ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA164

AUTOMATED SOFTWARE TESTING FOR CONTROL AND MONITORING A RADIO TELESCOPE

B. Xaia, T. Gatsi , O J. Mokone†, SKA SA, Cape Town, South Africa

Abstract

The 64-dish MeerKAT radio telescope, under construction in South Africa, will become the largest and most sensitive radio telescope in the Southern Hemisphere until integrated with the Square Kilometre Array (SKA). Software testing is an integral part of software development that is aimed at evaluating software quality; verifying and validating that the given requirements are met. This poster will present the approach, techniques and tools used to automate the testing of the software that controls and monitors the telescope. Jenkins continuous integration system is the server used to run the automated tests together with Git and Docker as the supporting tools to the process. In addition to the aforementioned tools we also use an Automated Qualification Framework (AQF) which is an in-house developed software that automates as much as possible of the functional testing of the Control and Monitoring (CAM) software. The AQF is invoked from Jenkins by launching a fully simulated CAM system and executing the Integrated CAM Tests against this simulated system as CAM Regression Testing. The advantages and limitations of the automated testing will be elaborated in the paper in detail.

INTRODUCTION

Nowadays any software functionality is required to be delivered faster and with minimum cost while maintaining the quality expected. This applies to any software and also to process automation applications. These critical applications need to be extensively tested to validate the requirements and ensure a smooth operation of the targetted instrument. It is generally accepted to divide tests according to their level of specificity into: unit testing, where a specific section of code is tested separately, and integration testing, where all individual units are put together to be checked globally. The paper describes the software environment where testing procedures, techniques and the test methods employed focusing basically in automated tests mechanisms as they are applied within the CAM team. Among the tools and techniques used in CAM for automated testing include Jenkins, Github, slack, vitech core, *Automated Qualification Testing Framework and Docker. Finally the paper summarises the results and analysis of the positives and drawbacks of applying these automated testing techniques as compared to manual testing.

CAM DEVELOPMENT PLAN

MeerKAT CAM software is developed through agile iterative implementation cycles with simple basic solutions being put in place first, which are then enhanced during subsequent development cycles, with close input from the system engineers and commissioners as the understanding of requirements matures [1]. This is managed through the MeerKAT CAM project plan.

Various integration levels will allow verification against a sequence of progressively more complete system element configurations, including Unit testing, Component testing, Integrated CAM testing, CAM Qualification Testing, Lab Integration Testing and CAM Acceptance Testing.

The CAM qualification stage for each cycle/timescale will include unit testing, Component testing, a continuous build server for Regression testing, Integrated CAM Testing against CAM verification requirements producing an automated Qualification Test Plan (QTP) and Qualification Test Report (QTR).

Qualification testing is performed to functionally prove the CAM design and implementation against the CAM requirements. Qualification testing of CAM software is performed on representative CAM hardware in the lab in Cape Town with all external subsystem/devices simulated. To ensure timely integration, the suppliers of subsystems that the CAM interfaces to provide Karoo Array Telescope Communication Protocol (KATCP) [2] simulators that represent their subsystem's external CAM interface. In cases where the suppliers do not provide a KATCP simulator for the subsystem, the CAM team will develop such a subsystem simulator. The CAM application software is released for deployment to site after successful CAM qualification testing.

Acceptance testing is performed to accept the deployed CAM subsystem on site. It will reuse a predefined set of the CAM qualification tests that are non-intrusive and benign and can therefore be executed on the real hardware and on site. Figure 1 show the integrations, qualification and acceptance testing in the cycle forms.

† ofaletse@ska.ac.za

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA164

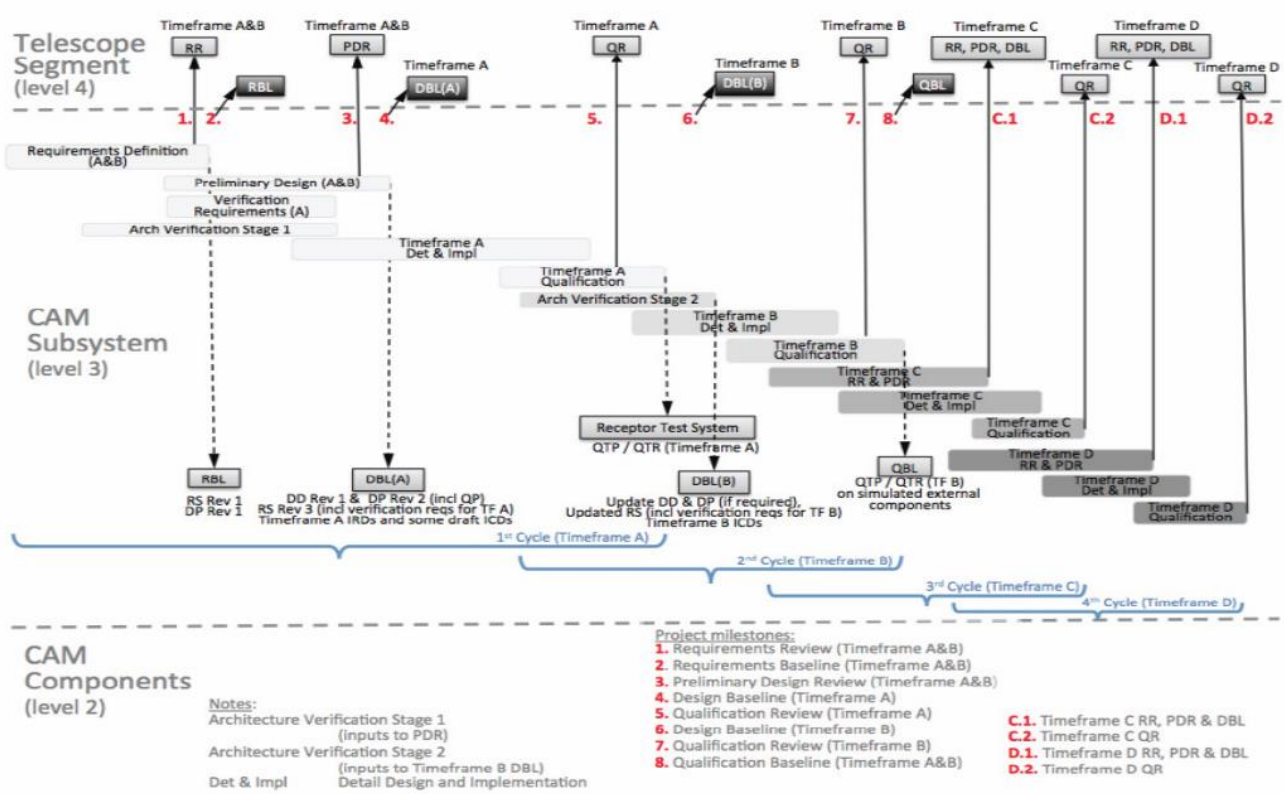

Figure 1: CAM development plan showing incremental software development and testing.

CORE MODEL

CAM requirements are captured in the MeerKAT project in the Vitech CORE systems engineering modelling database after the initial MeerKAT CAM Requirements Review (RR).

CAM verification requirements are captured in CORE and linked to requirements. Requirements for each cycle are then allocated to CAM functions and the CAM traceability matrix included in the CAM Requirement Specification (RS) [3] are generated from CORE.

Requirements for each cycle are allocated to CAM components in a traceability matrix for the CAM design at each phase.

TESTING ENVIRONMENT

It is possible to run a configuration including only simulated KATCP devices [2], or any combination of real and simulated devices combined. This allows full software development, unit testing and integration testing, and CAM subsystem qualification testing without dependency on the availability of hardware.

Although the full KATCP interface for each device is implemented in the simulators, the actual functionality of all the hardware components are not fully implemented; each simulator implements behaviour to the level required by CAM integration testing. However, antenna pointing and modes are simulated with realistic timing, and a representative simulation of the data output of the correlator will be implemented.

While the CAM team is responsible for developing most of the simulators, some of these device simulators are contractually delivered by the subsystem contractor to ensure that, given their knowledge of the device, the behaviour of the device is reflected with sufficient accuracy by the device simulator. Having a fully simulated system available is critical to automated testing.

UNIT TESTS

Unit tests validate the smallest components of the system, ensuring they handle known input and output correctly. Unit tests test individual classes in an application to verify that they work under expected boundary and negative cases.

There is a common myth among developers, that of being overscheduled and therefore one has no time for unit test with the hope that integrated tests will manage to catch one's bugs. That myth leads to the below vicious cycle (Figure 2) where a developer will postpone unit testing but end up with a less stable code with more bugs.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA164

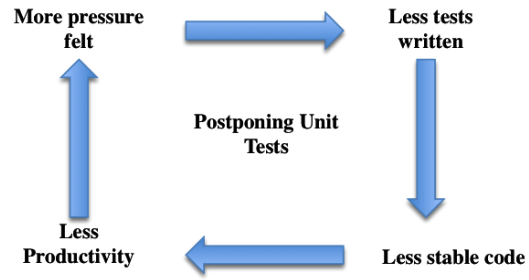

Figure 2: Vicious cycle of postponing unit testing.

Each CAM component is sufficiently covered by Unit Tests to ensure the units within the component are ready for component testing. Unit Testing is actually considered an output of test driven development. CAM has a continuous build server that executes all unit tests on standalone components/packages on a continuous basis.

Figure 3 gives a workflow of the software from when the developer pushes a commit to GitHub where GitHub uses a webhook to notify Jenkins of the update [4]. Jenkins then pulls the GitHub repository, including the Dockerfile describing the image, as well as the application and test code. Jenkins builds a Docker image on the Jenkins slave node and instantiates the Docker container on the slave node, and executes the appropriate tests. After tests have been run, a report is sent to Jenkins with test results and consecutively notifies the developer via Slack.

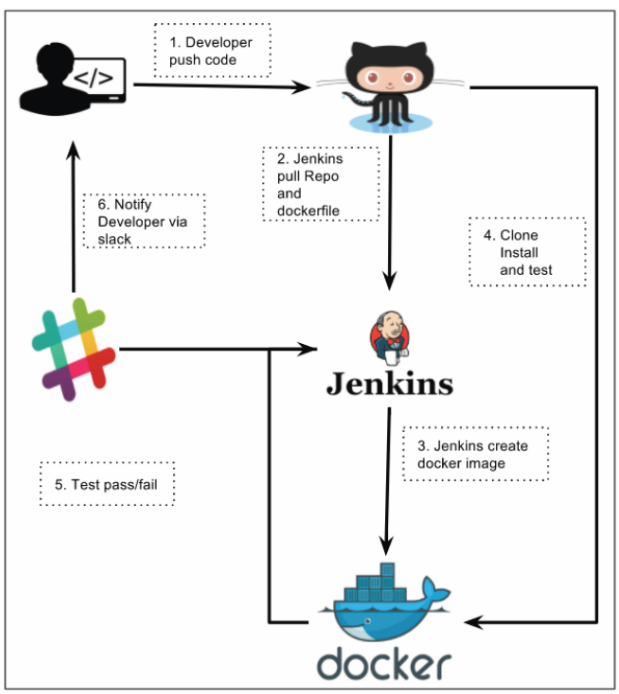

Figure 3: Unit testing workflow.

The CAM team utilises Docker to unify tests build and test environments across machines, and to provide an efficient mechanism for deploying applications. Integrating Docker into the Continuous Integration pipeline [5] has helped the CAM team to reduce job time, increase the volume of jobs run, enable flexibility in language stacks and improve overall infrastructure utilization.

AUTOMATED QUALIFICATION FRAMEWORK (AQF)

The AQF is a developed software item (a nosetests plugin called nosekatreport) supported by decorators in the Integrated CAM Tests that provide a framework for automated testing of the CAM Software and generating test procedure and test report documentation.

This software item is a test equipment, used to test the MeerKAT CAM, and is not part of the mission performing equipment of either MeerKAT. The intention of the qualification framework is to automate as much as possible of the functional testing of the CAM software, such that the largest portion of the Qualification Test Report (QTR) for each timeframe is automatically generated by the AQF when executing the Integrated CAM Tests. This is achieved by developing a set of integrated CAM tests with each test documented in-line so that the AQF generates the appropriate qualification documents from the tests. This generated document i.e. AQF specification record specifies the requirements for each Integrated CAM Test to support the AQF to extract relevant information from each of the tests. The documentation requirements will be for example, tags/decorators against each test to identify the requirements/verification requirements it implements, docstring requirements to describe the test, the format to identify/specify each test step, the type of test it is, etc.

In addition to the AQF being used for CAM qualification testing in the lab, the intention is also to use it for integrated CAM regression testing. In this context the AQF will be invoked from a build server (Jenkins) by launching a fully simulated CAM system once a week and executing the integrated CAM tests daily (overnight) against this simulated system as CAM regression testing.

Integration tests exercise an entire subsystem and ensure that a set of components play nicely together.

INTEGRATION TESTS

As mentioned in the AQF section the automated integrated CAM Tests are performed daily running against a fully simulated system to cover CAM functionality across multiple components and exercise the full CAM subsystem in a “true-to-life” framework. During the acceptance testing of a specific CAM component for integration, all requirements allocated to that component are covered by integrated tests. Verification requirements that require an integration tests are derived from system requirements in order to develop the automated integration tests. Each automated integrated CAM test is logged against the set of CAM verification requirements it implements. Of importance to note here is the fact that these tests are only run on the master branch which is the branch where development is happening. When all the integration tests pass we get a

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA164

stable branch that is automatically created and this branch can be used to create release branch which is tagged with a specific software version number during software deployment for production. Failing tests will require developers to fix the bug and the solution is merged to the master branch. The next automatically triggered tests will show the results of the fixed tests. These tests are invoked automatically at a time configured on the Jenkins server usually at midnight to verify and validate the whole CAM application software functionality. The Integrated CAM Tests are developed as nose tests that are invoked and decorated for the AQF. Once the Integrated Tests are run, a report with the results is produced with all the verification requirements covered. This test report forms the bulk of the QTR.

Figure 4 depicts the Jenkins platform displaying the results of a previously run CAM set of integrated tests showing the last successful and last failed runs as well as the duration of the tests. Within this platform one can navigate to tests results to view how individual tests ran and the failures which is then useful for fault finding and bug fixing.

CONCLUSION

Most of the automated tests precisely perform the similar operations every time they run; hence human errors are very much limited and almost eliminated. Automation streamlines software processes by following the same steps for a given test case to reproduce a defect. While automation tools can be expensive in the short-term, they save you money in the long-term. They not only do more than a human can in a given amount of time, they also find defects quicker. This allows the team to react more quickly, saving you both precious time and money. There is also a wider test coverage when running tests automatically as a lot of tests can be bundled together on one platform to execute all at once and the results at one glance on the Jenkins dashboard. Also on the Jenkins build server every developer can sign into the Jenkins testing system and see the results at any point in time. This allows for greater team collaboration and a better final product.

While the initial setup of test cases may take a while, once you've automated your tests, you're good to go. You won't have to continuously fill out the same information or remember to run certain tests. Everything is done for you automatically. Filling out the same forms time after time can be frustrating, and not to mention boring. Test automation solves this problem.

The process of setting up automated test cases takes coding and thought, which keeps your best technical minds involved and committed to the process. The QTP and QTR generation functionality of the AQF framework is a very useful component and output of the whole automated testing process. It allows for the analysis, investigation and interpretation of the test results. Figure 5 shows the QTR displaying the test results. While the automation process cuts down on the time it takes to test everything by hand, automated testing is still a time intensive process. A considerable amount of time goes into developing the automated tests and letting them run. While automated tests will detect most bugs in your system, there are limitations that involve visual considerations. Changes in these aspects can only be detected by manual testing, which means that not all testing can be done with automatic tools.

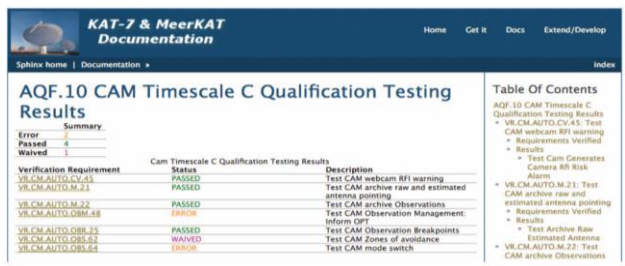

Figure 5: QTR report produced by CORE with test results.

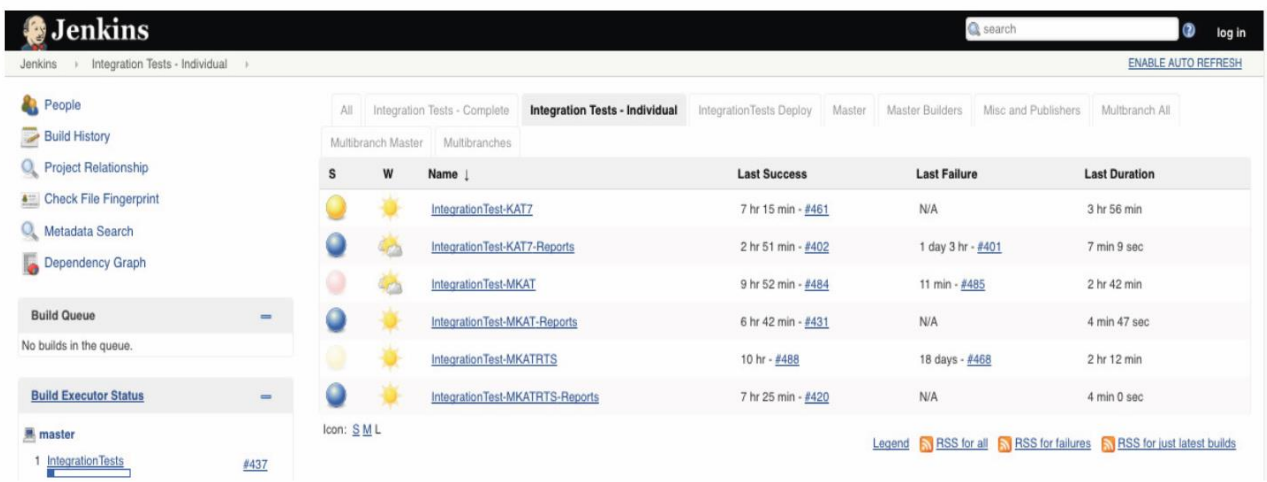

Figure 4: Continuous integration tests on Jenkins platform.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA164

REFERENCES

[1] L. Van den Heever *et.al.*, "CAM Development and Qualification Plan", SKA SA, Revision 5, (2015)

[2] S. Cross, "Guidelines for Communication with Devices", September 2017, https://media.readthedocs.org/pdf/katcp-python/latest/katcp-python.pdf

[3] Van Den Heever L., Swart P., Alberts T., Renil R., "MeerKAT Control and Monigtoring Requirement Specification", SKA SA, Revision 5, (2015)

[4] Jenkins with GitHub, Spetember 2015, https://jenkins.io/solutions/github/

[5] Building a Continuous Integration Pipeline with Docker, September 2017, https://www.docker.com/sites/default/files/UseCase/RA_CI%20with%20Docker_08.25.2015.pdf