Efficient Software Testing and Deployment for the ngVLA
ngVLA Computing Memo #1

P. Brandt

March 27, 2020

Abstract

Much effort has gone into streamlining software testing and deployment for large projects and large user-base tools such as ALMA and CASA. But NRAO has not yet achieved a fully automated software development pipeline for many of its software systems. The ngVLA will operate at a scale where automated software testing and deployment management will be critical to meeting the budget goals for operations, and for staff productivity.

1 Introduction

With NRAO preparing to tackle a project at the scale of the Next Generation Very Large Array (ngVLA), updated software development and test processes will be required to ensure reliable and cost-effective operations. Most companies now utilize sophisticated software pipelines following a continuous integration and continuous deployment (CICD) model. Several companies have even adopted such workflows as a cornerstone of their business model[1].

For the scale at which NRAO currently operates, the existing processes have worked well enough. However, with the advent of larger facilities of increased complexity, it is becoming increasingly apparent that a more modern approach to testing and deployment will soon be necessary to handle the scale of the observatory's software systems. This memo briefly discusses a possible range of processes that could improve how the observatory develops, delivers, and maintains software.

This memo is aimed squarely at the ngVLA telescope control software, though other groups could benefit from adopting similar processes. The processes described in this memo are also primarily discussed in the context of post-construction operations. However, investment in the infrastructure to support such processes will be important during early development and assembly, integration, and verification (AIV), and will aid the software testing required to accomplish both.

[1] For example, Google's "live at head"[4], a reference to immediately and automatically testing and, if successful, deploying developer changes as soon as they are committed to revision control. Netflix does this as well[2].

2 Requirements

The AIV concept and the operations concept documents lay out several requirements that directly support the need for a more automated testing and deployment.

The AIV concept explicitly states the need for "a comprehensive test framework to enable validation of stand-alone systems as well as partially integrated systems," "automated unit, integration, and regression testing suites," as well as "multiple software releases" during construction. It is also a stated goal of AIV to avoid "frozen" software releases, which "may require additional build and test infrastructure." [1]

While these items are discussed in the context of IPT deliverables of individual hardware and software modules or subsystems, similar needs and challenges requiring full-system testing will appear during commissioning and science validation (CSV), and will continue throughout the lifetime of the telescope. This is made clear in the operations concept by the need for concurrent science and testing subarrays with different software versions, even during steady-state operations [3].

Perhaps the most compelling factor is the project schedule. Early science, including pipeline delivery of science-ready data products (SRDP), is predicted to happen 3-4 years into the construction phase. With this constraint, it will be vital to have a full build, test, and deployment framework to allow rapid changes to software during development and early commissioning.

All of these factors combined make it clear that a project at the scale of the ngVLA requires a different approach to testing and deployment than that of the VLA or ALMA.

3 Value of Software Testing

At some point, it has probably become evident to almost everyone living in the modern era why software testing is important; every time a laptop, phone, or other computerized device "crashes" it creates inconvenience, and in some cases can lead to loss of vital resources or human life.

For completeness, we discuss a few salient reasons why software testing is important in the context of observatory operations. Though this memo is written specifically in the context of telescope control software, these points are equally applicable to almost every other software system in the observatory.

3.1 Fault Reduction

The primary justification for creating and maintaining test suites is usually to reduce and detect bugs. While true, it is important to keep in mind the caveat that many tests detecting defects can only be created after they have occurred at least once. Over time, the test suite ensures more robust software because of the hundreds or thousands of tests eliminating the possibility of different types of faults that have been found.

3.2 Confidence in Correctness

A major challenge for any software system is having confidence that the results it produces are correct, or as close to correct as possible. This is especially true when scientific results rely upon the software, as with data reduction and telescope control systems.

One method to maintain or increase the veracity of software is empirical testing; this is intended to prove that results which had previously been produced can be reproduced after changes have been made. Suites of tests accumulated over time can increasingly bound common, and sometimes uncommon but critical, developer mistakes to prevent them from reoccurring.

3.3 Repetition

With any change to critical software, it is important to do relevant testing. Testing is by nature repetitive, which can lead to important tests being skipped or performed incompletely. Though this is rarely intentional, it is often the result of insufficient knowledge of the effects a change may have on other parts of the system. Subjecting every software change to the same rigorous testing can catch issues that would otherwise fall through the cracks of a manual process.

Performing the tests can also be time consuming, so an automated testing pipeline would represent a large time savings for developers, engineers, scientists, and operators. Even in the absence of expert systems for data analysis, an automated testing pipeline that produces results easily analyzed by a human would be a great benefit over what is currently done with the VLA and ALMA.

4 Types of Testing

There are many types of software testing, and many levels at which the testing can be performed. Here we discuss some which are typically implemented for a distributed system with specialized hardware (i.e., a radio telescope control system). There may be additional considerations necessary for other supporting software, such as graphical user interfaces, databases, or maintenance tools used by engineers and technicians.

Not all levels of testing described below are done in every case, although that is typically the goal. As one moves up the hierarchy of testing, the cost of bugs and the risk of wasting resources are increased.

4.1 Unit Testing

The first level of testing is done at the scope of a single software module, such as a single file or software component (e.g. a single digitizer). This type of testing can often find simple bugs or violations of a component's interface requirements.

4.2 Hardware Emulation

This type of testing relies on additional software which simulates the real hardware at a reduced fidelity. Emulation can test basic expected behavior of the real hardware, and potentially for handling of invalid hardware configurations. Developing and maintaining this additional emulation software can be expensive for complex devices, however.

4.3 Hardware-in-the-Loop (HIL)

HIL testing involves using actual hardware components, or subsets of hardware components, in conjunction with their associated software. This can support the testing done in hardware

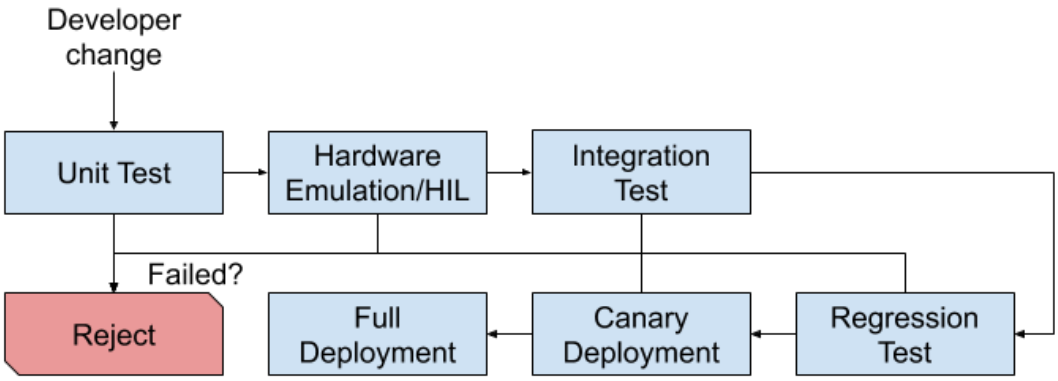

Figure 1: an example of a testing and deployment pipeline.

emulation, and further check that the hardware is kept in a functional state in different use cases.

For the ngVLA it is planned that antenna hardware test platforms will be maintained as part of an integrated test framework [1]. With the coordination of the electronics group, these testbeds could be used for HIL software testing.

4.4 Integration Testing

Integration testing involves bringing many software components together to ensure their interoperation stays consistent. This can cover basic assumptions about interfaces with respect to each other, as well as timing and synchronization. Integration testing can be done with hardware emulation, or hardware in the loop.

4.5 Regression Testing

The final level of testing involves using the full system, including hardware, to run a suite of standard modes. As the name implies, the purpose of regression testing is to prove that the final output of the system has not degraded. For an antenna control system, this would be running full observations, after which the data would be processed and checked for scientific validity.

5 Testing Automation

An automated test environment has several benefits, including but not limited to the following.

It provides a standard suite of tests that are typically designed to verify that common or past failure modes are not triggered. This addresses the "confidence" aspect that the software output has not changed[2]. Such tests can also be done manually, but in many cases the setup involved is time consuming and the task can be repetitive, which can lead to them being skipped.

[2]With the obvious caveat that sufficient tests have been developed to support that assumption.

Automated testing can also provide important localization of where bugs are introduced. It is not uncommon for "trivial" changes to be put into source code with limited or insufficient testing, or for seemingly innocent changes to have unintended and unexpected effects on another part of the system. Thus it is important to have immediate feedback on changes from as many levels of testing as possible. Ideally, even system level tests, including hardware, could be run at any time[3].

Automated testing platforms, such as Jenkins or Bamboo, can be configured to provide additional metrics such as static analysis, test coverage, and benchmarks. These tools can reveal trends in the performance of the system, or show when a sudden change in behavior began.

Lastly, test automation provides a uniform test environment for all changes. Eliminating configuration errors due to individual developers' systems is vital in avoiding "works for me" syndrome and time spent debugging issues unrelated to the ongoing work.

6 Deployment Automation

Most software deployments at NRAO are done after long development cycles in an all-or-nothing fashion (appropriately called "big bang"). This type of deployment can lead to periods of turbulence while bugs are discovered and fixed, and issues with the deployment on production hardware are solved.

By contrast, in most of the software industry deployments are done in phases, often with multiple versions operating in tandem. There are a number of different approaches for phased deployments, but one that is particularly appealing for the ngVLA control software is canary deployment.

In a canary deployment model, new software would be deployed to a select few components or antennas, and run in tandem with the old software. Provided the automated testing infrastructure described above is in place, feedback from testing can be gathered and analyzed. After a period without any significant issues, the new software could be rolled out on more and more antennas.

This type of deployment is well suited to the ngVLA because of the sub-array driven commissioning and maintenance model. This type of deployment was less feasible with the VLA and ALMA due to the relatively small number of antennas and difficulty in utilizing sub-arrays with different software versions, respectively.

In the long term this entire process, from testing to deployment, could be completely automated. This would free up time currently required from developers, computing staff, scientists, operators, and engineers involved in the release process that the VLA and ALMA use, representing a huge cost savings. The feedback loop from a committed change to verified performance is appreciably shortened in such a process too, improving fault localization, which may improve the overall productivity of the software development team.

7 Challenges

The processes described so far rely on several innovations or updates in NRAO's technology portfolio. The most relevant of these are described below.

[3]Though in practice, a daily or weekly time allocation of telescope resources is more realistic.

7.1 Automated Data Analysis

Full system testing and deployment automation for a control system would necessarily involve automatic data reduction and evaluation. The primary use case would be to identify when antennas running new versions of software are creating artifacts in the data, or otherwise failing at a higher rate than they were previously.

However, research in automatic astronomical data categorization with machine learning is currently being done by staff at NRAO for several purposes, for example RFI identification. Other "expert" systems will be developed to perform predictive maintenance and hardware failure analysis. Data calibration and imaging pipelines must be developed in order to realize SRDP from the ngVLA.

The work on all of these systems would feed directly into creating an image analysis system to detect whether a new software version is working well enough for a broader deployment, or if it is unsuitable for PI observations. Such a system would largely be a simplification of RFI detection, SRDP, and hardware failure identification, as its role would be to identify whether the data had changed significantly, without necessarily localizing a specific type of failure[4].

7.2 Scheduling

Several critical parts of facility management are still done by humans for the VLA and ALMA. One of the most difficult to automate well may be maintenance and test scheduling. It will be crucial to have automation of scheduling at many levels in order to maximize the science based on factors like weather, hardware availability on the array, and whether antennas may become available in the near future[5]. Additional responsibilities include automatically performing periodic calibrations, hardware self-tests, and of course running the software tests discussed in this memo.

7.3 Data Science

Despite these two words having common utterance within the observatory, data science[6] is relatively new to NRAO. Gathering data around these processes, and the behavior of the system adapting on its own in response, is key to making them function correctly.

Initiatives have been started within ALMA to use data science to improve observing efficiency, including many efforts to do predictive maintenance. The ngVLA design, as well as the integrated test framework design, will need to include additional monitor data points and diagnostic data streams at a high enough cadence to provide sufficient insight into the system behavior. This data then needs to be stored in a format that can be regularly queried by diagnostic systems, as well as interpreted in an easily human readable format.

[4] Localization would be useful for further analysis by humans, but not vital for the immediate needs of an automation pipeline.

[5] For example, if the telescopes are split into two separate science arrays, they may be rejoined to form a large array once both observations are complete. This could also factor in when maintenance is being performed and reintegrating antennas after they have been fixed.

[6] Per Wikipedia[5], "a concept to unify statistics, data analysis, machine learning and their related methods in order to understand and analyze actual phenomena with data."

8 Summary

In order for the ngVLA to succeed, it will be critical to reconsider the approach used for software testing and deployment. As described in this memo, the important elements will be automating many of the processes currently performed by humans: the various levels of software testing, analysis of test observations, scheduling, and managing software deployments.

If these processes are successfully adopted, it will represent a huge leap forward for how the observatory manages its facilities. This innovation could also pave the way for future projects to utilize the infrastructure developed to support the ngVLA.

References

[1] C. Langley et al. Assembly, integration, verification (aiv) concept. 2019.

[2] E. Bukoski et al. How we build code at netflix. `https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15`, 2016.

[3] E. Ford et al. Operations concept. 2018.

[4] Google. Why adopt abseil? `https://abseil.io/about/philosophy`, 2017.

[5] Wikipedia. Data science. `https://en.wikipedia.org/wiki/Data_science`.