Test Automation Framework for Embedded Linux Testing

Sahana K S, Maanas M D, M Govinda Raju,
RV College of Engineering, Bengaluru, India
RV College of Engineering, Bengaluru, India
RV College of Engineering, Bengaluru, India

sahanaks.ec17@rvce.edu.in, *maanasmd.ec17@rvce.edu.in*,
govindarajum@rvce.edu.in

Abstract

Software testing is one of the most important steps before its official roll–out to the market. Testing of embedded software is a very complex, tedious and time–consuming process. Manual execution of the test cases can pose a huge challenge to the test engineers in terms of the time consumed to execute and compile the results of hundreds of test cases. This paper presents a solution to automate the entire process of build and install of Board Support Package (BSP) using Jenkins, along with the automation of the testing process using Linaro Automation and Validation Architecture (LAVA). The proposed design is validated in a BSP testing environment to show the ease of testing using the framework mentioned in this paper.

Keywords— Jenkins, LAVA, Software Testing, Embedded Systems, Test Automation Framework

I. Introduction

The field of embedded systems has grown exponentially over the last few years. With this steep rise, the complexity level of the software associated with these systems has also increased greatly. Although there has been a significant improvement in the tools available to test this software, the layered architecture of the embedded software poses a big challenge during testing. Though testing is the last step in the product development cycle, it is a very crucial one.

Manual testing, of all the possible scenarios of the software of an embedded system, is a very time–consuming process. The delay caused, increases the time gap between development and deployment of the software. Manual testing also introduces human error which can compromise the quality of the product. Quality Assurance has long been in the background in the product development life cycle and in this paper, we aim to bring it to the foreground.

The main idea of this paper is to provide a test automation framework using open–source tools to overcome the limitations of manual testing. The presented methodologies can be used for testing of embedded BSP or any general software application.

II. Related Works

The following provides a brief idea about the recent research on our topic.

In Paper [1], the authors have highlighted the advantages of using a CICD system for big projects. The proposed solution makes use of Jenkins and the Ansible tool for the automation process. A pipeline architecture is presented for the execution of the process using Jenkins. The deployment is carried out using Ansible, which later is used for the testing process. The CICD architecture proposed in this paper helps to reduce the time gap between development and deployment.

Paper [2] proposes a test framework capable of testing heterogeneous embedded systems. The main idea of the paper is to classify the tests into separate hardware and software categories. Various characteristics considered include embedded characteristics, interaction, development practice and timeliness. The proposed methodology also includes the development of a discovery interface. The methodology has been implemented on a hardware setup and is found to be effective.

Paper [3] focuses on the limitations of embedded software testing. The authors suggest that the increased complexity of embedded software testing is due to the layered architecture. The tracing of a bug occurring at some layer has become hard to pinpoint. The paper has also suggested a few points which will be helpful for developers and testers.

In paper [4], the authors develop a tool to automate model–based embedded testing. The developed tool is called ESWST, which is a plugin for Eclipse. The main purpose of this tool is to automate the construction of the proposed model. The model proposed in the paper abstracts various artifacts such as hardware interface information, call graph, control flow graph, and data flow information. Using these artifacts, the embedded software is analysed and understood. The model–based approach is a very effective method. Larger code coverage for test cases can be achieved using the method proposed in this paper. The paper intends to expand the test model to cover more testing artifacts, enhancing its efficacy and coverage.

Paper [5] mainly illustrates the evolution of Jenkins from a Continuous Integration to Continuous Delivery tool. The new design proposed automates the delivery process along with the build. The paper also highlights a few challenges with Jenkins like the lack of a method to track the built environment. The paper gives the drawback of the traditional waterfall model and sheds light on how those drawbacks are overcome in the new agile technique.

In paper [6], Computer–Aided Specification and Testing (CAST) approach is proposed to automate the process of testing embedded systems. The flaws of manual testing have motivated the authors to switch to automation. The proposed system consists of three main parts: test cases in a domain–specific language, an execution engine to run all the test cases, and an interface to connect the interface to an embedded system. The solution proposed for the above problem is CAST, which is a framework for automatic test execution in real–time embedded systems. The domain–specific language used is TESLA which further supports test data generation. Through the proposed solution, the testing time has reduced to 15 to 30 minutes from 3 to 5 days. Therefore, it establishes the advantage of using automation over manual testing.

III. Proposed Solution

Jenkins and Linaro Automation and Validation Architecture (LAVA) are the main frameworks used in this paper. Jenkins helps in automating the install/build stages for the images. LAVA provides a platform to automate the deployment and testing of targets on various supported features.

A. Jenkins

Jenkins is a free and open–source Java–based automation tool that aids in the automation of portions of software development such as building, testing, and deploying, allowing for continuous integration and delivery. Jenkins is a robust tool that can handle both parallel and distributed builds. Continuous Integration (CI) is a development methodology in which developers often integrate code into a shared repository, with each integration being validated by an automated build and automated tests. Continuous Delivery (CD) is a process in which code modifications are built, tested, and prepared for production release. The figure 1 shows the CICD.

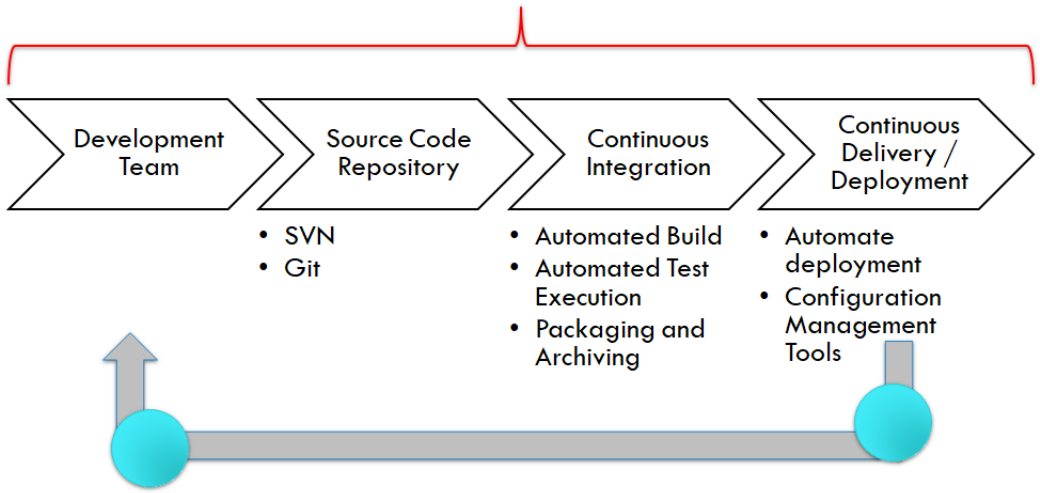

Fig. 1: Continuous Integration and Continuous Delivery [7]

1) Jenkins Pipeline: Jenkins Pipeline is a set of plugins that help you set up and use continuous delivery pipelines in Jenkins. The pipeline extends Jenkins with a robust collection of automation features, enabling use cases ranging from simple continuous integration to complex continuous delivery pipelines.

Blocks in a Jenkins Pipeline:

- Pipeline: A Pipeline is a CD pipeline model that is defined by the user. A Pipeline's code specifies the whole build process, which often includes phases for developing, testing, and deploying an application.
- Node: A Jenkins node is a machine that can run a Pipeline and is part of the Jenkins environment.
- Agent: An agent is a Pipeline–specific Declarative syntax using which Jenkins allocates an executor (on a node) and workspace for the entire Pipeline.
- Stage: Many plugins employ stage blocks to visualise or present Jenkins Pipeline status/progress. A stage block is a conceptually different subset of actions that are carried out along the Pipeline (e.g. "Build", "Test" and "Deploy" stages).
- Steps: Actions that will be executed in a certain stage are specified.

The figure 2 shows the different blocks in a Jenkins pipeline.

```
Jenkinsfile (Declarative Pipeline)
    pipeline {
        agent any
        stages {
            stage('Build') {
                steps {
                    //
                }
            }
            stage('Test') {
                steps {
                    //
                }
            }
            stage('Deploy') {
                steps {
                    //
                }
            }
        }
    }
```

Fig. 2: Jenkins Pipeline [8]

B. Linaro Automation and Validation Architecture (LAVA)

LAVA is a technique for deploying operating systems onto real and virtual hardware in order to do testing. Simple boot, bootloader, and system–level testing are all feasible, while some system tests may necessitate the use of extra hardware. The data can be exported for further examination and the results can be monitored over time. LAVA is designed for validation during development. LAVA is helpful for allowing developers to execute customised tests

on a variety of devices, some of which might be difficult to acquire or integrate.

A master and a worker are the two main components of a LAVA instance. The master and worker components can be run on the same system for the simplest configuration, but a larger instance can be built to handle numerous workers controlling a higher number of associated devices. The figure 3 shows the architecture of LAVA.

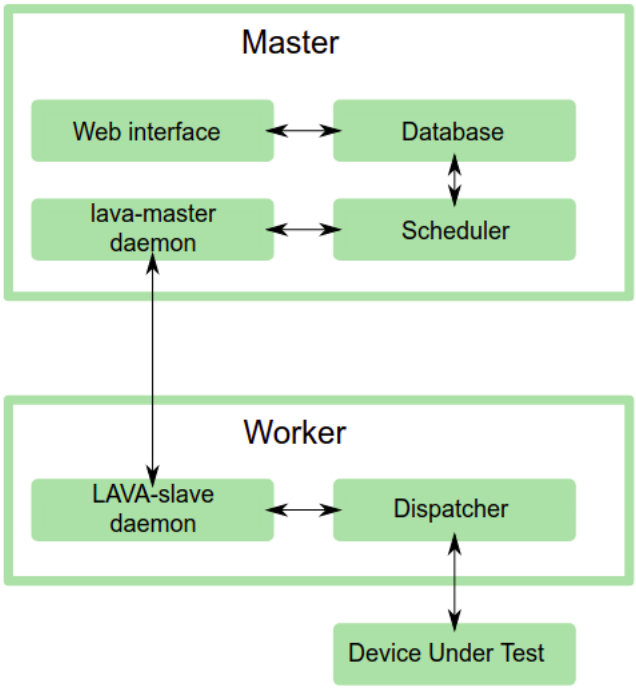

Fig. 3: LAVA architecture [9]

1) Elements of the master:

- Web interface – The Django web framework is used in conjunction with the Apache webserver and the WSGI application server to build a web interface.
- Database – The PostgreSQL is used locally on the master, with no provision for external access.
- Scheduler – The part responsible for jobs run. The database is scanned for queued test jobs and available test devices regularly, jobs are launched as soon as the resources are available.
- Lava–master daemon – The part responsible for communication with the worker(s). It uses ZMQ for communication.
- Lava–logs daemon – The part is responsible for the aggregation of the logs arriving from the dispatchers.

2) Elements of the worker:

- Lava–slave daemon – The part responsible for the reception of control messages from the master and uses ZMQ to deliver the logs and results back to it.

- Dispatcher – According to the task submission and device parameters given by the master, this controls all operations on the device under test.
- Device Under Test (DUT) – The target embedded device.

C. Integration of Jenkins and LAVA

Figure 4 shows the complete implementation of the automated test framework.

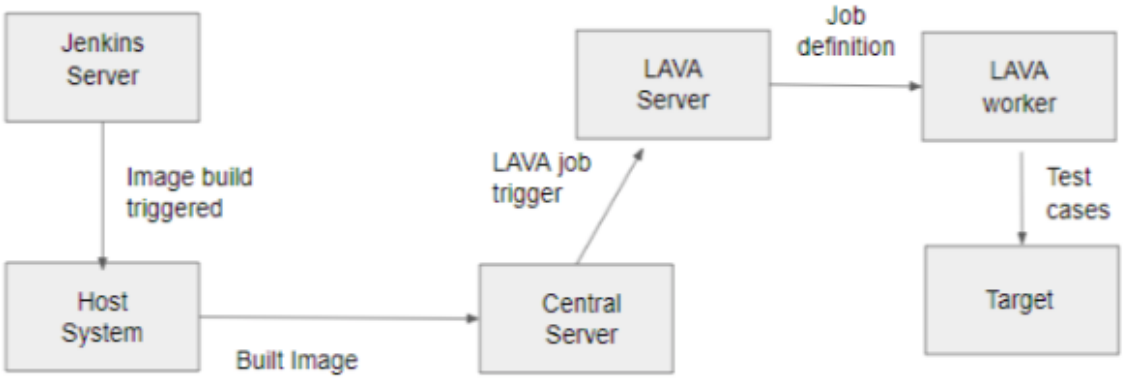

Fig. 4: Complete Implementation

- The Jenkins server triggers a build on a Host system or a node. The host system can be an Ubuntu or a Debian machine.
- The successfully built images get copied to a centralized server.
- A LAVA job is triggered wherein a LAVA server sends a job definition containing the instructions on deploying the built image from the centralized server onto a target.
- The LAVA server then makes the LAVA worker to deploy the job on the required DUT.
- The results are published by the LAVA worker to the LAVA server once the test is completed.
- The failed test cases are reviewed and corrected. The testing is carried out again until the required specifications are met.

IV. Results

The test framework developed is tested for around 32 build/install functions using Jenkins and around 150 test cases using LAVA. The BSP images built using Jenkins are deployed on various NXP, IPC, Xilinx, and Beaglebone targets, and are tested for different features manually and through automation. Using the proposed framework, the average amount of time spent by the test engineer reduced from 2 days to roughly 30 minutes and thereby freeing up the resources for other work.

V. Conclusion

With the advancement in technology, we aim to create an efficient and reliable alternative to the manual testing of embedded devices. With the help of CICD tools such as Jenkins and LAVA, we were able to create a coherent framework for automated testing. Jenkins is a sophisticated application that allows projects to be continuously integrated and delivered, regardless of the underlying platform. It is used to automate the process of building images for the BSPs. LAVA is used to deploy operating systems onto physical and virtual hardware for testing. This framework improves the efficiency of testing and makes it less prone to human error.

VI. Future work

The test automation framework is an essential requirement in the field of embedded testing where the majority of the testing is currently carried out manually. In the future, this framework can be expanded to test different software applications. There is scope for optimisation in the usage of Jenkins and LAVA. This framework can be expanded to include automatic test generation which would further reduce the time required for testing.

References

[1] S. Mysari and V. Bejgam, "Continuous integration and continuous deployment pipeline automation using jenkins ansible," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 2020, pp. 1–4.

[2] S. P. Karmore and A. R. Mahajan, "Universal methodology for embedded system testing," in *2013 8th International Conference on Computer Science Education*, 2013, pp. 567–572.

[3] A. Sung, S. Kim, Y. Kim, Y. Jang, and J. Kim, "Test automation and its limitations: A case study," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 1208–1209.

[4] C.-H. Liu, S.-L. Chen, and T.-C. Huang, "A model-based testing tool for embedded software," in *2012 Sixth International Conference on Genetic and Evolutionary Computing*, 2012, pp. 180–183.

[5] V. Armenise, "Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery," in *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, 2015, pp. 24–27.

[6] M. Wahler, E. Ferranti, R. Steiger, R. Jain, and K. Nagy, "Cast: Automating software tests for embedded systems," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 457–466.

[7] "Continuous Integration Continuous Delivery." [Online]. Available: https://subscription.packtpub.com/book/virtualization_and_cloud/9781788471060/1/01lvl1sec13/overview-of-the-ci-cd-pipeline

[8] "Jenkins Pipeline." [Online]. Available: https://www.jenkins.io/doc/book/pipeline/

[9] "Linaro Automation and Validation Architecture (LAVA) workflow." [Online]. Available: https://docs.lavasoftware.org/lava/index.html