TEST AUTOMATION IN CONTINUOUS INTEGRATION FOR HARDWARE VALIDATION

Pedro Dias Faria

Dissertation under supervision of *Prof. Rui Filipe Lima Maranhão de Abreu*
at *Synopsys Portugal Lda*

1. Motivation

The hardware validation process is directly related with verifying if the different clients configurations will be fulfilled. As such, the validation process can become a subjective process, since it involves assessing how the behavior of the hardware will operate in the most diverse applications and conditions. The process typically consists of activities which include system modeling, prototyping and evaluation by the user[1][2].

With this dissertation, we helped to build up a continuous integration environment for hardware validation by developing a Jenkins plugin in form of a dashboard in order to help Synopsys R&D teams on the hardware prototyping process.

2. Goals

The subjectivity of validation process in any kind of software development requires the use of some type of structure and categorization of data. This can help an easier access in troubleshooting flaws in the development.

With our solution, the productivity in development teams should increase by having this information displayed in a streamlined, precise and straightforward way.

In order to validate this, the system had to be implemented in a real project and feedback collected by the team to understand if the value added is noticeable.

The goals for the dissertation are:

- Define an automatic test management structure for Hardware validation;
- Define techniques to label and manage the Hardware validation test results;
- Development of an web application to support the automatic test system;
- Test the web application and validate its usefulness.

3. Work Description

Since Synopsys had already setup the CI server in Jenkins, it was chosen to develop the Plugin for it. We also take the avantage of Jenkins being very customizable, with hundreds of extension points and plugins available to support our needs [3].

3.1. Use Cases

Taking into consideration the R&D project organization of Jenkins, it was discussed with the collaborating IPK team that the best option was to develop the Dashboard as a View plugin, implementing the class ViewGroup, in order to aggregate different views inside it.

In the Use Case diagram in the figure 1 are pictured the actions a user can perform to display our Dashboard in the Jenkins environment.

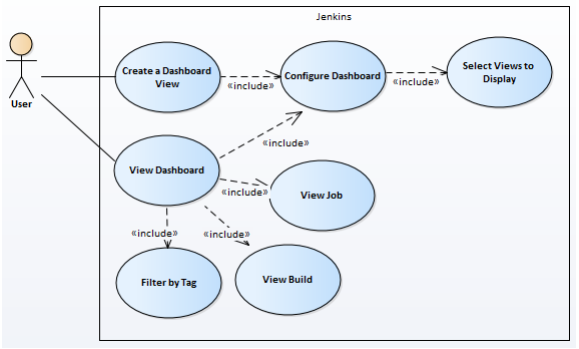

Fig. 1 – Dashboard plugin Use Case diagram

The plugin is designed to be as simple as creating a new View inside the tool, selecting which other Views are requested to display. Afterwards, the user can freely change its configuration, namely add, remove or change which views will be displayed.

3.2. Jenkins Plugin - Filtered Dashboard View

3.2.1. Mission Control Plugin

Accounting that we want to create a Dashboard, we came across the Mission Control Plugin [4], that has practically all the base to work with, a full dashboard view that features:

- Job status display;
- Build history and build queue;

- Extends a View Object, which can be added to the main dashboard.

Despite being a starting point it needed to be trimmed and adapted to our requirements.

3.2.2. Metadata Plugin

Considering that one of the requirements is filtering the data displayed with the information of each test configuration and since Jenkins has the possibility to store information on each of its CI items in XML format, then it could be stored some type of metadata as well.

To facilitate this, we used the extension point that Metadata Plugin [5] provides, in conjunction with the Jenkins API, to apply relevant filters on each job for an easier parsing of these configurations information.

This Metadata can be applied to Jobs simply through the customization menu, which will apply it to every future Build after this point.

Or alternatively, automate the process simply by calling a post-build CLI command that is provided from the same plugin.

3.3. Server Side Implementation Design

Understanding how the back end in Java interacts with Jenkins and how projects were organized in Views, we defined a *Top-down* strategy to obtain and parse the information of our items of interest. This means we had to process View by View as individual projects to relate their child items in our Dashboard as seen in the figure 2.

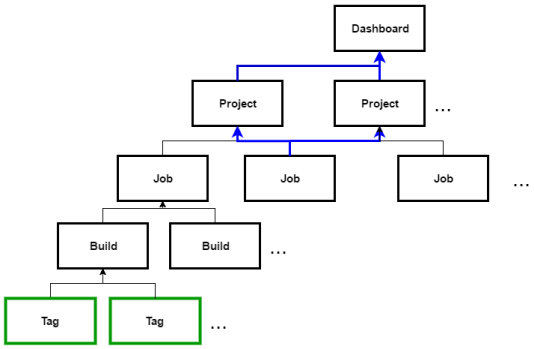

Fig. 2 – *Top-down* approach diagram

Finally, the addition of Tags to filter the information is the key feature of our Dashboard. This concept helps to maintain track of each configuration, with the minimum requisite of adding, or modifying, these labels to Jobs in the building process.

This design permits a totally scalable structure, since it has no limitations on the number of Views associated to the Dashboard, nor its Jobs and Builds. It can also be extendable to put more information in each of its custom classes for additional metrics and indicators in the future if desired. All this information only depends on the Jenkins API and what it supports.

3.4. Front-End Implementation Design

For information organization structure, in the front-end display of the Dashboard, we based on the same scripts of the Mission Control Plugin, described in section 3.2.1, along with several common Web Application libraries and frameworks such as Javascript, jQuery and Bootstrap, all processed in Apache Jelly, a Java and XML based scripting and processing engine that allows Jenkins its UI to be extended by plugins.

Since the data exported to the API is already well structured, the presentation process was completed with ease, only requiring a way to exhibit the view of a project with an intuitive display.

To accomplish the aforementioned, was decided to display every Job inside the project as a column in a table, being each row cell its Builds organized in a descending chronological order. Each cell has its Tags associated for a quick reference of the configurations utilized in the respective Build.

4. Conclusions

In the end of this project we were able to develop successfully the intended Dashboard Plugin [6].

Although narrowed down some features expected in the beginning of the project, such as displaying metrics about the hardware validation process due to time constraints between the end of development and testing of the Plugin, we believe it will greatly help development teams to keep track of their projects in a simpler and streamlined way.

References

[1] Troy Scott. Modern fpga-based prototyping systems accelerate the transition from stand-alone ip block validation to integrated systems. `https://www.synopsys.com/designware-ip/newsletters/technical-bulletin/modern-prototyping-systems.html`.

[2] Stephan Puri-Jobi. Test Automation for NFC ICs using Jenkins and NUnit. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–4. IEEE, apr 2015.

[3] JenkinsCI. Jenkins continuous integration tool. `https://jenkins.io/`, June 2016.

[4] Andrey Shevtsov. Mission control plugin - jenkins. `https://wiki.jenkins-ci.org/display/JENKINS/Mission+Control+Plugin`, 2015.

[5] Robert Sandell and Tomas Westling. Metadata plugin - jenkins. `https://wiki.jenkins-ci.org/display/JENKINS/Metadata+Plugin`, 2012.

[6] Pedro Faria. Filtered views dashboard plugin - jenkins. `https://wiki.jenkins-ci.org/display/JENKINS/Filtered+Views+Dashboard+Plugin`, 2017.