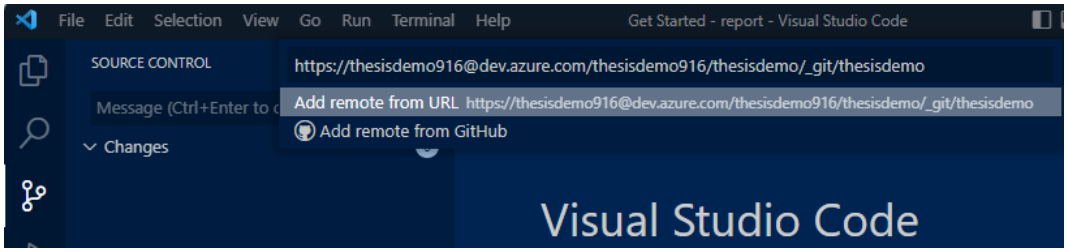


Figure 34. Adding remote repository in VSCode

At this step, this project created completely the integration environment between Azure Boards work tracking, Git repository with Azure and GitHub and Visual studio code editor.

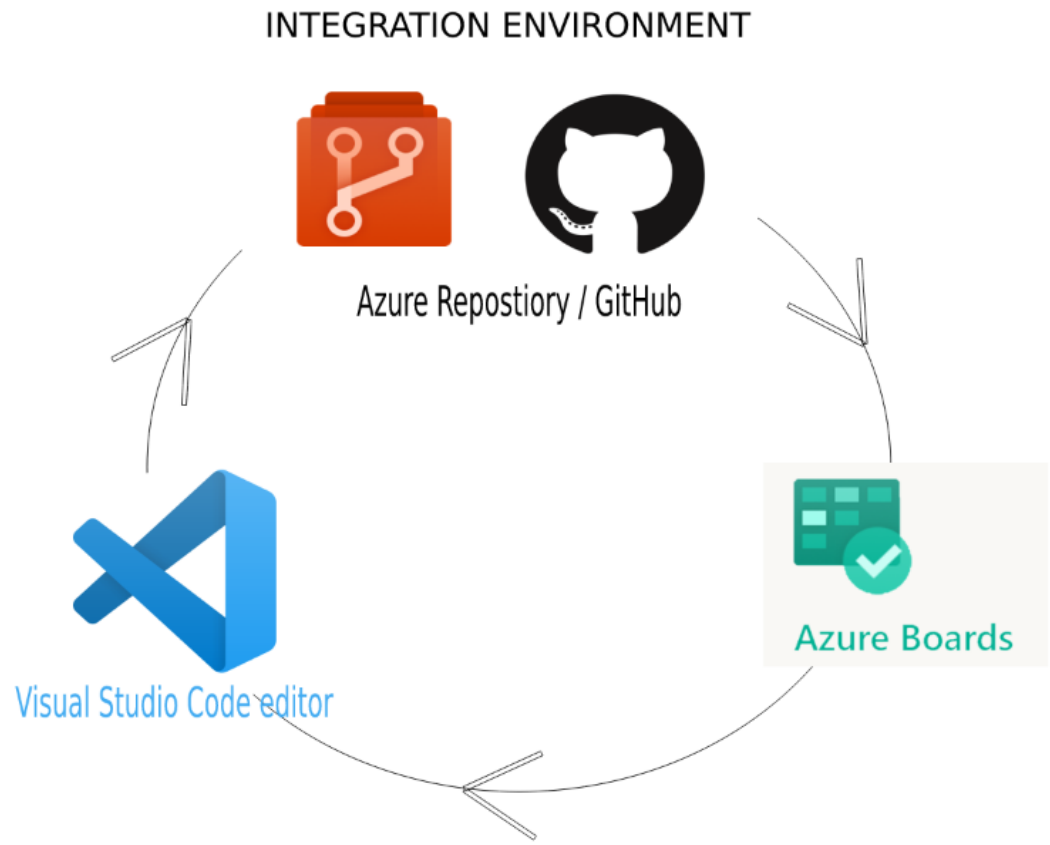


Figure 35. Visualizing integration environment

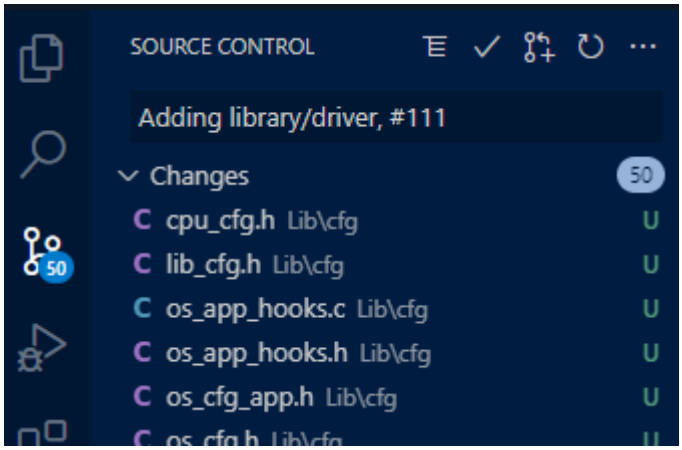


Figure 36. Commit linking with Task#111

After all these settings for environment, the next pat was for testing environment integration. The work item was linked in git commit message:

- Azure repo: “#ID”
- GitHub: “AB#ID”

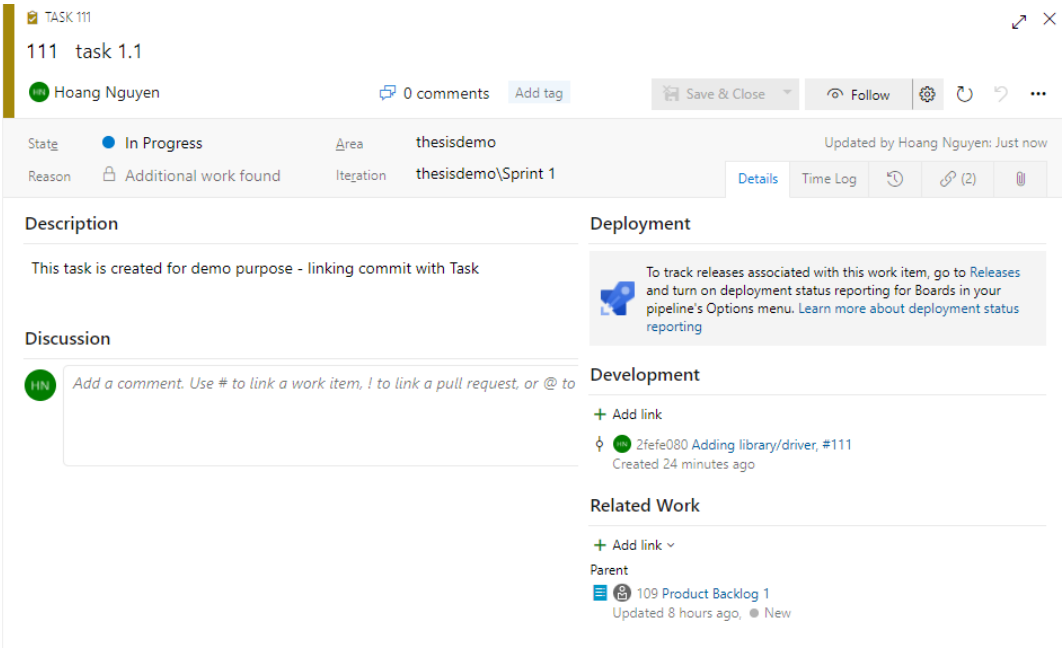


Figure 37. Task 111 had the link development commit as Figure 36 on the right

To close the task, just by adding **Fix** in commit message.

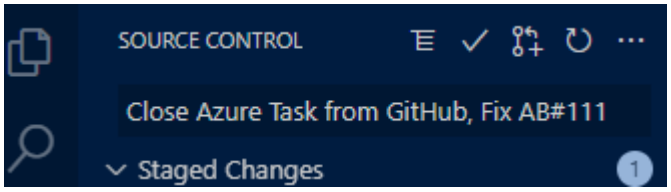


Figure 38. Commit linking to close Task#111

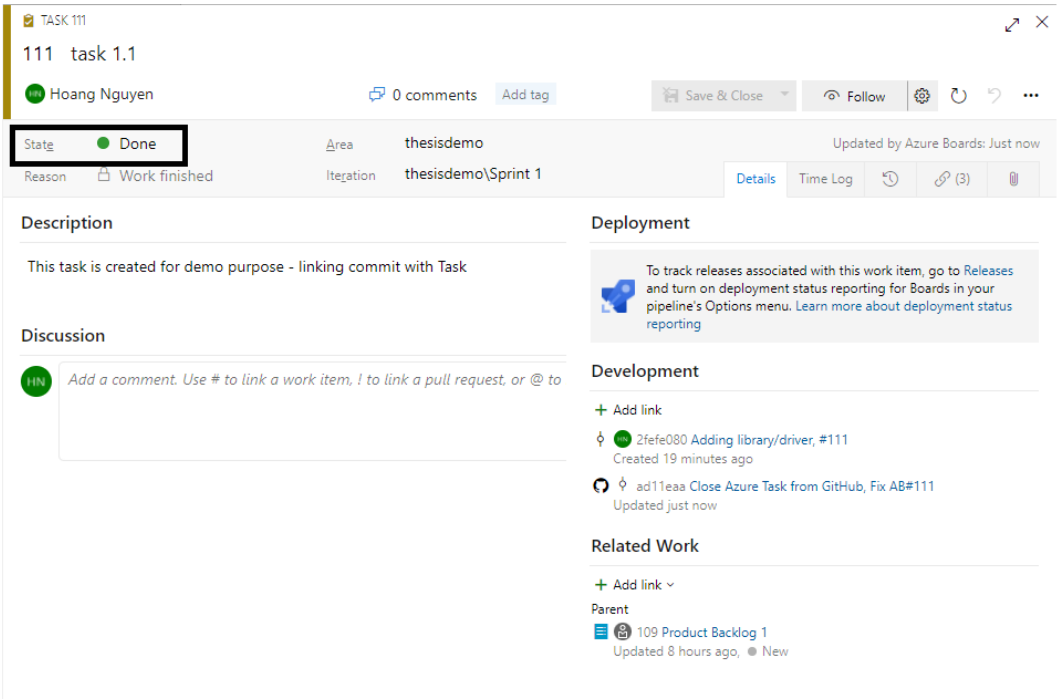


Figure 39. Task#111 was automatically transferred to done state with Fix commit

3.2.4 CI pipeline

A pipeline is defined using a YAML file which is an instruction file for guiding pipeline to run corresponding to configuration in repository. YAML (YAML ain't markup language) – a recursive acronym – is also a superset of JSON.

Using Python style indentation to indicate nesting /28/. Usually, this file named azure-pipeline.yml and is located at the root of repository.

In azure, pipeline is generated through 4 simple steps

- **Connect** to GitHub or Azure Repo Git
- **Select** the exact repository
- **Configure** the pipeline by editing the YML file
- **Review** the YML file code

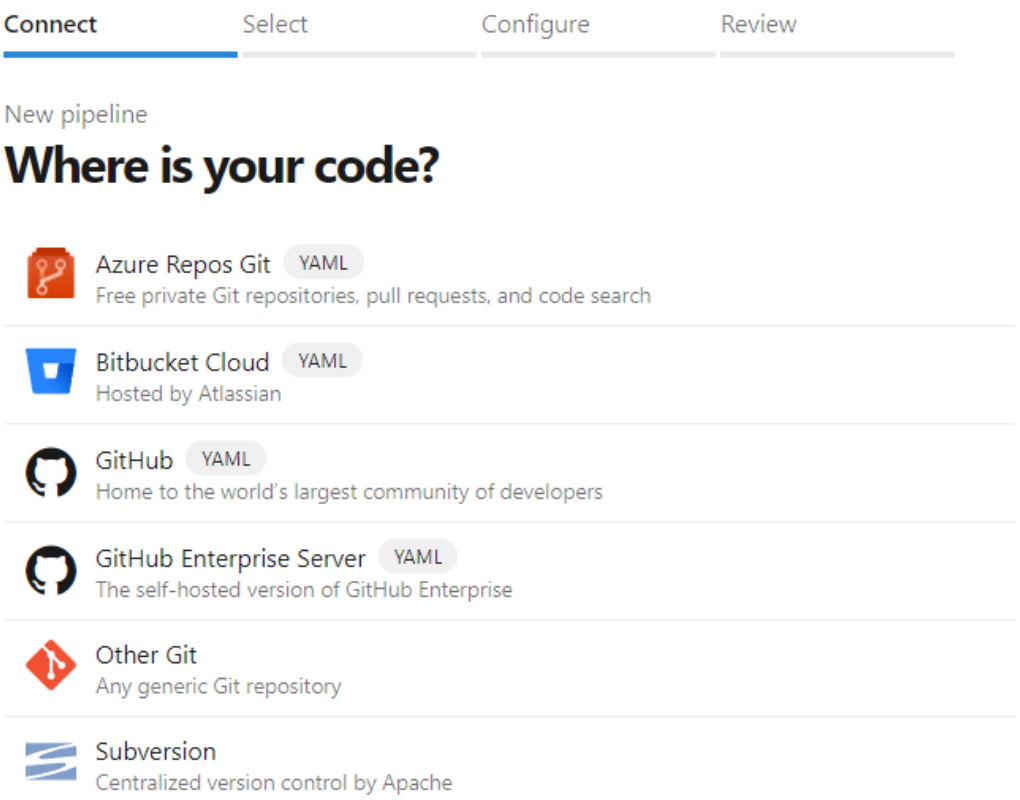


Figure 40. Connect to Azure Git repository

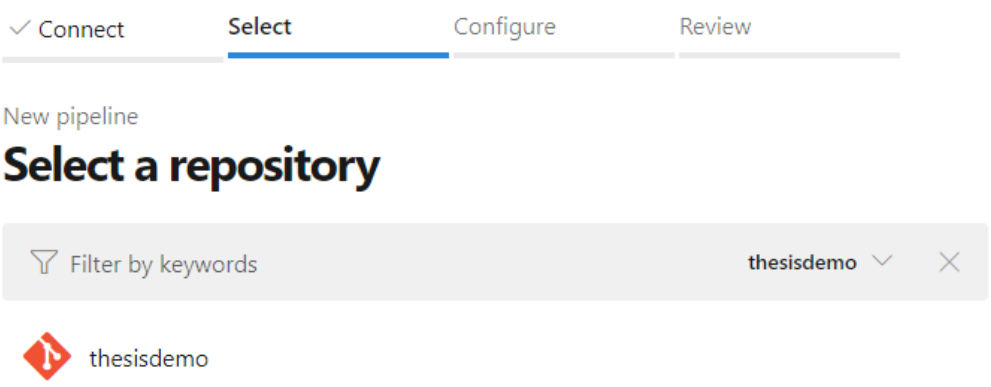


Figure 41. Select repository in Azure Git repository

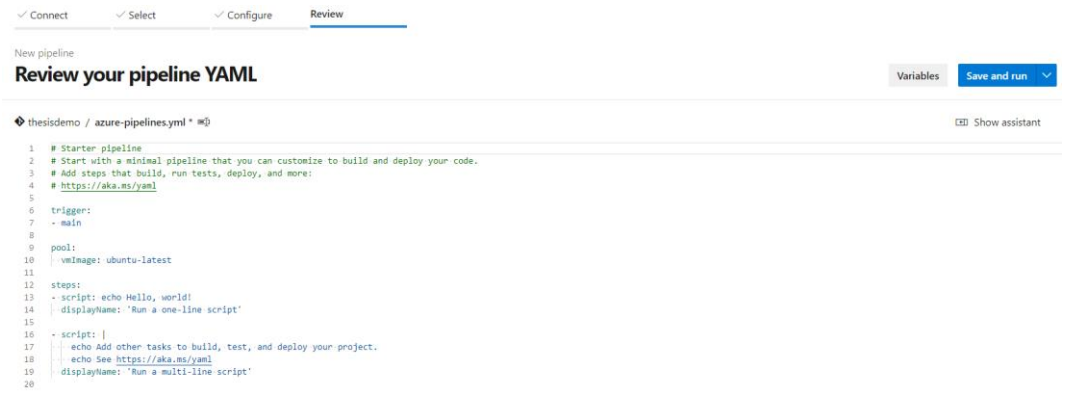


Figure 42. Editing pseudo YAML file

To automate continuous integration code, the Test branch was set as trigger branch whenever there was a change in Test. The **batch** was set to be **true** that indicated the running pipeline had to complete before starting a new one to avoid trigger changes simultaneously.

```
trigger:
  batch: true
  branches:
    include:
    - Test
    exclude:
    - main
    - Development
```

Code snippet 1. Trigger feature

This project pipeline was run only Linux agent from Microsoft-hosted agents by defining it above all the running stages.

```
pool:
  vmImage: ubuntu-latest
```

Code snippet 2. Linux virtual machine

As Figure 13, pipeline initialized by Branch Verify stage first. This stage was to check if there were already existed 3 determined branches or not. In case developers in the team forget to create, this pipeline would help them to generate Test and Development branch. The stage was divined into 2 jobs Check and Create. Check used git rev-parse command to verify branch name and returned latest commit hash ID of that branch. Otherwise, it would return failed. The next job – Check - only ran if the first job failed by marking **dependsOn** and **condition** in code. Therefore, it generated 2 branches for developers by using git branch command.

Next stage was running unit testing by Googletest. In this stage, the Makefile was used to compile the self-test code linking with Gtest library and any other necessary library. As Appendix x, **make test** was designed to compile and generate the executable file named **my_test.exe**. The Google unit test had to download its source and dependent library in order to build the self-test code base on Gtest. Therefore, Dockerfiler was the most optimized manner to pull the source code, to set the environment variable and to mount the volume eventually in just a single file. In code snippet 3, pulling the base image as **FROM ubuntu** initialized the Dockerfile which set the base image for entire subsequent instructions. Every next **RUN** instruction was an image build step that executed on top of the current image and created a new layer by committing to the container image. To accomplish building unit test code, Dockerfile ran relevant required packages installation via advance packaging tool. These packages were just to support the **make test** executaion command because the compilation and linking was persistent in Makefile.

- **Git** to run git clone
- **Gcc/ make/ g++** for building the unit Gtest code

```
FROM ubuntu


RUN apt-get update
RUN apt-get -y install git gcc make g++
RUN mkdir /opt/src
WORKDIR /opt/src
RUN git clone https://github.com/google/googletest.git

ENV GTEST_DIR=/opt/src/googletest/googletest/
WORKDIR $GTEST_DIR/src
COPY . .

RUN make test
RUN cp my_test.exe /
CMD ["./my_test.exe"]
```

Code snippet 3. Dockerfile code

Set the environment variable by **ENV** command for shorter alias name, change working directory by **WORKDIR** that performed same as CD command in Unix. Before running make command, the current directory in container had to have the source code to compile. Therefore, copy command was used to copy the entire local source code into the current directory in container by using double dot symbols respectively. Then, ran the make test command to generate the output executable file and copy that file into root directory for clarity location path.

The first task in the second stage was to build the Docker image through Dockerfile. All the instruction in the Dockerfile now layered into a package called image. Image was run and became container through docker run command in the next task named “Mounted exefile volume in Docker”. The executable file became an artifact because it was produced during build pipeline. Therefore, not only to run the executable file, but this pipeline also published that artifact. To get the file inside the container, mount volume was used to share data/file between local machine and docker container. Run docker with option **-v** to indicate a working container and mouth volume path by /Local/host/path:/inside/Container/path. In

other words, both two folders would be an imitative replication of each other, whenever one folder changes anything, this also happened to the others. The end of docker run command ended up with a overwrite shell form command that copy the my_test.exe file from root directory into /con_output folder to make a synchronization with $(Build.SourcesDirectory)/my_output. Without this mounted step, the artifact was not able to store in the build directory Azure. The Copy Task was to copy any file which had the extension of exe, dll, pdb into the specific directory before publishing anything in that directory. To run the executable artifact, the next script downloaded the artifact then gave executable permission to run the file.

Build stage divided into 2 jobs Build and Git. Firstly,in Build, it downloaded Pip Installs Packages (PIP) in order to download PlatformIO package. Pio check command performed static analysis check on PlatformIO based project, by default Cppcheck analysis tool is used. This type of analysis addresses weaknesses in source code that might lead to vulnerabilities. The –fail-on-defect flag would return 0 if code was low severity as Table 1. After running static code analysis successfully, the code was now being built by pio run command. PlatformIO will detects framework like STM32 or Arduino, etc, in platformio.ini configuration file. The built code was placed in hidden .pio/build folder. Next task installed the zip package to archive the build folder into the second artifact - zip file named build.zip. The next two steps were same as previous stage which were copy the zip file and publish it the artifact within the same directory. The last job Git was run in the last event after everything was tested and built reliably and flawlessly. This script was to merge Test branch into Development branch.

Table 1. Defect severity on PlatformIO

Severity	Meaning
High	Issues that are possibly bugs
Medium	Suggestions about defensive programming in order to prevent potential bugs
Low	Issues related to code cleanup and performance

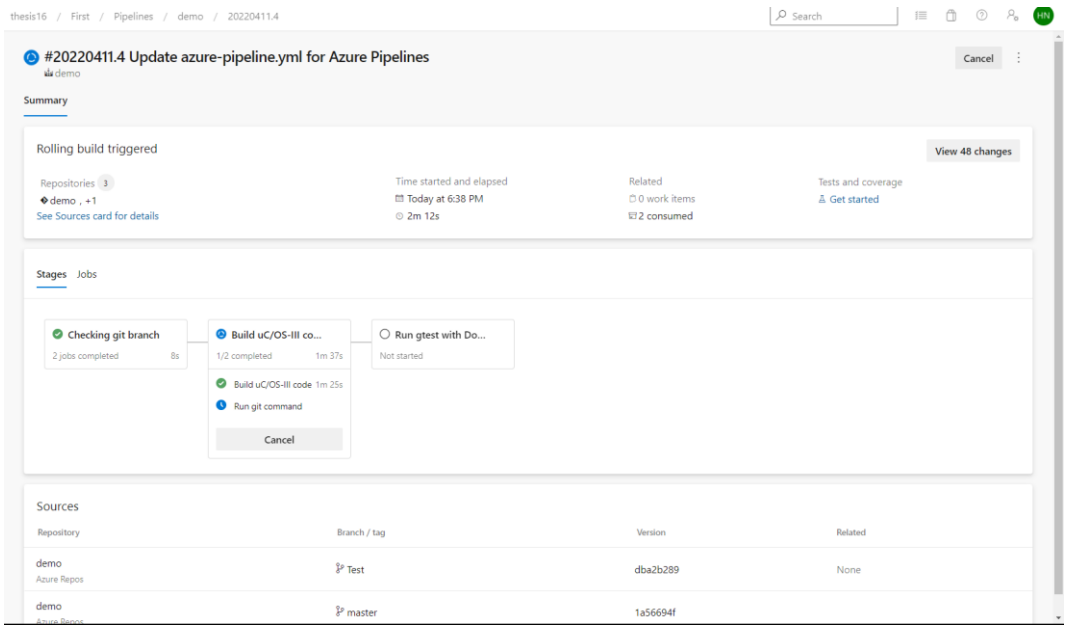


Figure 43. Running pipeline UI

To verify this pipeline worked well, another coder submitted their changing code into this repository containing auzre-pipeline.yml file. The pipeline was triggered and ran the same 3 stages, run unit googletest and build the uC code. The result was expected.

3.3 Result

After over 200 commits and with over 100 building pipeline times, I have successfully created this CI pipeline. The pipeline can create automatically specific branches now. The code is able to automate continuous integration by building and testing without manual step. Final, the Development branch is always up-to-date when Test branch build pipeline is accomplished by merging.

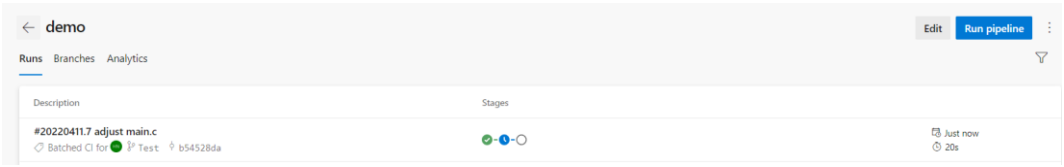


Figure 44. Batch CI for automate trigger

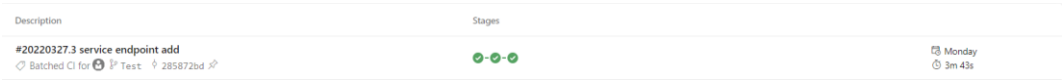


Figure 45. A Complete pipeline

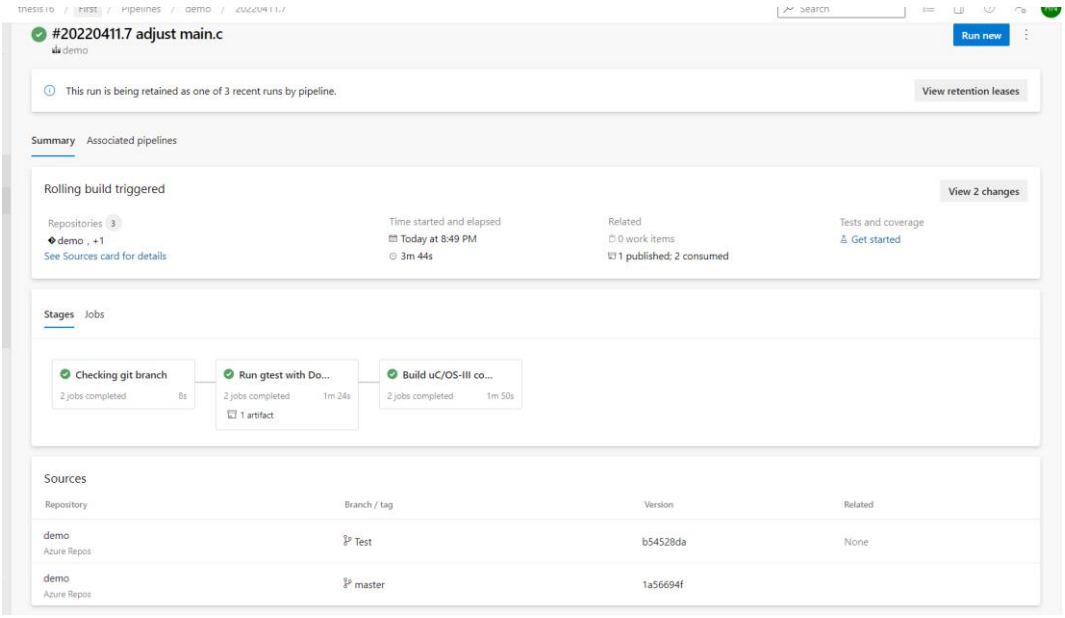


Figure 46. All stages were succeeded

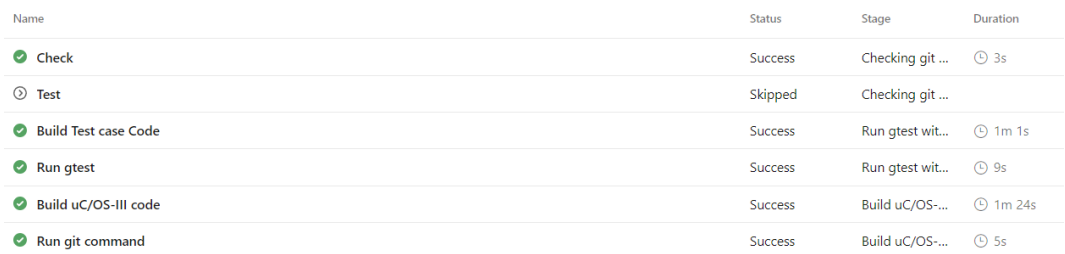


Figure 47. All jobs were succeeded

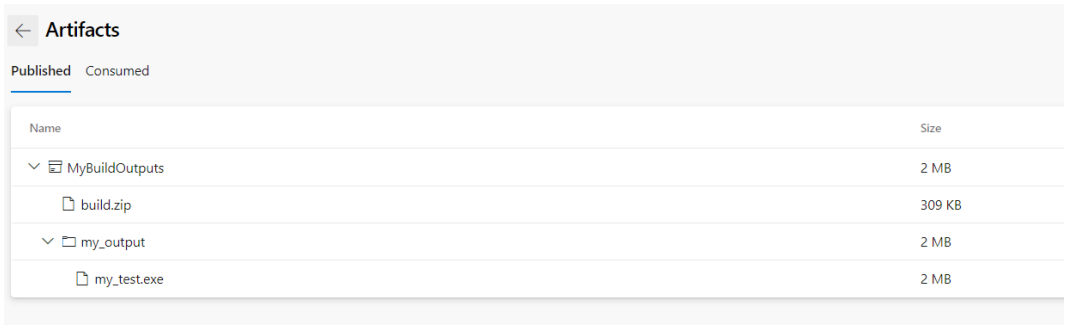


Figure 48. Two published Artifacts

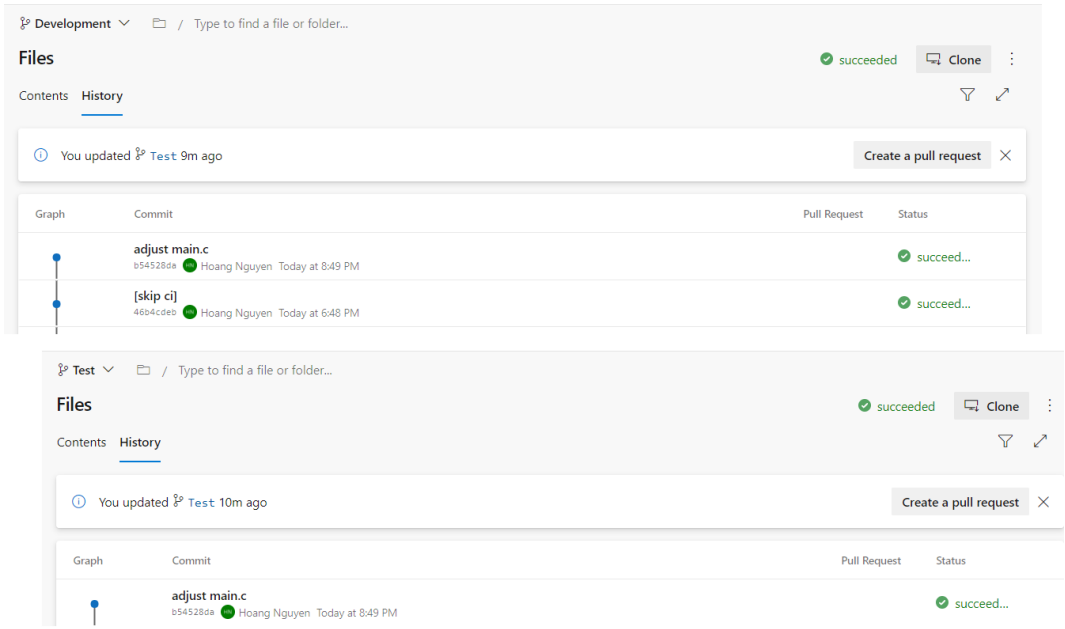


Figure 49. Development branch was merged automatically when pipeline run successfully

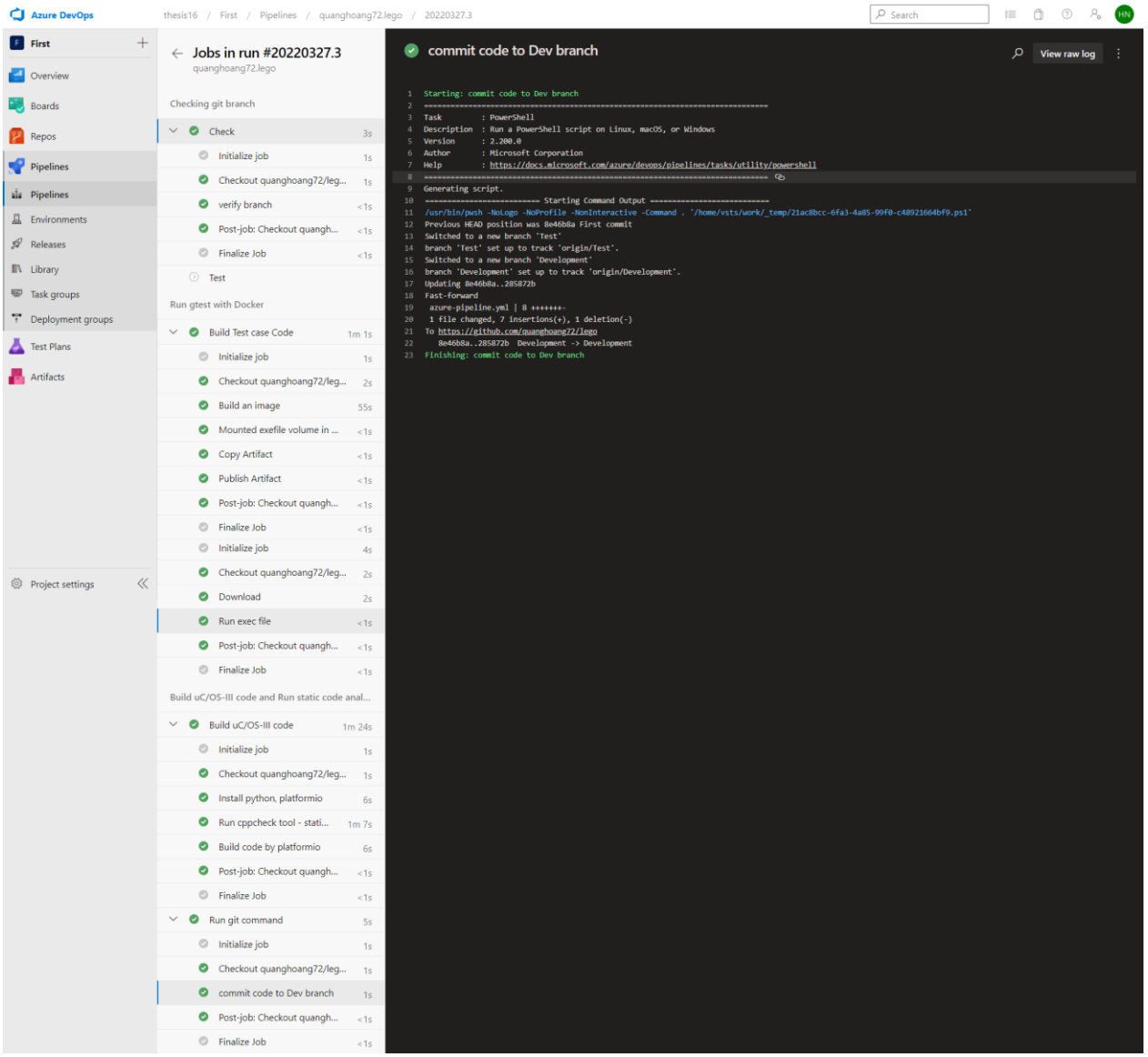


Figure 50. Total view of Pipeline tasks

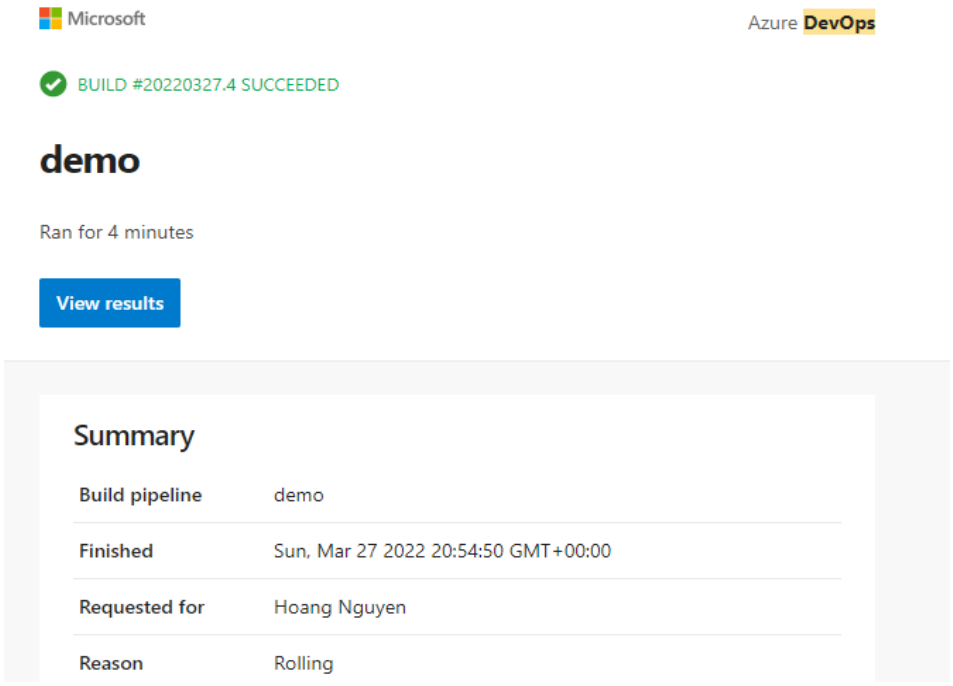


Figure 51. Notification about successful build pipeline

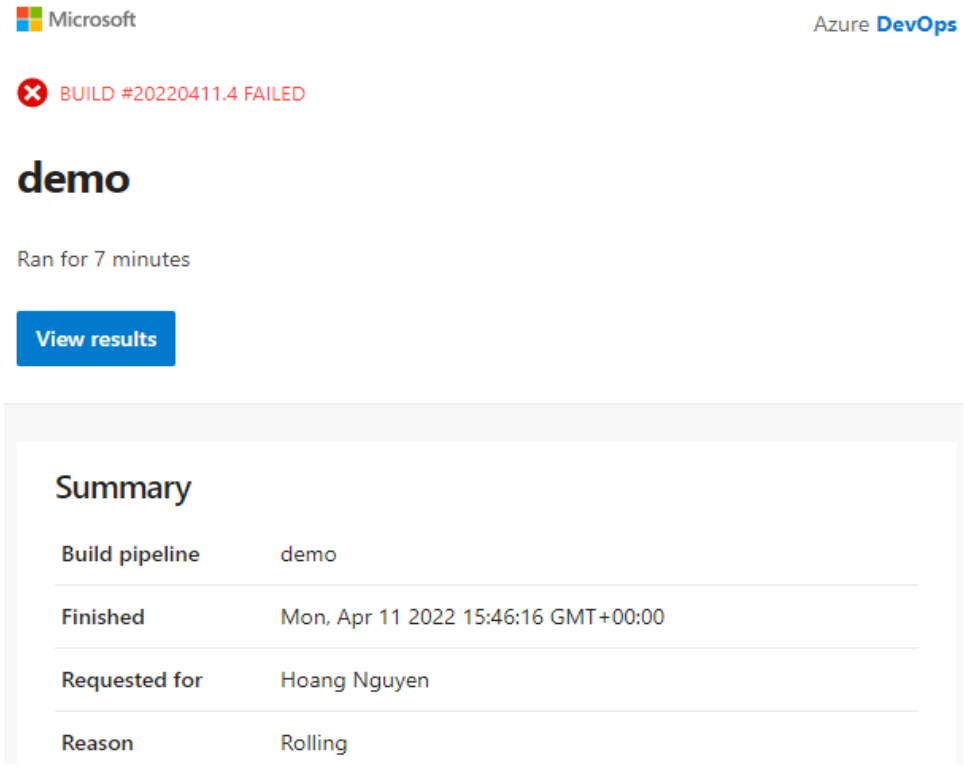


Figure 52. Notification about failed build pipeline

```yaml
# Starter pipeline

# Start with a minimal pipeline that you can customize to build and deploy y
our code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml

trigger:
  batch: true
  branches:
    include:
    - Test
    exclude:
    - main
    - Development
#pr:
 #- master

variables:
  tag: '$(Build.BuildId)'
  imageName: 'pipelines'

pool:
  vmImage: ubuntu-latest
stages:
- stage: GitBranch
  displayName: Checking git branch
  jobs:
  - job: Check
    steps:
    - checkout: git://First/demo@master
    - script:
        git rev-parse --verify origin/main ||
        git rev-parse --verify origin/Test &&
        git rev-parse --verify origin/Development
      displayName: 'verify branch'
  - job: Test
    dependsOn: Check
    condition: failed()
    steps:
    - checkout: self
      persistCredentials: true
    - script: |
          git branch Test
          git push origin Test
          git branch Development
          git push origin Development
      displayName: create branch
- stage: Test
  displayName: Run gtest with Docker
  condition: always()
  jobs:
    - job: Build
      displayName: Build Test case Code
```

Code snippet 4. Azure-pipeline.yml code

4 CONCLUSION

Beside designing and creating the pipeline, this thesis has done a deepresearch of theoretical DevOps background. The definitions of CI/CD, Agile, Scrum and DevOps are completely introduced. Moreover, the pros and cons of these methodologies/ frameworks/ cultures are introduced. The relationship between those terms is also mentioned in this thesis.

This project thesis mainly implements the CI pipeline with expected and desired results. The integration between Azure Boards, Git Repositories and Visual Studio Code editor works functionally. The Embedded project language and test was built easily and reliably with this pipeline. The code was merged after building and passed all tests. In addition, there is also a notification/report about building the pipeline status.

In future work application, this pipeline can be applied further for students in any software development life cycle majoring in Embedded System. This pipeline can also be implemented to a continuous delivery pipeline which is able to deploy artifact software/ driver or simulated product.

REFERENCES

/1/ What is Pipelining? Study Tonight. Accessed 2 April 2022. https://www.study-tonight.com/computer-architecture/pipelining

/2/ What is Software Development Life Cycle, Synopsis. Accessed on 2 April 2022. **https://www.synopsys.com/glossary/what-is-sdlc.html**

/3/ Software Development Life Cycle. BIGWATER. Accessed on 2 April 2022. https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc/

/4/ Chathmini J. 2020, Waterfall Methodology, Medium. Accessed 2 April 2022.https://medium.com/@chathmini96/waterfall-vs-agile-methodology-28001a9ca487

/5/ SDLC-Waterfall Model, Tutorialspoint. Accessed 2 April 2022. https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

/6/ Hamilton, Thomas. 2022. Agile Vs. DevOps: What's the difference? Accessed on 2 April 2022. https://www.guru99.com/agile-vs-devops.html#:~:text=KEY%20DIFFERENCE,process%20focuses%20on%20constant%20changes.

/7/ Agile essential manifesto for Agile Software Development. Accessed on 2 April 2022.https://www.agilealliance.org/agile101/the-agile-manifesto/

/8/ Mobilejon. Applying the Agile Methodology to the Modern Workplace. Mobile Jon's Blog. Accessed on 2 April 2022. https://mobile-jon.com/2021/04/05/applying-the-agile-methodology-to-the-modern-workplace/

/9/ Scrum (software development). Wikipedia. Accessed on 2 April 2022. https://en.wikipedia.org/wiki/Scrum_(software_development)

/10/ Petrova, Sandra. 7 Effective Ways to Improve Scrum Team Productivity. Adeva. Accessed on 2 April 2022. https://adevait.com/workplace/agile-software-development

/11/ Hamilton, Thomas.2022. Agile Methodology: What is Agile Model in Software Testing? Guru99. Accessed on 2 April 2022. https://www.guru99.com/agile-scrum-extreme-testing.html

/12/ What is Scru., Scrum.org. Accessed on 2 April 2022. https://www.scrum.org/resources/what-is-scrum

/13/ Drumond, Claire. Scrum, Atlassian Agile Coach. Accessed on 2 April 2022 https://www.atlassian.com/agile/scrum.

/14/ Kaya Ismail.2018. Agile vs DevOps: What's the Difference? Accessed on 2 April 2022.https://www.cmswire.com/information-management/agile-vs-devops-whats-the-difference/

/15/ DevOps, The Office of the Chief Software Officer. Accessed on 2 April 2022.https://software.af.mil/training/devops/

/16/ Hamilton, Thomas.lt 2022. Agile vs DevOps: What's the difference?, Guru99. Accessed on 2 April 2022. https://www.guru99.com/agile-vs-devops.html#:~:text=KEY%20DIFFERENCE,process%20focuses%20on%20constant%20changes.

/17/ What is CI/CD, 2018, Accessed on 3 April 2022. https://www.redhat.com/en/topics/devops/what-is-ci-cd?sc_cid=7013a000002pwNlAAI&gclid=Cj0KCQjw6J-SBhCrARIsAH0yMZjzRPXwm4ljsCoQnTeqR7pJzzlUPdWWKxB6jEpGQ83lD9VfSy_t8VcaArDNEALw_wcB&gclsrc=aw.ds

/18/ Prinze, Suzie. 2016. The Product Managers' Guide to Continuous Delivery and DevOps, mindthePRODUCT. Accessed on 3 April 2022 https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheat-sheet-for-the-rest-of-us/

/19/ Continuous testing. 2021. Wikipedia. Accessed on 3 April 2022.https://en.wikipedia.org/wiki/Continuous_testing

/20/ CI-CD as a manufacturing process., CRM trilogix. Accessed on 3 April 2022. https://crmtrilogix.com/Cloud-Blog/Application-Transformation/CI-CD-as-a-manufacturing-process/321

/21/ Continuous Integration vs Continuous Delivery vs Continuous Deployment #CICD, DevTipCurator. Accessed on 3 April 2022. https://devtipscurator.wordpress.com/2017/02/03/continuous-integration-vs-continuous-delivery-vs-continuous-deployment/

/22/ What is Azure DevOps.2022. Microsoft documentation. Accessed on 3 April 2022.https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

/23/ What is Azure Boards. 2022. Microsoft documentation. Accessed on 3 April 2022. https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops

/24/ What is Azure Pipelines. 2022. Microsoft documentation. Accessed on 3 April 2022.https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops

/25/ What is PlatformIO, PlatformIO. Accessed on 3 Apri 2022. https://docs.platformio.org/en/latest/what-is-platformio.html

/26/ Visual studio code. Accessed on 3 April 2022. https://code.visualstudio.com/

/27/ Manage your Scrum process work item types and workflow, Microsoft Documentation. Accessed on 3 Apirl 2022. https://docs.microsoft.com/en-us/azure/devops/boards/work-items/guidance/scrum-process-workflow?view=azure-devops

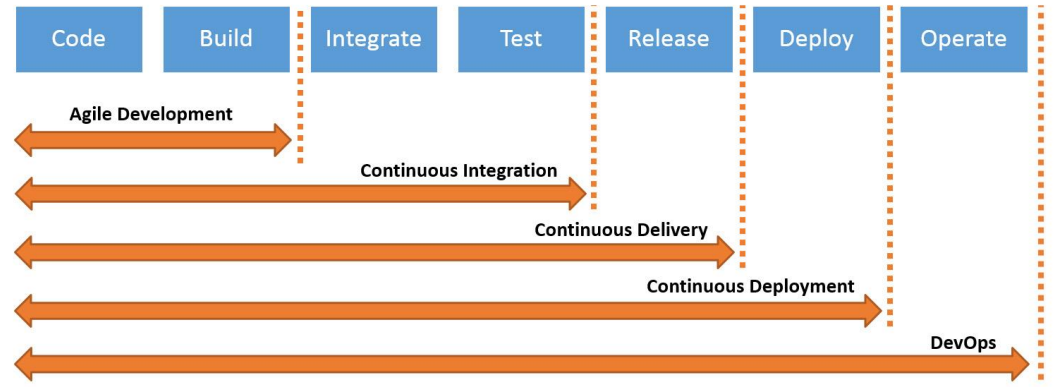
/28/ What is YAML, 2021, RetHat. Accessed on 3 April 2022. https://www.redhat.com/en/topics/automation/what-is-yaml

/29/ Docker overview, Docker Documentation. Accessed on 3 April 2022 https://docs.docker.com/get-started/overview/

/30/ Manage your Scrum processes work item types and workflow, 2022. Accessed on 3 April 2022. https://docs.microsoft.com/en-us/azure/devops/boards/work-items/guidance/scrum-process-workflow?view=azure-devops

APPENDIX 1

Comparison Agile vs CI/CD vs DevOps



APPENDIX 2

Makefile Code

```
#make  - to compile normal run
#make test - to compile for unit testing
### declare directory
GTEST_DIR=/opt/src/googletest/googletest
SRC_DIR=source
TCASE_DIR=test_case
INC_DIR=include

### declare variable
PROJ=project_functions
TCASE := $(or $(TCASE_DIR)/*.c, $(TCASE_DIR)/*.cpp)
DEPS= $(INC_DIR)/$(PROJ).h

### FLAGS
COPTS=-Wall -funsigned-char -fpermissive -fprofile-arcs -ftest-coverage
LDFLAGS=-lm -lgcov --coverage
CXXFLAGS= -isystem $(GTEST_DIR)/include -I$(GTEST_DIR)


######### Main targets ##########
release: main.o $(SRC_DIR)/$(PROJ).o
    $(CC) $^ -o $@ $(LDFLAGS)

#### We need -pthread as Google Test uses thread
#### Build a test depends on test_case foldler #####
.PHONY:test
test: test_file.o $(PROJ)_test.o libgtest.a
    $(CXX)  $(LDFLAGS) $(CXXFLAGS) -pthread $^ -o my_test.exe

########## Normal - Release ############
*.o: $(SRC_DIR)/*.c
    $(CC) -c $<

########## Unit test - test ########
$(PROJ)_test.o: $(SRC_DIR)/$(PROJ).c $(DEPS)
    $(CXX) $(CXXFLAGS) -c $< -I$(INC_DIR) -o $@

###test_case_file
test_file.o: $(TCASE) $(DEPS)
    $(CXX)  $(CXXFLAGS) -c $< -I$(INC_DIR) -I-$(COPTS)

########## Google Test framework ############
#For simplicity and to avoid depending on Google Test's
# implementation details, the dependencies specified below are
# conservative and not optimized.  This is fine as Google Test
```