CI/CD with Gerrit, AI-Enhanced Review, and Hardware-in-the-Loop Testing in Jenkins Pipelines

Amarula Solutions

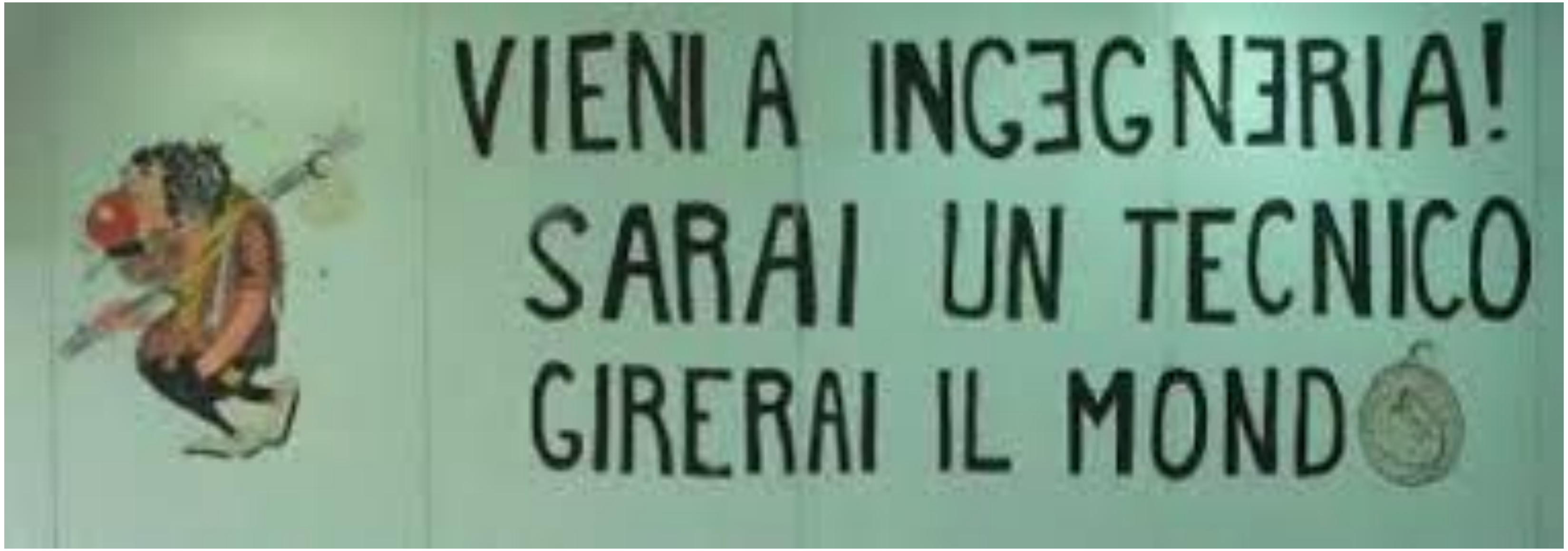
ITALY
Amarula Solutions SRL
*Via F. Cavallotti, 25D, 41012 Carpi *MO)*
+39-0422/435310

Michael Trimarchi
michael@amarulasolutions.com
https://github.com/panicking

About me

- Embedded Linux engineer at **Amarula Solutions**:
  - Embedded Linux and Android expertise
  - Development, consulting and training
  - Open source projects
- Open source contributor
  - Buildroot
  - Linux
    - Contributor to Linux Scheduler (Long time ago)
    - Developed the hiface audio driver
  - U-Boot
    - Co-Custodian with Dario Binacchi for NAND subsystem
  - Jenkins plugin contribuitor

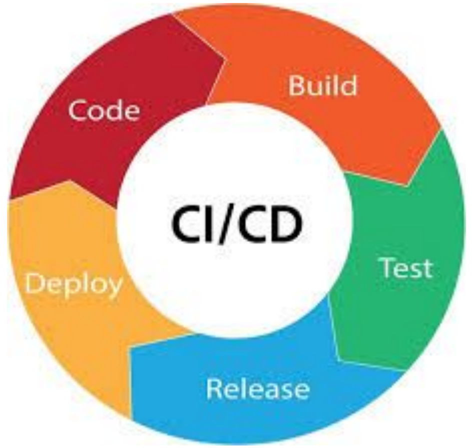
You are an engineering, you will visit the world



Things are getting better…

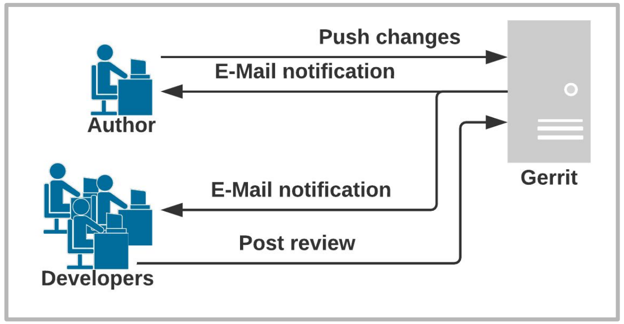
Continuous integration

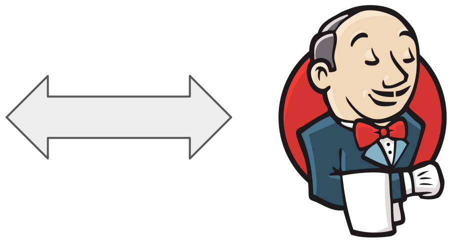
- What is Continuous Integration?
- do we need?
- Why we should adopt?
- Different phases of adopting ***Continuous Integration***
- How CI/CD tools and jenkins can help us



What is Continuous Integration?

- Developers commit code to a shared repository on a regular basis.
- The version control system is monitored, and a build is automatically triggered when a commit is detected.
- Developers will receive immediate notifications through platforms like email, Mattermost, or Slack if the build fails, test are not passing, static analysis fail, etc..



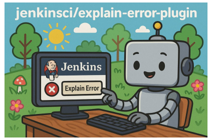


Why do we need Continuous Integration?

- The goal is to identify issues and bugs early in the development process.
- Constant integration, building, and testing of the full codebase helps us catch bugs and errors earlier in the development cycle, which means better software quality.

An open source continuous Integration?

- Jenkins CI/CD
- Gerrit for code review
- Notification to team using Mattermost
- Static analysis
  - Sonarqube
  - Codechecker
- AI Tools
  - Explain error
  - AI code review



Gerrit AI Plugin

What is Jenkins?

- Automation tool
  - CI/CD
  - Administration
  - Maintenance
- Extensible with plugins and Shared Libraries
- Web UI
- Customisable access control

amarula solutions

Jenkins integration

- Many plugins and support in general to integrate with other tools

Jenkins integration

- Many plugins and support in general to integrate with other tools
- Most notable
  - Gerrit Trigger

Gerrit Trigger

- Triggers builds on Gerrit events
- Change verification before merging
- Update/Deploy on merge

Jenkins integration

- Many plugins and support in general to integrate with other tools
- Most notable
  - Gerrit Trigger
  - Artifact Managers

Artifact Managers

- Upload Artifacts to different stores like AWS S3, MS Azure by default

Jenkins integration

- Many plugins and support in general to integrate with other tools
- Most notable
  - Gerrit Trigger
  - Artifact Managers
  - Lockable Resources

Lockable Resources

- Synchronize access to a resource or section of code

Jenkins integration

- Many plugins and support in general to integrate with other tools
- Most notable
  - Gerrit Trigger
  - Artifact Managers
  - Lockable Resources
  - Warning-ng

Warning-ng

- plugin is a Jenkins post-build tool that collects, analyzes, and visualizes compiler warnings or issues from static analysis tools, providing a comprehensive dashboard to track code quality trends over time.

Jenkins integration

- Many plugins and support in general to integrate with other tools
- Most notable
  - Gerrit Trigger
  - Artifact Managers
  - Lockable Resources
  - Warning-ng
  - Explain error plugin

Warning-ng

- AI-powered Jenkins tool that automatically analyzes console logs from failed builds to provide human-readable explanations and actionable fix suggestions, eliminating the need to manually interpret complex error traces.

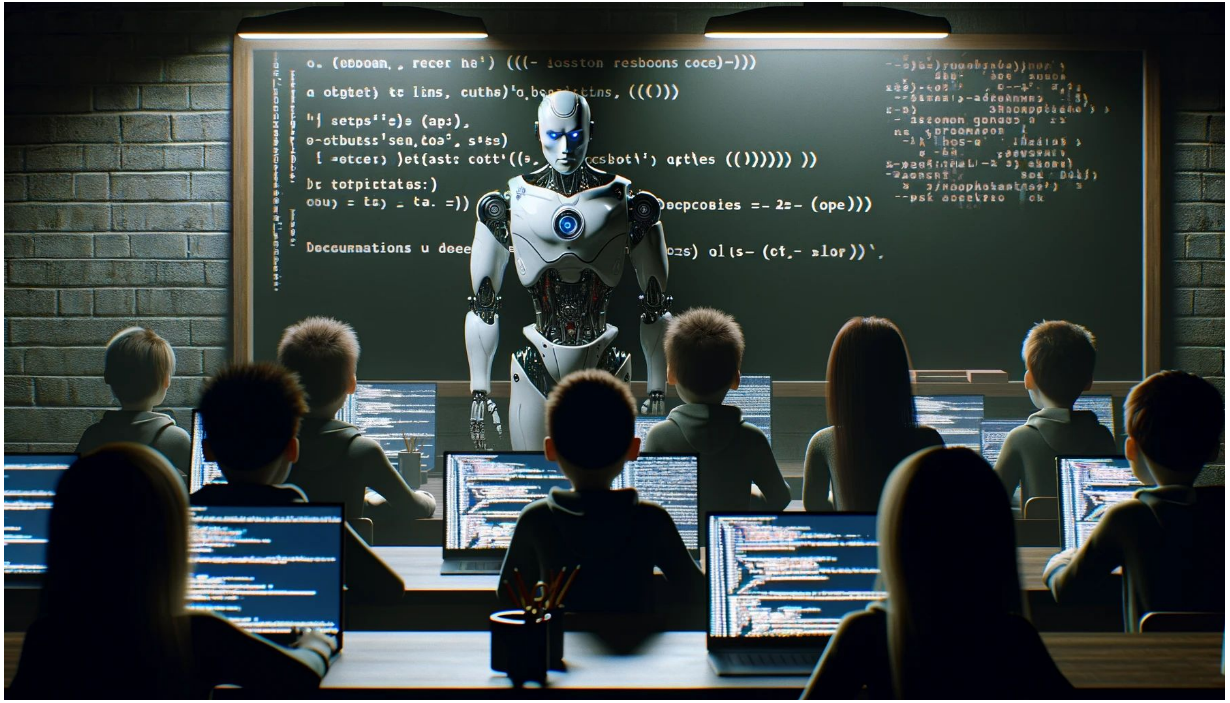
Why Adopting continuous integration?

- Better Business results
- Bringing product to market faster
- Being more flexible during the entire project duration
- Better collaboration between our team members
- Improving overall product quality

Code review

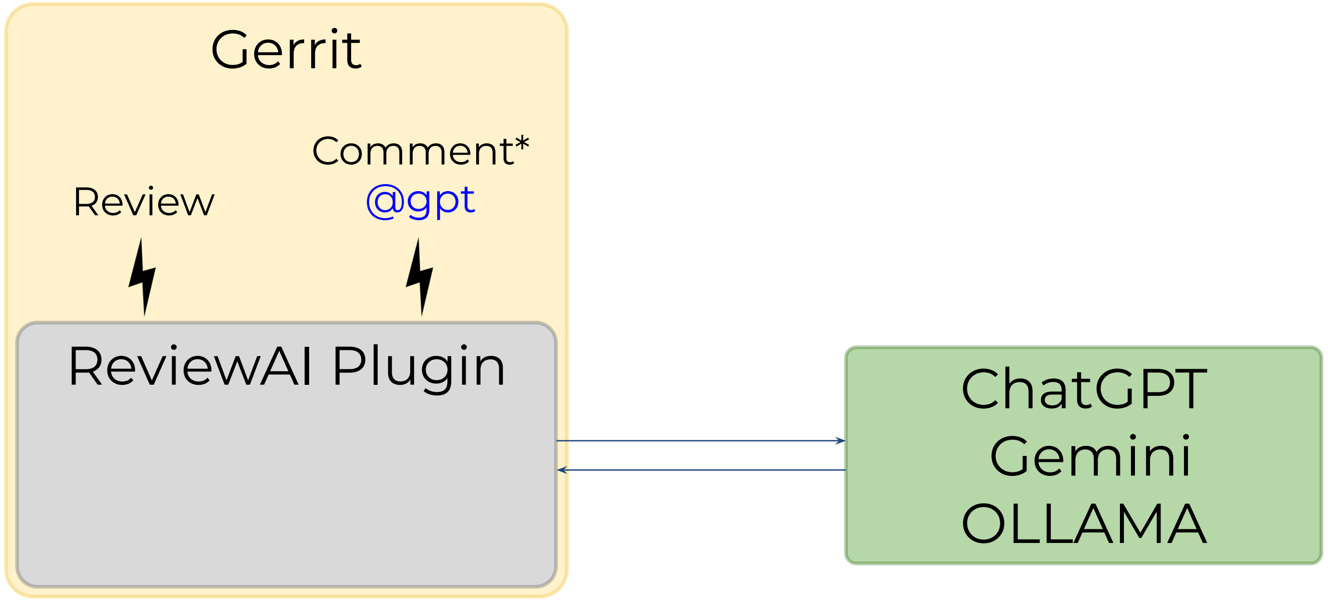
1. Code review is performed by a developer that did not participate in the development phase.
2. The code review is to ensure good code quality, modularity as well as to ensure that unit and integration tests are correctly specified and sufficient.
3. If the code review is not passed it is sent back to the developer along with comments and remarks.
4. If the code review is passed the feature will move on to testing. The topic branch will then be merged into a common code branch name master (or any branch code that is consider the release one)

AI Code Reviewer: **Nightmare**?...



amarula solutions

AI Plugin Workflow



*Comment = Question | Request | Command
https://github.com/amarula/reviewai-gerrit-plugin

The initial challenge

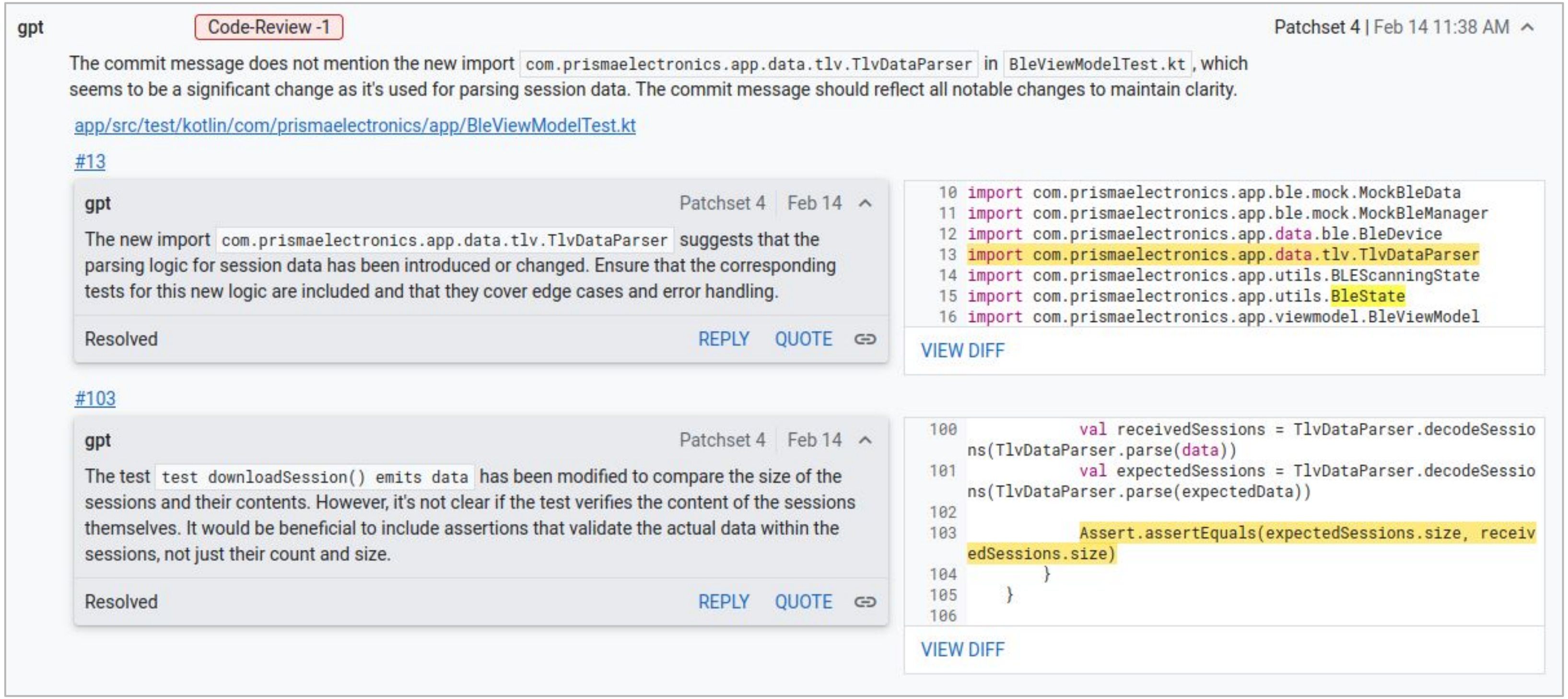
At Amarula Solutions, we extensively utilize Gerrit for our development processes, leveraging its robust code review and collaboration features. In light of its features, we concluded that the pairing of Gerrit and AI was a great fit. AI, as a Virtual Gerrit User, can:

- Offer Insights and Suggestions
- Provide Automated Code Analysis
- Votes
- Respond to Queries from Developers

Current status of the plugin

- Our journey began forking a basic version of the Gerrit plugin that integrated ChatGPT for reviewing code at Patch Set submissions.
- We updated the plugin to include new functionalities:
  - Capability for AI to respond to Inline Comments.
  - Functionality for the plugin to cast Votes.
  - Ability to interpret Commands within comments.
  - Submission of interaction history data to ChatGPT for context.
  - File diff selective submission to ChatGPT for review, filtered by specified file extensions.
  - Reviews filtered by author, group, or topic.
  - Support multiple different AI models.
- We published the project on GerritHub to have contributions by Open Source Community.

Use Case: Code Review at every Submission



gpt Code-Review -1 Patchset 4 | Feb 14 11:38 AM

The commit message does not mention the new import `com.prismaelectronics.app.data.tlv.TlvDataParser` in `BleViewModelTest.kt`, which seems to be a significant change as it's used for parsing session data. The commit message should reflect all notable changes to maintain clarity.

app/src/test/kotlin/com/prismaelectronics/app/BleViewModelTest.kt

#13

gpt Patchset 4 Feb 14

The new import `com.prismaelectronics.app.data.tlv.TlvDataParser` suggests that the parsing logic for session data has been introduced or changed. Ensure that the corresponding tests for this new logic are included and that they cover edge cases and error handling.

Resolved REPLY QUOTE

```
10 import com.prismaelectronics.app.ble.mock.MockBleData
11 import com.prismaelectronics.app.ble.mock.MockBleManager
12 import com.prismaelectronics.app.data.ble.BleDevice
13 import com.prismaelectronics.app.data.tlv.TlvDataParser
14 import com.prismaelectronics.app.utils.BLEScanningState
15 import com.prismaelectronics.app.utils.BleState
16 import com.prismaelectronics.app.viewmodel.BleViewModel
```

VIEW DIFF

#103

gpt Patchset 4 Feb 14

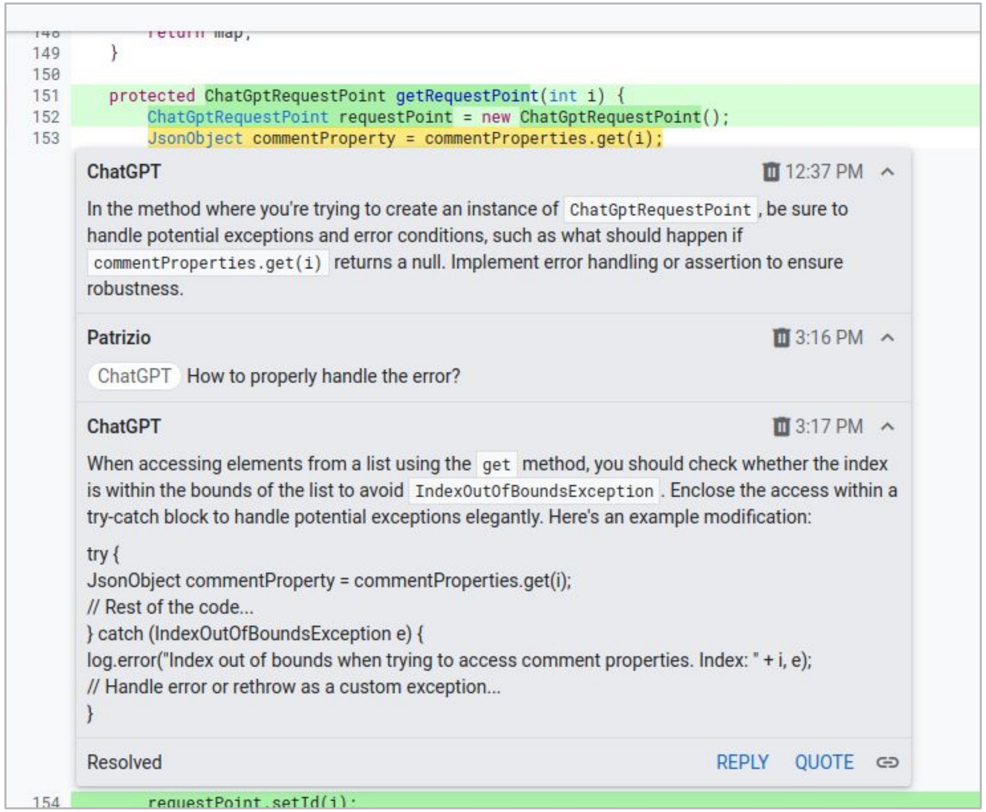
The test `test downloadSession() emits data` has been modified to compare the size of the sessions and their contents. However, it's not clear if the test verifies the content of the sessions themselves. It would be beneficial to include assertions that validate the actual data within the sessions, not just their count and size.

Resolved REPLY QUOTE

```
100             val receivedSessions = TlvDataParser.decodeSessio
    ns(TlvDataParser.parse(data))
101             val expectedSessions = TlvDataParser.decodeSessio
    ns(TlvDataParser.parse(expectedData))
102
103             Assert.assertEquals(expectedSessions.size, receiv
    edSessions.size)
104         }
105     }
106
```

VIEW DIFF

Use Case: Inline Code Interactions



```
        return map;
    }

    protected ChatGptRequestPoint getRequestPoint(int i) {
        ChatGptRequestPoint requestPoint = new ChatGptRequestPoint();
        JsonObject commentProperty = commentProperties.get(i);
```

ChatGPT 12:37 PM

In the method where you're trying to create an instance of `ChatGptRequestPoint`, be sure to handle potential exceptions and error conditions, such as what should happen if `commentProperties.get(i)` returns a null. Implement error handling or assertion to ensure robustness.

Patrizio 3:16 PM

ChatGPT How to properly handle the error?

ChatGPT 3:17 PM

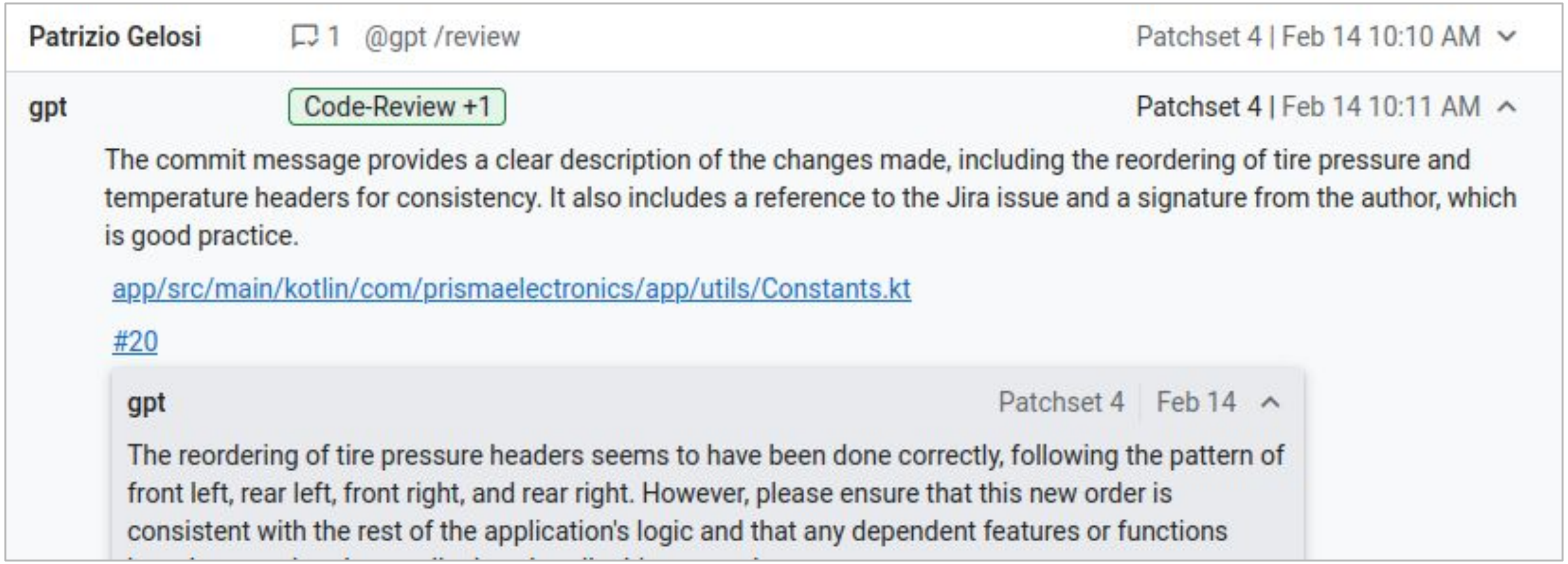
When accessing elements from a list using the `get` method, you should check whether the index is within the bounds of the list to avoid `IndexOutOfBoundsException`. Enclose the access within a try-catch block to handle potential exceptions elegantly. Here's an example modification:

```
try {
JsonObject commentProperty = commentProperties.get(i);
// Rest of the code...
} catch (IndexOutOfBoundsException e) {
log.error("Index out of bounds when trying to access comment properties. Index: " + i, e);
// Handle error or rethrow as a custom exception...
}
```

Resolved REPLY QUOTE

```
        requestPoint.setId(i);
```

Use Case: Command Support in Comments



Patrizio Gelosi 1 @gpt /review Patchset 4 | Feb 14 10:10 AM

gpt Code-Review +1 Patchset 4 | Feb 14 10:11 AM

The commit message provides a clear description of the changes made, including the reordering of tire pressure and temperature headers for consistency. It also includes a reference to the Jira issue and a signature from the author, which is good practice.

app/src/main/kotlin/com/prismaelectronics/app/utils/Constants.kt

#20

gpt Patchset 4 Feb 14

The reordering of tire pressure headers seems to have been done correctly, following the pattern of front left, rear left, front right, and rear right. However, please ensure that this new order is consistent with the rest of the application's logic and that any dependent features or functions

Plugin roadmap

- Facilitating interactions with tailor-made Assistants to accumulate and apply Project-Specific and Organizational Knowledge. (already started with ai-instructions.md patchset)
- Integrate it on gerrit-flow new feature. Allow the plugin to be trigger on specific CI/CD results or step (example, build is ok, etc)
- Allow to apply suggested code changes on the interface
- Integrate better with the latest gerrit API for AI

Testing

1. Changeset is always tested using automation
2. When code changes have been merged QA topic build is launched and integration tests are executed.
3. Afterwards the build is passed to the QA team for functional requirements testing and regression testing, if there no automatic testable functionality.
4. Security tests are also automatically run
5. Platform testing is always done on devices that replicate production environment (you can run the test on several hardware variants in parallel)

What are the components required to…?

- Tool for tracking activities and tasks/bugs
- Documentation space
- Source code repository
- Hybrid CI infrastructure (cloud + on premise). This solve hardware testing on heavy building tasks
- Artifactory storage. Archive our libraries, OS, releases
- Download proxy cache and state cache. CI cost in time and developer wait for some build result
- Easy way to communicate between team and external team (Chat, call etc)
- Static analyzer tools and security tools

Jenkins and testing

Jenkins

- Allow to create the pipeline to run testing
- Automate our process for product delivery
- Allow to run testing direct on the hardware using Labgrid

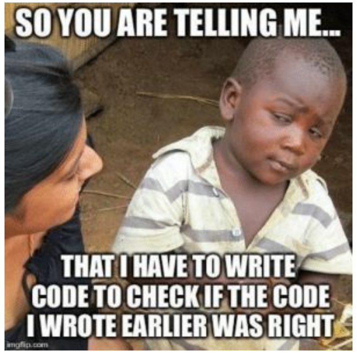
Unit Test

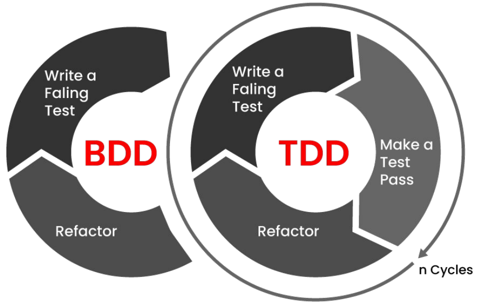
Pro:
- Unit tests help to fix bugs early in the development cycle and save costs.
- It helps the developers to understand the testing code base and enables them to make changes quickly.
- Good unit tests serve as project documentation.
- Unit tests help with code reuse.
- Debugging processes are made easier.

Cons:
- More lines of test code may need to be written to test one line of code-creating a potential time investment.

Tools:
- Google Test
- Catch2 Test
- Boost test
- Qt Squish

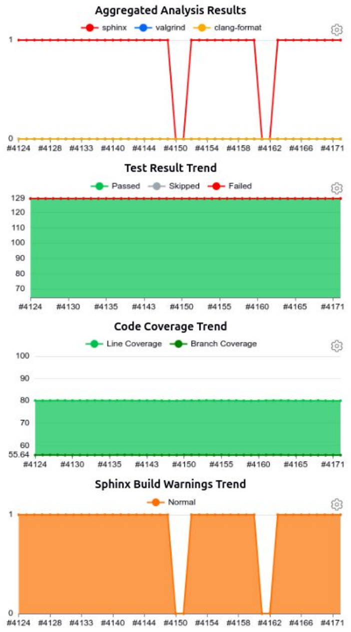
Some references:
- https://en.wikipedia.org/wiki/Unit_testing
- https://en.wikipedia.org/wiki/Test-driven_development
- https://mindmajix.com/tdd-vs-bdd
- https://github.com/google/googletest


SO YOU ARE TELLING ME...
THAT I HAVE TO WRITE CODE TO CHECK IF THE CODE I WROTE EARLIER WAS RIGHT
imgflip.com

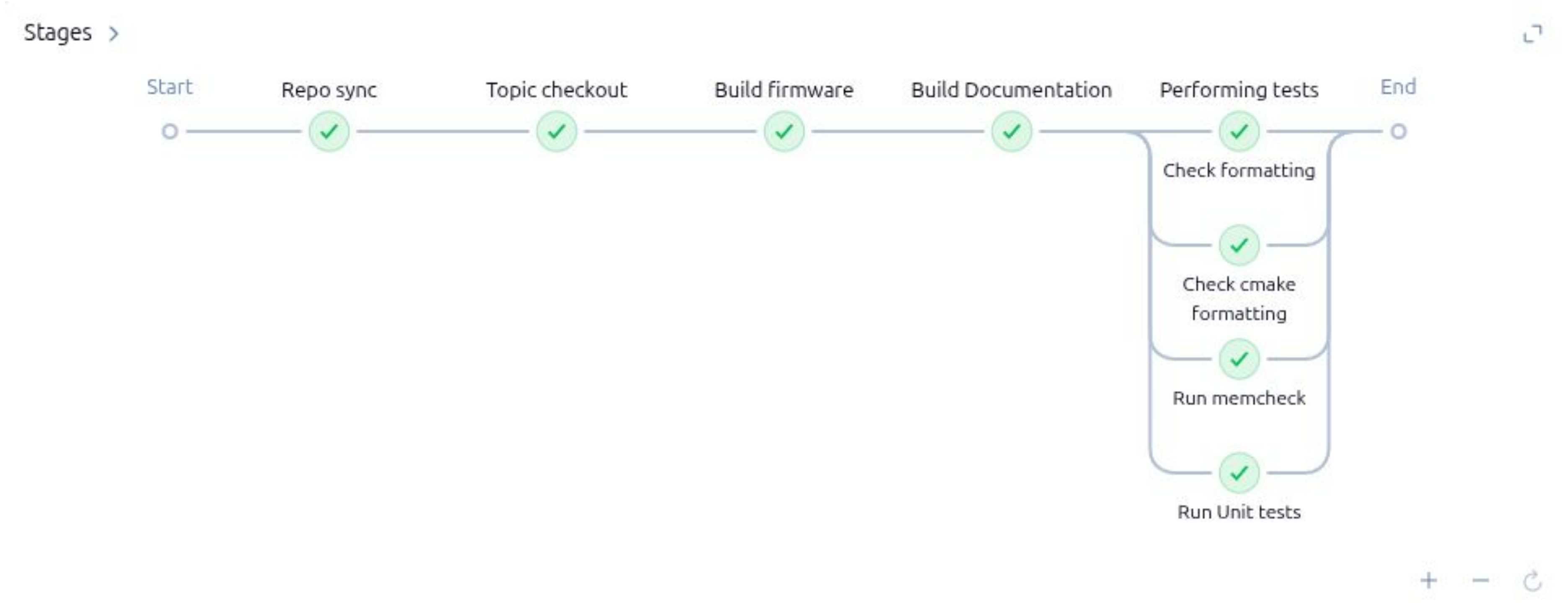


Automatic checks from Unit Test

- **Unit tests**
- **Code coverage**
  - Get a clearer picture of what parts of the code might still have bugs or vulnerabilities due to lack of testing
- **Memcheck**
  - Valgrind allocation/free path check
- **Documentation warnings**
  - We need to check all type of change in a project



Unit test pipeline

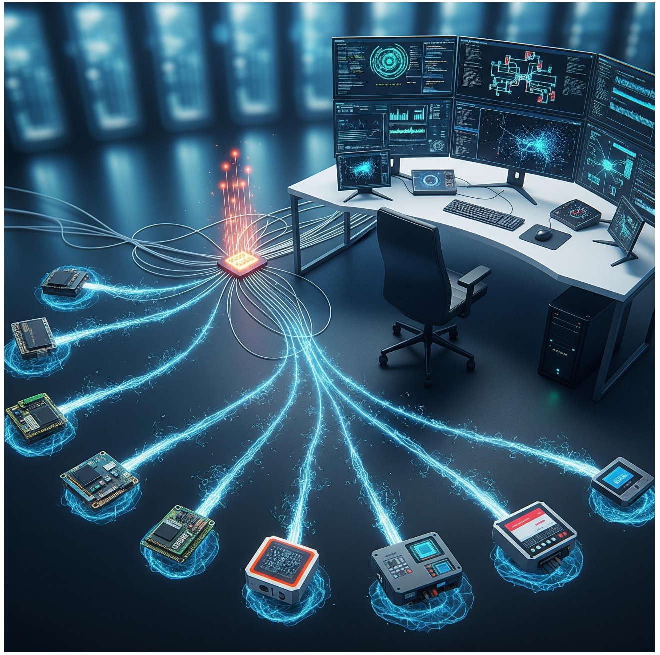


Testing pipeline example

- https://wiki.amarulasolutions.com/articles/jenkins/warning-ng-jenkins-valgrind.html
- https://wiki.amarulasolutions.com/articles/jenkins/warning-ng-jenkins-codechecker.html

What is Labgrid?

Labgrid is a powerful testing framework for embedded systems. It enables remote control and power management of devices, provides hardware access across diverse platforms, and automates lab environments to support continuous integration workflows.



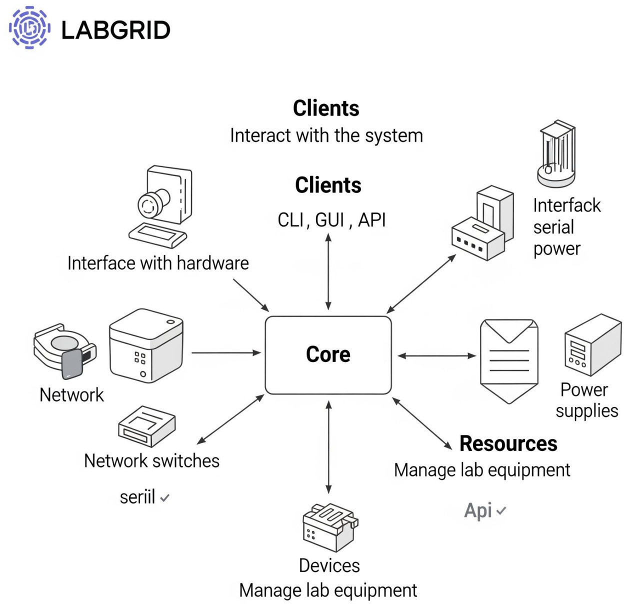
Labgrid Architecture

Core Components

Labgrid is a Python library for embedded systems testing and automation. It can run standalone (`labgrid-client`) or inside pytest.

Key concepts:

- **Targets:** the device under test (DUT).
- **Resources:** components of the target (e.g. serial port).
- **Places:** logical groups of resources (all for one device, multiple devices, or a subset like display testing).

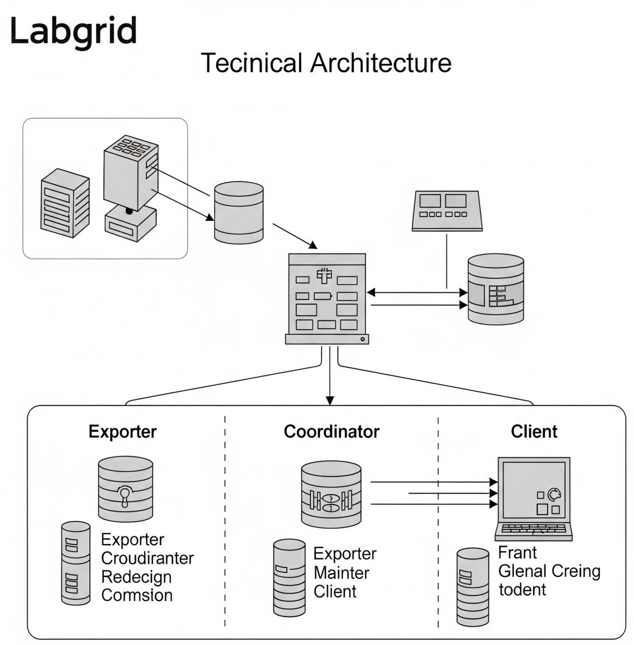


Labgrid Architecture

Core Components

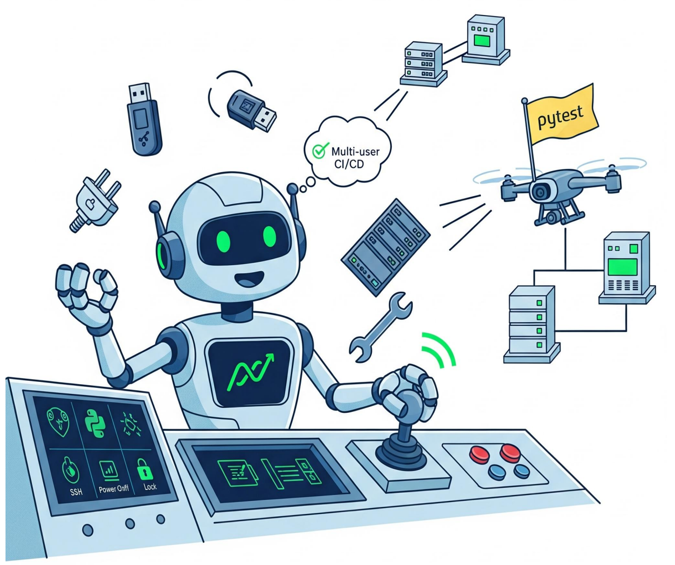
Labgrid operates through three components working together:

- **Exporter:** Connected directly to the DUT via interfaces (serial, USB, etc.). Declares and manages Resources in a local configuration file, making them available to the Coordinator.
- **Coordinator:** A gRPC server that manages multiple Exporters. Maintains a central list of all Resources and Places. Handles resource allocation by: Matching requested Places with available Resources. Locking those Resources to prevent conflicts (e.g. two CI jobs using the same serial port). Providing Clients with connection details or tokens. Once released, Resources become available again for other Clients.
- **Clients:** The user-facing component that runs tests. Uses its own configuration file (configuration.yaml) to define Targets and Drivers. Requests Places from the Coordinator, receives instantiated Drivers (e.g. SerialDriver, ShellDriver), and executes test logic. After tests complete, releases Places so the Coordinator can unlock Resources.



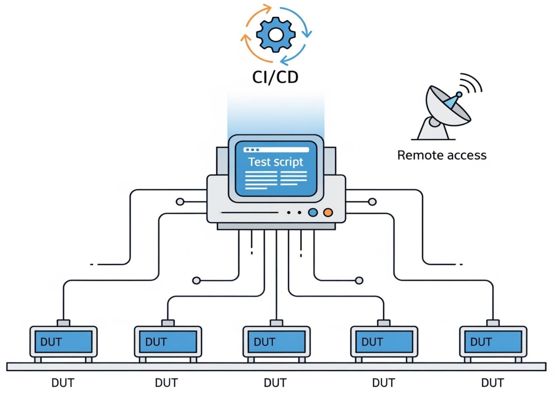
Key Features

- Remote control of devices (serial, SSH, USB)
- Power management (on/off, reset, cycling)
- Resource allocation & locking for safe multi-user CI pipelines
- Seamless integration with pytest for automated testing
- Support for multiple devices and shared lab environments



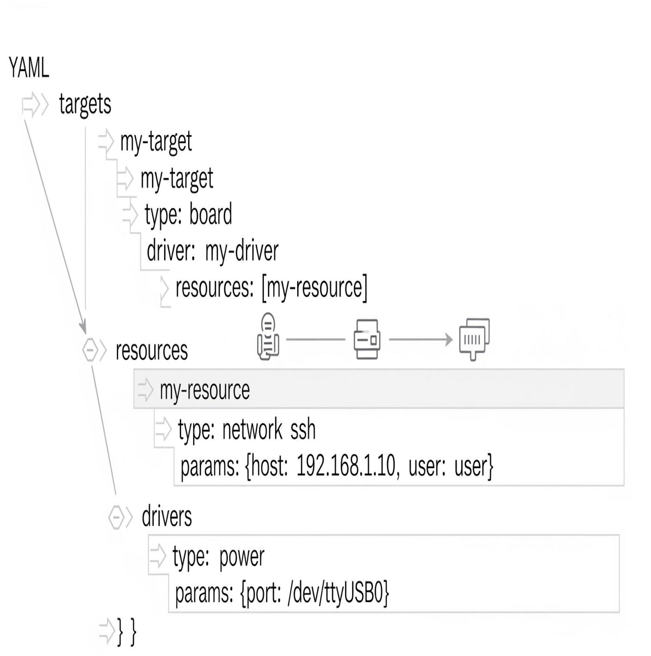
Use Cases

- Regression testing on embedded hardware.
- Automated lab environments for continuous integration (CI/CD).
- Hardware validation during development.
- Remote debugging and reproducible test setups.



YAML Configuration

In Labgrid, **YAML** is the declarative configuration file that defines targets, drivers, and resources. It tells Labgrid how to connect to and control devices under test, ensuring reproducibility and clarity across projects.



```
YAML
  targets
      my-target
        my-target
          type: board
          driver:  my-driver
            resources: [my-resource]
    resources
      my-resource
        type: network ssh
        params: {host: 192.168.1.10, user: user}
     drivers
        type:  power
        params: {port: /dev/ttyUSB0}
   } }
```

Tests Example

In order to run tests add a conftest.py file. The conftest.py is placed in the root of a pytest project. It defines fixtures that bind resources such as drivers or protocols to pytest. Tests can then use these fixtures automatically without any extra imports.

Example of conftest.py:

```
import pytest
from labgrid.target import Target
from labgrid.protocol import CommandProtocol

@pytest.fixture(scope="session")
def command(target: Target) -> CommandProtocol:
 """Provide a CommandProtocol for running DUT"""
   command= target.get_driver("ShellDriver")
   target.activate(command)
   return target.get_driver("CommandProtocol")
```

Tests Example

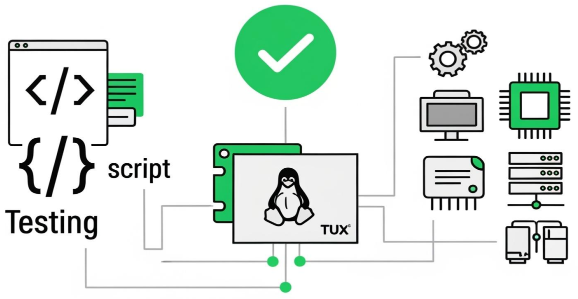
```
def test_os_release(command: CommandProtocol) -> None:
   """This function tests the contents of the system's
'/etc/os-release'.

   :param cmd_shell: An instance of a CommandProtocol
   :return: None
   """
   runner = Runner(command)
   rc, _, stderr = runner.cat("/etc/os-release")
   msg = f"The test failed. rc: {rc}, stderr: {stderr}"
   assert rc == 0, msg
```

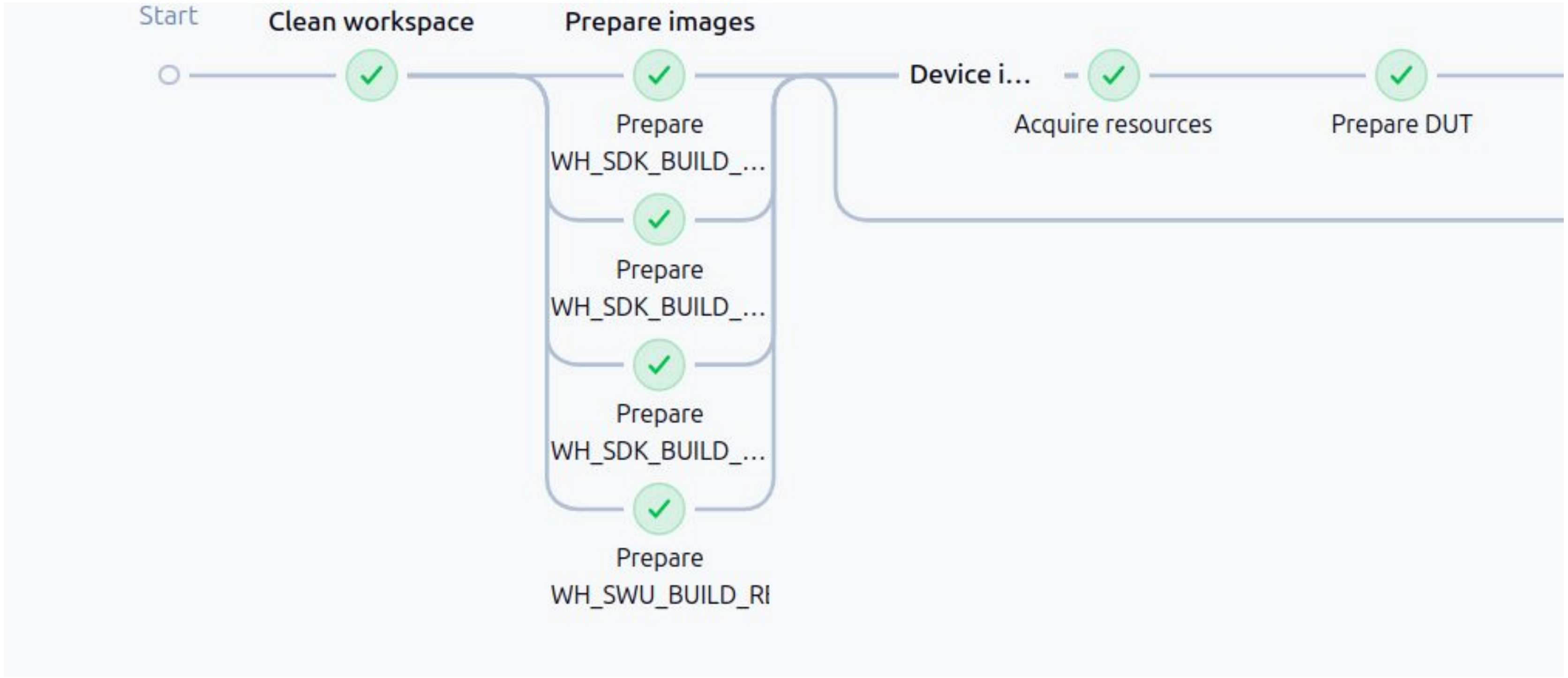
```
def test_os_release():
{
def test_os_release(): ...
}
```

Hardware testing benefits

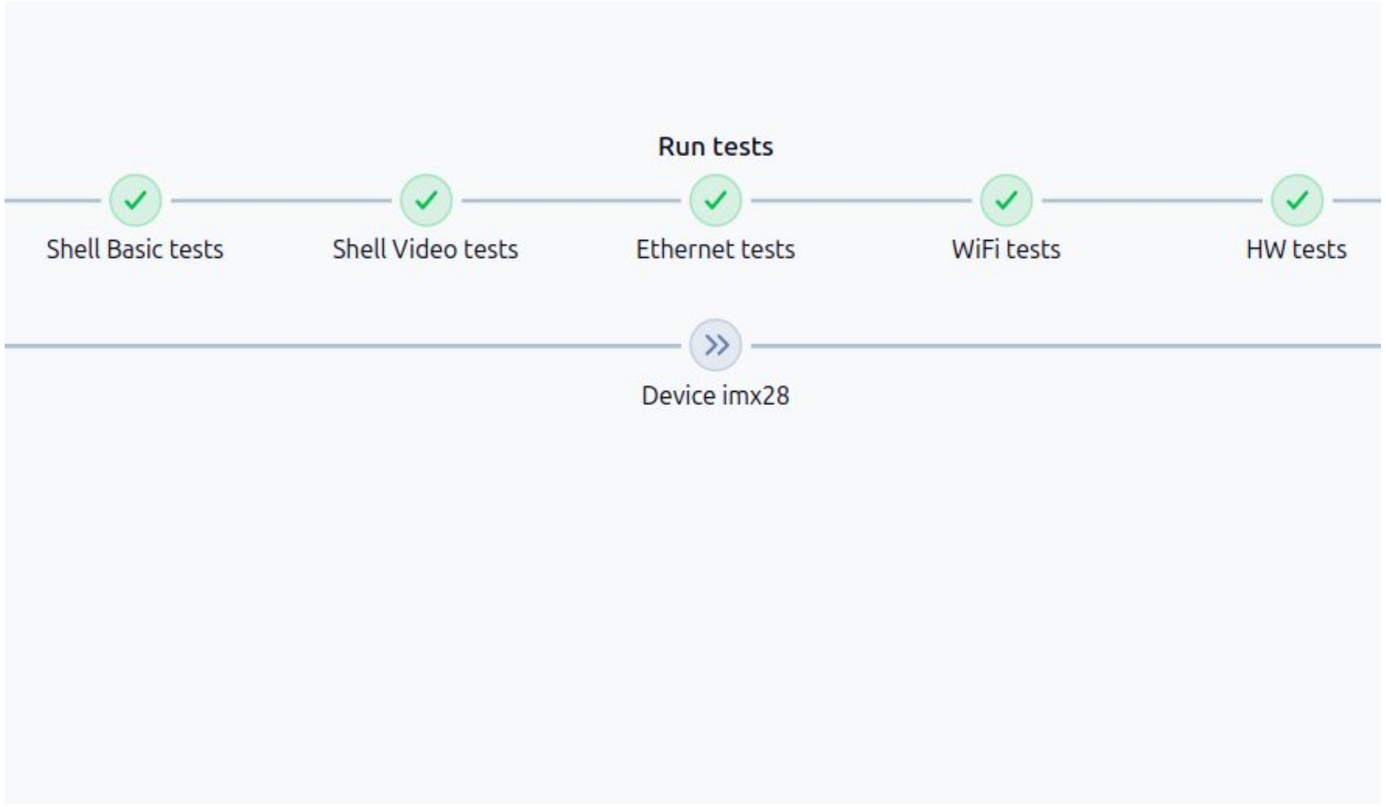
Labgrid makes testing embedded Linux boards easier and more reliable. It provides a unified way to control devices under test (DUT) through drivers and protocols. By using Labgrid with pytest, tests become reproducible, automated, and consistent. This reduces manual effort, speeds up debugging, and improves confidence in hardware and software integration.



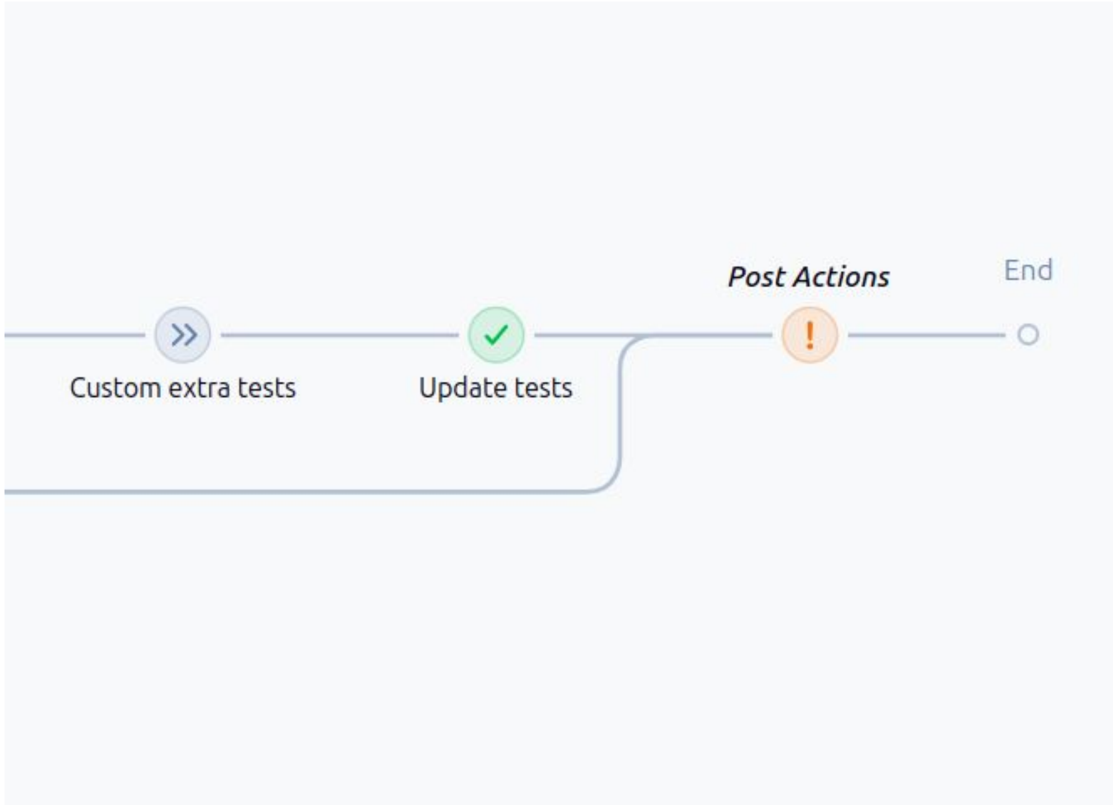
Hardware test pipeline using labgrid (1)



Hardware test pipeline using labgrid (2)



Hardware test pipeline using labgrid (3)



Labgrid pipeline jenkins example

https://wiki.amarulasolutions.com/articles/jenkins/how-validate-os-build-labgrid-testing.html

Building Failure (how deal with them)

Use a Jenkins plugin that integrates **Generative AI** (OpenAI, Gemini, Ollama) directly into your CI/CD pipeline to diagnose build failures.
The Problem it Solves: Developers often waste time digging through thousands of lines of verbose console logs to find a single "needle in a haystack" error.
Key Value Propositions:

- **Instant Context:** Translates cryptic stack traces in any configure language.
- **Actionable Advice:** Provides specific steps or code snippets to fix the identified issue.
- **Efficiency:** Reduces "Mean Time to Repair" (MTTR) by automating the initial debugging phase.

Seamless Workflow Integration

A dedicated "Explain Error" button appears on the build sidebar and next to console log entries for on-demand analysis.

Easily automated within a Jenkinsfile to provide summaries as soon as a build fails.

Configuration Essentials:

- LLM Choice: Supports cloud-based (GPT-4, Gemini) or private local models (Ollama) for data privacy.
- Smart Filtering: Uses regex to strip out "noise" (timestamps, progress bars) so the AI stays focused on the actual error.

Roadmap explain error plugin (my view)

- Integrate with Pipeline View Plugin
  https://plugins.jenkins.io/pipeline-graph-view/
- Deal with parallel stages build and multi-reason failures
  - Improve
    https://github.com/jenkinsci/explain-error-plugin/pull/69/changes/e5a9e95cef896f4e84edda3c00aea78c0e31c837
- Integrate it with CI/CD and notification using some tailored groovy library

Let’s see some usage example…

Useful links

https://community.jenkins.io/
https://wiki.amarulasolutions.com/ci/index.html
https://wiki.amarulasolutions.com/articles/jenkins/index.html
https://github.com/amarula/chatgpt-code-review-gerrit-plugin
https://wiki.amarulasolutions.com/opensource/chatgpt-gerrit.html
https://amarula.github.io/chatgpt-code-review-website/

Collaborating on open source projects is an exceptional way to refine your programming skills. There is immense value in creating software that benefits the wider community. If you are looking for a place to start, Jenkins has a vibrant community with numerous plugins that welcome new contributors.

Michael Trimarchi
michael@amarulasolutions.com
https://github.com/panicking

ITALY
Amarula Solutions SRL
*Via F. Cavallotti, 25D, 41012 Carpi *MO)*
+39-0422/435310