NORTHWEST NAZARENE UNIVERSITY

Creating and Configuring a Continuous Integration Machine for a Software Package

THESIS

Submitted to the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

Nathan Emerson

2017

THESIS

Submitted to the Department of Mathematics and Computer Science

in partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

Nathan Emerson

2017

Creating and Configuring a Continuous Integration Machine for a Software Package

Author: Nathan Emerson

Nathan Emerson

Approved:

Barry Myers, Ph.D., Professor of Computer Science, Department of Mathematics and Computer Science, Faculty Advisor

Approved: Apr 19, 2017

David Johnson, Senior Software Engineer, Schweitzer Engineering Laboratories, Inc.

Approved:

Barry Myers, Ph.D., Chair,

Department of Mathematics and Computer Science

ABSTRACT

Integration of a Continuous Integration Machine with Automated Functional Test Suite for a Software Package.

EMERSON, NATHAN (Department of Mathematics & Computer Science), MYERS, DR BARRY (Department of Mathematics & Computer Science).

A system for running Continuous Integration (CI) testing on a commercial software package called Device Manager, created at Schweitzer Engineering Laboratories, Inc., was created using Bamboo, an enterprise business tool from Atlassian. This system utilizes existing tests and makes them compatible with Bamboo. The system uses Windows commands to utilize file operations on the host system and pulls files down from code repositories. The CI machine builds the solution using a code base, then runs tests if the build is successful. The system runs through two types of tests. First, it executes several thousand unit tests, which test source code functionality. Second, the CI machine records all the test results and build files and publishes them as artifacts that a user can download and utilize for debugging or manual testing. The CI machine was completed and successfully runs on each new build of the software, providing pass or fail values for tests that were run, which gives Software Engineers the knowledge needed to correct or maintain the software's codebase.

Acknowledgments

I would like to thank the Device Manager team at SEL that were on the team throughout the length of my work on the project. David Johnson, Alan Morrison, John Bird, and Chad Johnson, thank you for your support throughout my internship and my work on this project. I would also like to thank Amanda Marble, without whose help with career-advancing activities and resume guidance I may not have been set up for success in my internship. I would lastly like to thank Barry Myers and Dale Hamilton, the professors I have learned from over the last four years of class.

Table of Contents

Table of Figures

Background

When creating a software product, several different stages and milestones are achieved before the software is released to the public. These stages include planning, implementing, and testing. When implementing a software product, code produced by a software engineer needs to be verified for correctness and functionality. This is commonly referred to as unit testing. "A unit test is a piece of code that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions are wrong, the unit test has failed" (Osherove 4). These unit tests are executed by a developer before their changes are put into production code, but for every version of the software built before a public release, unit tests need to be run so often that it becomes burdensome. Each test needs to be run for each build in order to verify consistent quality of the software product. This is where a Continuous Integration (CI) machine comes into the picture.

A CI machine automates several of the stages that a developer would normally take to perform all created unit tests, as well as extend the possibility of what resources a developer has available. All of this happens on a Virtual Machine (VM). A VM is used because a developer has higher control over the state of the Operating System. The developer is able to control the software, settings, and system resources on the VM. This limits the variability of the environment that the CI build will run in. An example of a fully-functional CI machine is where the machine pulls the source code from one or more repositories, compiles and builds it, then runs unit tests on the built code. After the

unit tests have completed, the CI machine generates a report of the test results, showing where failures occurred, if they did occur at all. CI machines are used on products that go through several milestones and versions in order to check consistency across the entire application.

Schweitzer Engineering Laboratories, Inc. (SEL) produces a product called QuickSet, which allows power engineers to work directly with settings on devices within a power substation. These devices include digital electric relays, metering devices, security devices for the power grid, and communication devices that can group several devices together. Within this QuickSet application is a plugin called Device Manager. Device Manager is aptly named, because it extends the capabilities of QuickSet and allows the user to manage the devices themselves instead of just their settings. This includes but is not limited to the names and IDs on the devices, the connection paths the devices take, and the versions of settings on the devices. It also manages any scripts, passwords, or users that may interact with these devices. A project that worked with Device Manager was desired for work on a senior capstone.

Deciding on a worthwhile project was difficult. There was an opportunity to combine work for this project with an internship at SEL. In this way, the project overlaps both the academic and professional spheres, giving benefit to both. To work on the software package that the team is developing would be too large of a project and unfeasible because of the types of tasks that would likely be assigned in addition to this project. On top of this, the time constraints placed on the project within the software development cycle were a challenge. Each milestone and the work done on that

milestone would be too small and would require too short of a time frame for the requirements of this project before moving to a new task.

The idea of moving existing CI machine's functionality to work on Bamboo was brought up. Bamboo is a CI build machine tool produced by Atlassian, and it couples with Atlassian's other tools to make a combined single experience for the user. After consideration, this project fulfilled the needs of a group of users, was wanted by those users, and met the requirements of the project scope. The CI machine capability that the team was currently using was adequate, but lacked several features that had the potential to save the team time and effort.

When work on Bamboo was initialized, the amount of people in the company that were using it for their team was low. Since the Atlassian tools were new, most teams that had migrated to using them were still learning how to use the error reporting and repository management tools. Bamboo was largely pushed to the side as something to focus on once everything else is complete. Because of this, there was not a lot of internal documentation from other SEL employees that could be used to help with understanding or implementation.

Planning

At the start of the project, the Device Manager team already had a CI build machine running and functional, using a service called Continua. This service had many features that the team wanted and needed. What Continua did not have was a good tie-in to other software systems and tools that the team used. The team had recently transitioned to using the Atlassian suite of tools, which contained tools such as Jira, Stash/Bitbucket, Confluence, and Bamboo. Jira is an issue tracking tool, where team

members are assigned tasks to do, and tasks are created that relate to one another, leading to easier issue management. Stash is a tool that hosts repositories on a server. Partway through the project, Atlassian changed the name of Stash to Bitbucket, and it will be called that throughout the rest of this document. This repository also has pull request features. This means that a user can make code changes then use a version control system (like Git) to commit these changes, push them to a server, then "request" a merge of the changes to the "main" branch upon approval of a set of reviewers. Confluence is a knowledge base, which can take several different meanings depending on what a team is using it for. Some teams store specifications there, others store how-to documents, and others use it for project lifecycle tracking. Bamboo is Atlassian's CI machine and build deployment tool. This allows Bamboo to run as a CI machine (like Continua) and also send projects out or on to their next stage in the development cycle.

While Continua was good, it was not great for meeting the needs of the team. With unit tests running in Bamboo, a failed test could occur, and Bamboo could easily create an issue in Jira, rather than having to manually type out the issue and ensure its correctness. Bamboo does this work for the developer. Since the developer is not spending time writing up issues, they can be actively solving the issue instead.

Execution

Bamboo uses five different categories of granularity within a Bamboo project. There is the project itself, the plan, the stage, the job, and the task. The "project" is an umbrella term, which is often used for teams to put several different CI machine setups underneath. For instance, the project was named "Device Manager CI" for this project. Under this project may be several plans. Each plan is a specific build and test

environment for the software package. An example may be if a team wanted to test their software on a Windows 10 platform and also a Linux Ubuntu platform, they may create a plan for each and have them run simultaneously.

A plan consists of several subcategories (Figure 1): stages, jobs, and tasks.

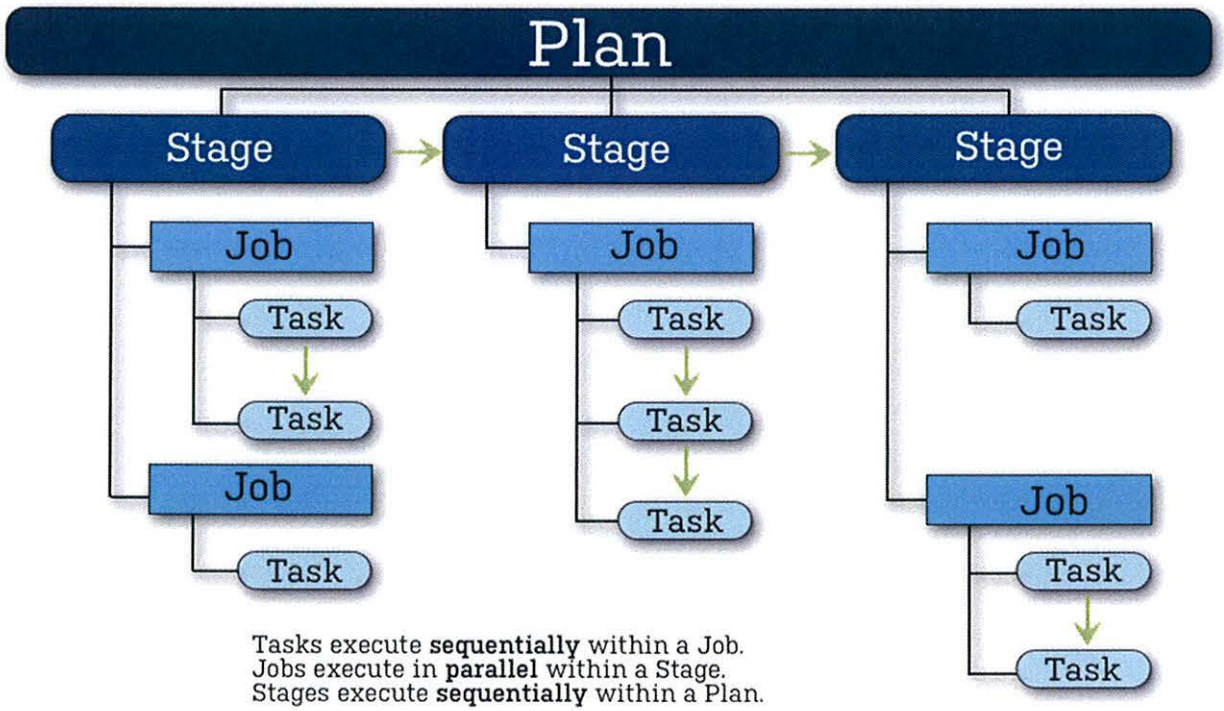


Figure 1 - Bamboo Plan (Bamboo)

When a plan has more than one stage, the stages execute sequentially, they may not execute out of order. For this project, there is only one stage. A user may want multiple Stages, to be used for building the software, testing the software, perhaps installing the software, and running automated functional tests. Another possible Stage is a deployment stage, where the software is pushed out to a public server for download or to the next step in the software development lifecycle, depending on if the tests in previous Stages passed or failed.

A job is executed at the same time as other jobs within the stage. Each Job could also be what has been described above for potential various Stages. In this matter it is up to the Bamboo plan creator to determine if the tasks that need to be done should be done sequentially or in parallel. Also in the case of this work, there is only one Job. The Job is titled "Compile and Run Unit Tests." In the diagram there are actually two Jobs, the other being "Run Automated Functional Tests." When this project started it was the intent to get Bamboo running then to add the functional test suite that tests the software as a user would see it. However, due to the evolving needs of the team, that ended up dropping off of the priority list, as the software projects that the team was working on took a turn towards a different testing implementation. The Job that was created, however, had several tasks in it.

A task is the smallest level of project building, as it defines exactly what should happen and when. Tasks run sequentially, so this is as close to programming as possible for a Plan like this. A task for this project could be checking out the code from Git repositories, building the code, or copying files to various locations on the VM.

The first step undertaken was analysis of the current Continua CI machine. In order to get a complete and accurate analysis, each task needed to be understood enough to port over to Bamboo. Continua and Bamboo have similar functionalities and abilities, but they do each task in slightly different ways. A task that was different was how each handles file management. Continua has built-in procedures that move files, while Bamboo relies on the user to create scripts that will do that when called. Most of the built-in procedures of Continua do not match up well with built-in features of

Bamboo, so most of the tasks in Bamboo are called using a command line script on the VM.

These differences were not realized at first. Initially, switching the tasks from one CI service to another seemed to be fairly straightforward. The tasks included checking out the source code from all the repositories needed, compiling three different solution files, running the unit tests, then parsing those unit tests. In Continua, all of these tasks had built-in components, able to run each just by giving some parameters and locations in the options for that task. For example, there was a compile using Visual Studio command in Continua. A similar Visual studio compilation task was available in Bamboo, but each time it would try to compile the code, the compilation would hang indefinitely and the build would go on without ceasing until a user manually stopped the build. Often this was found out when a build would be started right before clocking out for the day, returning the next time to the office (typically about 20 hours later) and finding the build still running, stuck on a compilation task, not advancing or giving any sort of indication that an error had occurred. When this happened, the run would be stopped, the log file downloaded, and it would be scoured to find what went wrong or what happened.

What was found is that one of the services was run by a user, and since no user had been assigned, it used the default value, which is where the local system acts as the user to the service. Because this is an SEL application in a test build, an SEL username was required for that step, so the fact that no SEL user was introducing that step was holding up the build process while it waited for an SEL account to allow access to a certain feature of the program.

Once this was discovered, an attempt was undergone to remove restrictions on the VM that the build was on. Running the Bamboo service uses the local system to run. This was changed in order for the service to run off of a username that the Windows operating system recognized. This was the username of an SEL employee. It worked, and it continued to work until later in the process of creating the implementation of this tool for use by the Device Manager team. Because this solution worked short-term, it was kept and it allowed for progress on other parts of Bamboo. Later in the project, the service was updated to use a username that was created specifically for the purpose of use on CI build machines.

After this setback, much of the task building and ordering was fairly straightforward. As seen in Figure 2 (A.2), fifteen tasks were created. The first two tasks were "Delete [Unit Test NLog Files]" and "Clear NUnit Cache." These tasks were put in place to clean up any potential residue left over from tests that ran before the current build, in an attempt to get the VM to as clean of a state as possible, limiting outside factors that could influence the test results. The next step found all the repositories that Device Manager needed code from and checked out the specified branch that the code was to run from. This task could be customized to fit each plan. For instance, if one plan was to always test production code, it would stay on the Master branch. If a plan was to test a certain step in the software development lifecycle, then it would be set to a task branch or a project branch, all hosted in Bitbucket. This shows another useful tie-in to the other Atlassian tools, for Bitbucket and Bamboo work seamlessly with one another on tasks like this one.

After copying a DLL file to another location on the VM, the Job starts to compile three different Visual Studio solution files. It compiles a common library that SEL uses, it compiles the project itself, then it compiles a separate solution that includes the unit tests for the project. To run a lot of the unit tests, a pseudo-database needs to be set up, so the next task does just that. After copying some build files for artifact saving later, Bamboo then starts running the unit tests. Since Device Manager is written in C#, there is an open source unit test framework called NUnit that is also written in C#, so that framework is used for writing and executing unit tests. Bamboo has an NUnit runner, but as mentioned before, the configurations did not meet the needs of the team, so a general command line script was used instead. After the tests ran to completion, there needed to be a way for the tests to be validated and checked for passing or failing indicators.

Out of the fifteen tasks, only two were not started using a command line script. Parsing the NUnit results was one of them. The NUnit tests are parsed by this task and displays within Bamboo to show what tests passed, what tests failed (if any), and shows how many tests were skipped (if any). Skipped tests are skipped because of the C# code in the test itself. Skipped tests are reported in the final output (Appendix A.8, A.9) This task is where it all comes together and becomes a usable software tool, becoming valuable to the team. Once this step is complete, some files are copied in order to be used as artifacts, namely the Device Manager NUnit log files. Now that the Stage has completed, the build completes, and the result is a pass or fail on the build, artifacts from the build, and a clean VM. Once the build finishes, it reverts the VM to a clean state, removing all SEL software to protect against potential influence of remnant code.

Conclusion

As the implementation on Bamboo was completed, several ideas and lessons were learned. One lesson learned was how to analyze actions taken by people before any involvement with a project to see what they meant by taking those actions. This was shown in the ability to look at the way that Continua was set up, analyze what it meant, and adapt the tasks to fit a Bamboo implementation of the same goals.

Porting the Continua CI machine to a Bamboo implementation was a valuable and important step in the team at SEL. By moving the build machine over from a good service to a great service that also has tie-ins to other services the team is actively using, the benefits are invaluable. To have a consistency between tools is desired, and it helps that the GUI look and feel is the same throughout the tooling, making it simple for a user to navigate between issues and builds, documentation and code repositories. Throughout this project the ability to communicate effectively was necessary, and throughout the experience more effective communication was learned.

A challenge experienced throughout the project was the lack of knowledge about the system and what Bamboo could do without having to write the scripts myself. Because each build took upwards of fifteen minutes, only a couple builds could be run a day if there was to be time between builds to analyze the results and fix or add to what was already there. Another challenge was the time commitment between working on Bamboo and working on other tasks assigned for different projects. Because this project was completed while at work, other tasks assigned that were not part of this project needed to be completed at the same time. Those tasks were often maintenance tasks

where a small issue would be fixed, but there was also a large addition to the software interface and user experience that needed to be done, and that often lasted for several hours at a time. Most of the work on the Bamboo project was completed while taking classes at school as well, so only part-time work during weekdays was available.

Overall, while project requirements change and timing does not always work to previous plans, this project became functional and usable by the Device Manager team, and the value of this Bamboo CI machine will only continue to grow as more is added.

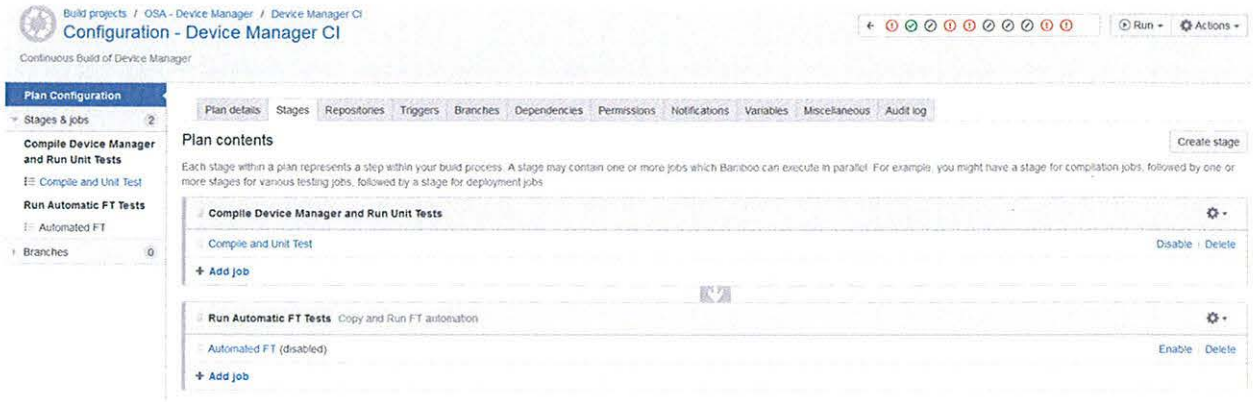
Works Cited

Bamboo Plan Anatomy. Digital image. *Atlassian*. Atlassian, 2017. Web. 30 Mar. 2017. <https://confluence.atlassian.com/bamboo/configuring-plans-289276853.html>.

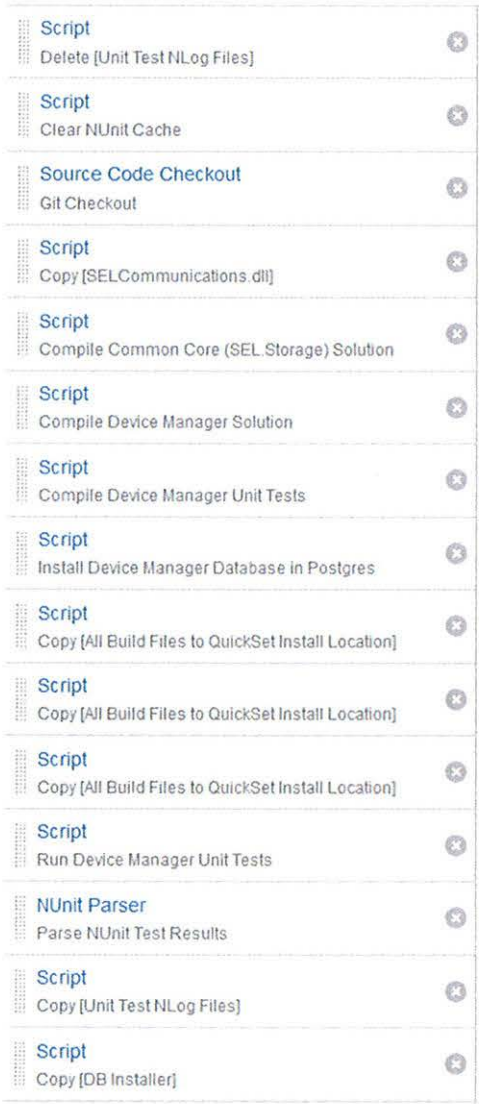
Osherove, Roy. *The art of unit testing: with examples in C#*. Shelter Island, NY: Manning, 2014. Print.

Appendix

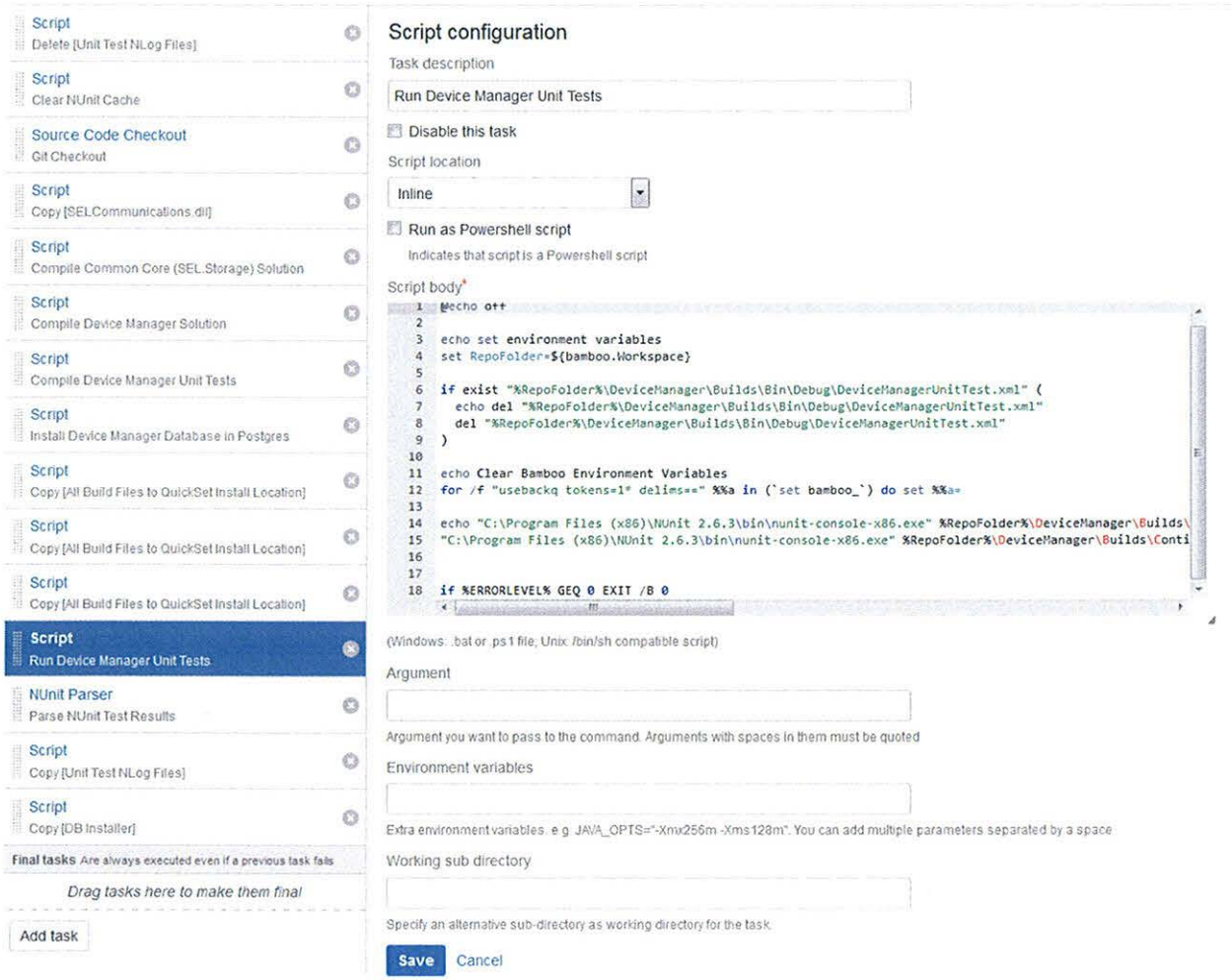
A.1 – Bamboo Project View



A.2 – List of Tasks on "Compile and Unit Test" Job



A.3 – View of the task editor



A.4 – Build Results

✓ #24	Rebuilt by Nathan Emerson	1 month ago	6 minutes
! #23	Rebuilt by Nathan Emerson	1 month ago	56 seconds
✓ #22	Rebuilt by Nathan Emerson	1 month ago	7 minutes
! #21	Rebuilt by Nathan Emerson	1 month ago	8 minutes
✓ #20	Rebuilt by Nathan Emerson	1 month ago	6 minutes
! #19	Rebuilt by Nathan Emerson	1 month ago	6 minutes
! #18	Rebuilt by Nathan Emerson	1 month ago	6 minutes
! #17	Rebuilt by Nathan Emerson	1 month ago	7 minutes
! #16	Rebuilt by Nathan Emerson	1 month ago	6 minutes
! #15	Manual run by Nathan Emerson	1 month ago	6 minutes
! #14	Rebuilt by Nathan Emerson	1 month ago	7 minutes
! #13	Manual run by Nathan Emerson	1 month ago	7 minutes
! #12	Manual run by Nathan Emerson	1 month ago	5 minutes
! #11	Rebuilt by Nathan Emerson	2 months ago	1 minute
! #10	Rebuilt by Nathan Emerson	2 months ago	1 minute
! #9	Manual run by Nathan Emerson	2 months ago	1 minute
! #8	Rebuilt by Nathan Emerson	2 months ago	1 minute
⊘ #7	Rebuilt by Nathan Emerson	2 months ago	1230 minutes
! #6	Manual run by Nathan Emerson	2 months ago	2 minutes
! #5	Manual run by Nathan Emerson	2 months ago	1 minute
! #4	Manual run by Nathan Emerson	2 months ago	1 minute
! #3	Manual run by Nathan Emerson	2 months ago	1 minute
⊘ #2	Rebuilt by Nathan Emerson	2 months ago	Unknown
✓ #1	Manual run by Nathan Emerson	2 months ago	16 minutes

A.5 – Sample output during a build runtime

A.6 – End-of-project Build View

#84	Manual run by Nathan Emerson	2 days ago	16 minutes	6265 passed
#83	Rebuilt by Nathan Emerson	2 days ago	16 minutes	3 of 6265 failed
#82	Manual run by Nathan Emerson		Unknown	Testless build
#81	Manual run by Nathan Emerson	3 days ago	16 minutes	6265 passed
#80	Rebuilt by Nathan Emerson	3 days ago	16 minutes	6265 passed
#79	Manual run by Nathan Emerson	3 days ago	16 minutes	6265 passed
#78	Manual run by Nathan Emerson	3 days ago	< 1 second	Testless build
#77	Scheduled	3 days ago	15 minutes	1 of 6265 failed
#76	Scheduled	4 days ago	15 minutes	6265 passed
#75	Scheduled	5 days ago	15 minutes	6265 passed

A.7 – Completion of a Continua CI build

A.8 – Bamboo test results with a failed test

Test results

6,295 tests in total | **1** test failed | **1** failure is new | **71** tests were quarantined / skipped | **14 minutes** taken in total

New test failures 1

Test
RdbFileInfoTests FinalizeFileRecovery_FileNotRecovered_UsingNewFile

```
TearDown : System.IO.IOException : The directory is not empty.
```

Skipped tests 71

Test	Failing since
DeviceDispatchBuilderTests Shutdown_DeadChild_HardShutdownInvoked	
ImportExportUtilityTest ExportObjectsToFile_ExportWithCustomPassword_ExportSuccessful	#112 (Scheduled)
ImportExportUtilityTest ExportObjectsToFile_ExportOptionalDataWithLargeObjectsWithCustomPassword_ExportSuccessful	#112 (Scheduled)
ImportExportUtilityTest ExportObjectsToFile_ExportIntegratedSettingsOptionalDataWithBlankPassword_ExportSuccessful	#112 (Scheduled)

A.9 – Bamboo test results with the failed test fixed

Test results

6,349 tests in total **1** test was fixed **71** tests were quarantined / skipped **9 minutes** taken in total

Fixed tests 1

Test	Failing since
RdbFileInfoTests FinalizeFileRecovery_FileNotRecovered_UsingNewFile	#400 (Scheduled)

Skipped tests 71

Test	Failing since
DeviceDispatchBuilderTests Shutdown_DeadChild_HardShutdownInvoked	
ImportExportUtilityTest ExportObjectsToFile_ExportWithCustomPassword_ExportSuccessful	#112 (Scheduled)
ImportExportUtilityTest ExportObjectsToFile_ExportOptionalDataWithLargeObjectsWithCustomPassword_ExportSuccessful	#112 (Scheduled)
ImportExportUtilityTest ExportObjectsToFile_ExportIntegratedSettingsOptionalDataWithBlankPassword_ExportSuccessful	#112 (Scheduled)