

## Chapter 1

In the beginning there was Clae, Josh and Alex on a quest to build a magnificent deprecated project. While we first thought this would be an easy task due to the sheer number of project choices before the mighty Another-One-Bytes-the-Dust team, we were quickly proven wrong when our first, second and third project choices failed to bear fruit. As a result, we switched to Moodle, and immediately saw the benefits. But first, let's talk about failures.

### Section 1 - Sigmah

Our original choice was Sigmah, a free software developed to help international aid organizations manage the information from their projects including reports, indicators, schedules, documents, etc. We chose this project because it was written in java and seemed to have some pretty good documentation. For a while we had success in the first step of building the project, which involved building the postgres server that handles our database. But after this we were given the task to copy and adjust the pom.xml file as well as the settings.xml of the Maven build. Having tried to run it with the command “mvn install -Psigmah-dev” we received a fatal error message that included the following:

“org.apache.maven.plugins:maven-compiler-plugin:3.1:compile.” (**Figure 1.1**) We tried searching stack overflow for a source of the issue, but there wasn't very good documentation on what we were experiencing. We did find one article suggesting to run Maven clean install which did not work. We later added the <build> tag below to the settings.xml file which also didn't work (**Figure 1.2**).

**Figure 1.1**

```
BUILD FAILURE
-----
Total time: 2.591 s
Finished at: 2017-03-01T13:09:47+05:30
Final Memory: 21M/347M
-----
Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.1
/Users/vshukla/git/Prism/src/main/java/main/MainUITabbed.java:[26,22] pac
/Users/vshukla/git/Prism/src/main/java/main/MainUITabbed.java:[93,41] can
symbol: class Application
/Users/vshukla/git/Prism/src/main/java/main/MainUITabbed.java:[93,67] can
symbol: variable Application
-> [Help 1]

To see the full stack trace of the errors, re-run Maven with the -e switch
Re-run Maven using the -X switch to enable full debug logging.

For more information about the errors and possible solutions, please read
[Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureExce
```

**Figure 1.2**

```

</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>

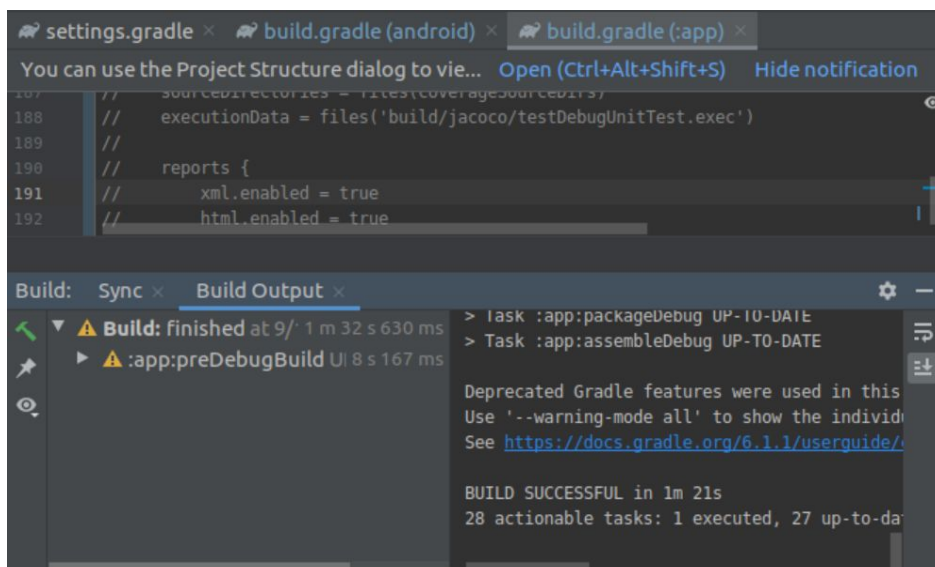
```

We came to a point where there were still other issues we were anticipating having to deal with where it was better to cut our losses and try Glucosio.

## Section 2 - Glucosio

After having exhausted ourselves attempting to remedy the issues with Sigmah, we reluctantly turned our attention to Glucosio. We chose Glucosio for a number of reasons including that it could be worked on either as an android project or an ios project. We thought that working on Glucosio would be a good opportunity to learn some objective-c, an apple based language becoming more frequently used. Unfortunately, we also ran into a number of issues with Glucosio.

Glucosio failed in the mobile app build due to one of the Jocosio dependencies no longer being supported and when searching for a solution we came up with no fix. So in a desperate attempt to keep the Glucosio train rolling, we commented out the Jocosio lines of code, and proceeded to run the build. We had some success, where the build would complete successfully (**Figure 1.3**).

**Figure 1.3**

So now we figured we were in the clear. All we had to do was get it to show up in the Android Studio emulator of our choice. However we ran into the issue where we had not enabled VT-x/AMD-V nested virtualization on the virtual machine. So we ran the command: "VBoxManage modifyvm Nubuntu2 --nested-hw-virt on" which turned on the nested virtualization feature. At this point the build was building successfully, and everything was as it needed to be. However, when running the project the emulator remained a black screen with no indicator that anything was wrong or even happening. This was a difficult problem to diagnose, as we may have been a few simple steps away. However, after searching for a solution we found none and decided to move on to the next option Miradi, which quickly turned into a decision to move on to Moodle.

### Section 3 - Moodle and Some Success

Despite our reluctance to abandon yet another project, we forced ourselves to reevaluate our options. We took a look at our third choice, Miradi, but quickly came to the conclusion that it was not in our best interest to pursue. We felt that Miradi lacked sufficient documentation, and instead opted for Moodle precisely because Moodle had a significant amount of documentation.

Moodle attracted the interest of our group both because of its level of documentation, and because we felt as a group that we could work effectively with this project. Additionally, the prospect of working with php, a language nobody on our team has used other than on occasion, excited all three of us. Sure enough, we fairly quickly worked through the entire build process and had Moodle up and running in about an hour. Nonetheless, we did run into a number of issues in our efforts to build the project.

Our first issue arose while editing the mysql configuration file. We ran into an error which stated: "The Job for mysql.service failed because the control process exited with error code." Turns out we were running a later version of Ubuntu where the three lines identifying the storage engine, the db files per table variable and the file format were not necessary to add. Accordingly, we went back and removed these lines (**Figure 1.4**).

**Figure 1.4**

```
default_storage_engine = innodb
```

```
innodb_file_per_table = 1
```

```
innodb_file_format = Barracuda
```

Another issue was dealing with the configuration.php file. We ran the command “chmod -R 777 /var/www/html/moodle” immediately followed by “chmod -R 0755 /var/www/html/moodle.” The first command allows reading, writing, and executing privileges to all users, but the problem arises when running the 0755 command immediately after the 777 command, which removes writing access to the users and essentially undoes part of the prior command. In the documentation, the flow is as if these commands should be run right after each other (**Figure 1.5**), however it would be more clear to specify that the 0755 command should be called later when necessary. If run too early though, it causes issues in setting up the configuration.php.

**Figure 1.5**

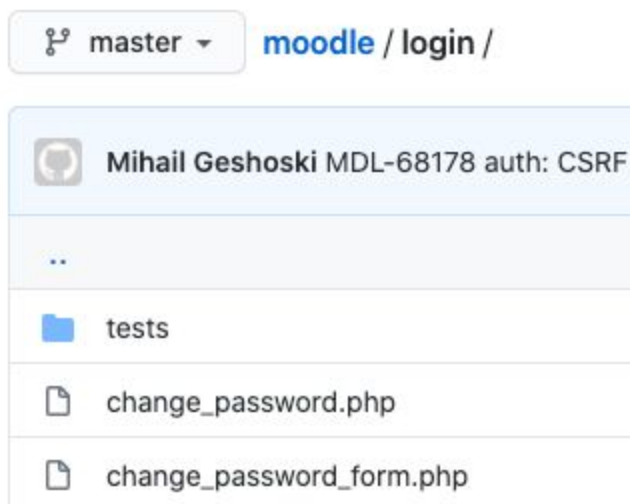
```
sudo chmod -R 777 /var/www/html/moodle
```

After you have ran the installer and you have moodle command.

```
sudo chmod -R 0755 /var/www/html/moodle
```

Now that we have Moodle set up properly and running on Localhost, it's time to examine the various testing folders of each component, and there are plenty to choose from. For instance, in the login folder, there is a folder inside labeled “test” (**Figure 1.6**).

**Figure 1.6**



If we go a little further into the tests folder, we find a php file called lib\_test.php with a class inside (**Figure 1.7**). It is becoming clearer that we are going to need some other tools in order to run this unit test along with the others, but how do you unit test php test classes?

**Figure 1.7**

```

class core_login_lib_testcase extends advanced_testcase {

    public function test_core_login_process_password_reset_one_time_without_username_protection() {
        global $CFG;

        $this->resetAfterTest();
        $CFG->protectusernames = 0;
        $user = $this->getDataGenerator()->create_user(array('auth' => 'manual'));

        $sink = $this->redirectEmails();

        list($status, $notice, $url) = core_login_process_password_reset($user->username, null);
        $this->assertSame('emailresetconfirmsent', $status);
        $emails = $sink->get_messages();
        $this->assertCount(1, $emails);
        $email = reset($emails);
        $this->assertSame($user->email, $email->to);
        $this->assertNotEmpty($email->header);
        $this->assertNotEmpty($email->body);
        $this->assertRegExp('/A password reset was requested for your account/', quoted_printable_decode($email->body));
        $sink->clear();
    }
}

```

In order to do this we needed two tools: Composer and PHPUnit. Composer is a dependency management tool for php while PHPUnit, as its name suggests, is a unit tester for php code. Getting Composer installed didn't give us much of an issue. However, when it came to getting the PHPUnit environment setup, there were a few extra things we had to figure out. We were confronted with our first issue when trying to run the Composer command to “require phpunit/phpunit” which would connect the phpunit dependency allowing us to setup the environment. The error message we were getting was in regards to the version of phpunit that we were trying to get. When you run “php composer require phpunit/phpunit” the version that is brought in as a dependency is that of the latest version, in our case version 9. Well the version of Moodle that we are running has unit tests that are only compatible with phpunit version 7. So after running the command “php composer require phpunit/phpunit ^7” we had success (**Figure 1.8**)!

**Figure 1.8**

```

josh@josh-VirtualBox:/opt/moodle$ composer require --dev p
hpunit/phpunit ^7
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Nothing to install or update
Package container-interop/container-interop is abandoned,
you should avoid using it. Use psr/container instead.
Package phpunit/dbunit is abandoned, you should avoid usin
g it. No replacement was suggested.
Package phpunit/php-token-stream is abandoned, you should
avoid using it. No replacement was suggested.
Writing lock file
Generating autoload files
12 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

```



Now it's time to run and everything should go perfectly as is always the case in running older software right? Wrong. We started dbdriver errors along with other errors that we weren't able to recognize at first, but after a while we started to notice a pattern (**Figure 1.9**). All of the errors were dealing with variables within our configuration file. We were under the impression that these variables were setting automatically given that we had successfully gotten moodle to run. However, even though we had the screen showing up properly, if we had tried to do something that involved accessing our mysql database, we would have run into some ugly issues. Thankfully we were able to catch this in the process of setting up the testing environment. So, sure enough, we entered the config.php file within our root moodle directory and noticed that certain variables had not been manually set (**Figure 1.10**). For example, "wwwroot" needs to be setting to "127.0.0.1/moodle" in our case. Another one of these examples was our dbdriver wasn't set to "mysql" along with our username and password being off. Once we adjusted for this issue, we had some promising results (**Figure 1.11**).

**Figure 1.9**

```
Debug info:
Error code: dbdriverproblem
Stack trace: * line 140 of /lib/dml/pgsql_native_moodle_database.php: dml_exception thrown
* line 231 of /lib/dml/moodle_read_slave_trait.php: call to pgsql_native_moodle_database->raw_connect()
* line 217 of /lib/dml/moodle_read_slave_trait.php: call to pgsql_native_moodle_database->set_dbhwrite()
* line 340 of /lib/dml/lib.php: call to pgsql_native_moodle_database->connect()
* line 624 of /lib/setup.php: call to setup_DB()
* line 220 of /lib/phpunit/bootstrap.php: call to require()
* line 83 of /admin/tool/phpunit/cli/util.php: call to req
```

**Figure 1.10**

```
// http://docs.moodle.org/en/masquerading

$CFG->wwwroot = 'http://example.com/moodle';
```

**Figure 1.10**

```
-->tinymce_moodleemoticon
++ Success ++
-->tinymce_moodleimage
++ Success ++
-->tinymce_moodlemedia
++ Success ++
-->tinymce_moodlenolink
++ Success ++
-->tinymce_pdw
++ Success ++
-->tinymce_spellchecker
++ Success ++
-->tinymce_wrap
++ Success ++
-->logstore_database
++ Success ++
-->logstore_legacy
++ Success ++
-->logstore_standard
++ Success ++

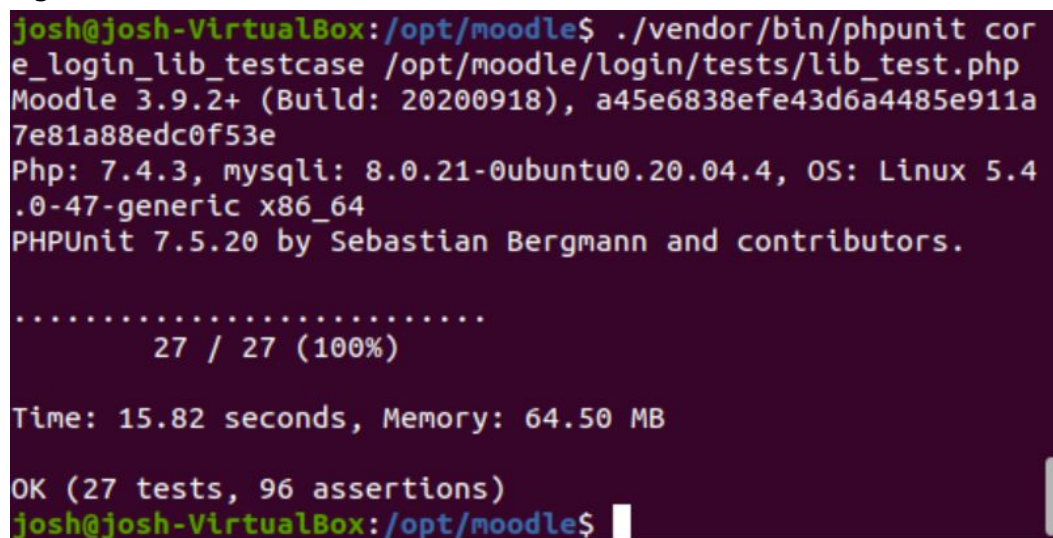
PHPUnit test environment setup complete.
josh@josh-VirtualBox:/opt/moodle$
```

As you can see in Figure 1.10 above, our PHPUnit testing environment has been properly set up. Now we can run some tests with the following command structure:

```
./path/to/phpunit      class to test  /path/to/test/file/to/test
```

So after running this command relevant to our machine, we got the output below (**Figure 1.11**). The output shows that 27 tests, were run from this class and there were 100% successful with 96 assertions returning boolean expressions. A successful test! Along with the lib\_testcase, we ran many other tests in many other folders of the different components of Moodle. The tests had the same successful outcome.

**Figure 1.11**



```
josh@josh-VirtualBox:/opt/moodle$ ./vendor/bin/phpunit core_login_lib_testcase /opt/moodle/login/tests/lib_test.php
Moodle 3.9.2+ (Build: 20200918), a45e6838efe43d6a4485e911a7e81a88edc0f53e
Php: 7.4.3, mysql: 8.0.21-0ubuntu0.20.04.4, OS: Linux 5.4
.0-47-generic x86_64
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

.....
27 / 27 (100%)

Time: 15.82 seconds, Memory: 64.50 MB

OK (27 tests, 96 assertions)
josh@josh-VirtualBox:/opt/moodle$
```

In conclusion of Chapter 1, we clearly learned a lot as a team. One of the main takeaways from this initial stage was how difficult it was to build/run outdated software that is dependent on outdated or deprecated dependencies. Also configuring software to run on your machine can be particularly difficult depending on your familiarity with the particular software. For example, we really struggled in getting the pom.xml and settings.xml files to work properly in the maven builds. This may be due to a lack of experience or a lack of documentation on the part of the tool. In either case, we had some real success with Moodle because of the fabulous documentation of the system. Which brings us to our final takeaway, that is, documentation, documentation, documentation! It is so important to properly document your projects. It could be the difference between a good-doer successfully contributing to your project and the slow-creeping irrelevance to everything you've worked towards.