Testing and Build Tooling Assessment

The testing analysis of the software project, on google code’s website, revealed several interesting points useful in teaching me about project models, code structuring and modularization. Prior to this phase of the project’s effort, I was largely ignorant of processes and utilities available in the automated build tooling realm, if I had heard of them. I had known of ant and maven and have configured them on Windows 7 before, build their hello world projects to observe their basic usage for learning purposes but not for more constructive purposes, such as exploring in context of an actual large-scale project. I learned Maven is somewhat like ANT and it presumably has all of ANT’s features plus more, but I am still unaware whether Maven can achieve exactly what ANT can, entirely, and whether I can improve the projects usage of it in an automated building context, by the end.

To describe the build of my project, I initially used Mercurial to clone the project’s codebase and followed extensive instructions after combing through a reasonable amount of website literature but found trial-and-error to be about as effective as any other learning strategy, when trying to understand the build and testing structure/process for my project. Two documents from the project code were of some assistance to me in setting up the build and test environment: [CONFIGURE.txt](https://code.google.com/p/opendatakit/source/browse/CONFIGURE.txt?repo=aggregate) (explains setting up a build environment in Eclipse IDE – see appendix (a)) and [README.txt](https://code.google.com/p/opendatakit/source/browse/src/main/libs/readme.txt?repo=aggregate) (explains pre-reqs for getting MAVEN to build/compile, regardless of using an IDE). To build the project using “maven install”, three steps are required, first, exclusive of whether the developer elects unit testing/IT based on the: MySQL, PostgreSQL, or GoogleAppEngine-Datastore database backend/endpoint:

1. The user must obtain the necessary .jar file for the Database/-Connection (ie:MySQLconnectorJ.jar)
2. A preliminary build inside the “odk-gae-settings” subproject must be completed using ant and the generated .jar moved inside the “eclipse-aggregate-gae\war\WEB-INF\lib”.
3. Inside src/main/libs folder, using the README.txt commands, additionally libraries must be added so maven is aware of them (This is the current instruction from the README file…)

Although, the build process has worked for me, to this point, at least as far as “maven install” has successfully completed all Phases of the build order, ie: without FAILURE, the level of automation associated with the build seems amenable to improvement. For starters there may be some inconsistency between downloaded jars and the versioning that maven looks for, so it is possible the settings.xml file could be updated, according to rules on maven’s website, under the: ‘dependencyManagementMechanism’ initiative.

**As a ‘DISCLAIMER’**: going into this effort AND as of this point, still, my level of experience with MAVEN, ANT, Maven2Eclipse plugin is minimal at best, and all the above occurred first on a Windows 8 environment, as the project’s primary developer/engineer uses Windows environment exclusively. The project seems to lend itself to Linux(/Ubuntu distro) testing and development, but I have not successfully executed the unit tests/IT *ON EITHER* environment, though, I am able to build and visit localhost in the browser of either to verify the sources/project compiles and runs successfully.

To explain the test processes in place, aside from the 3 database backend options available thereto, there are automated testing facilities available through maven and associated EMMA plugin to some degree. The code analyzed primarily serves to accept and validate XForm submissions to the system, which uses a java-based application server (Tomcat or GAE – see appendix (b)) to store the form into a database, so it mostly tests the Server/Upload functionalities and uses JUnit for that.

**Appendix: Additional considerations:**

1. Interestingly, at line 362 (point #12) in the CONFIGURE file referenced above, there is mention of using the Maven plugin for Eclipse (m2e) yet also that support for it is ‘dodgy in Eclipse,’ so not surprisingly, I did not have much luck building to any level of understandability or practicality using this Eclipse utility. However, I did take an IDE/GUI approach, building the odk-gae-settings subproject using ANT in Eclipse, which then involves ‘mov’ing the target/ .jar file from there to a sister directory, in order to build the overall project and run a local(-host) instance of the application from the generated overall-project .jar.
2. I downloaded several .JARs, which google provides to replicate its AppEngine (GAE) cloud-based server-datastorage environment, experimenting and learning about the testing environments associated to it (vs. the Tomcat backend) and walked through a ‘[GAE tests](https://developers.google.com/appengine/docs/java/tools/localunittesting)’ tutorial to analyze the AppEngine context.

**REPORTS AND CODE COVERAGE**

Surefire reporting module is used to generate/cover integration tests. However, instead of accommodating the Cobertura plugin for ‘testing -code coverage’, this project makes use of the EMMA maven plugin [from mojo.codehaus.org](http://mojo.codehaus.org/emma-maven-plugin/); and, whereas maven is always capable of producing Cobertura reports, says the default “mvn site”-produced literature on [EMMA's website](http://mojo.codehaus.org/emma-maven-plugin/project-reports.html), [EMMA produced reports](http://emma.sourceforge.net/coverage_sample_a/index.html) indicate coverage at the: METHOD and BLOCK levels, in addition to Cobertura’s CLASS-,LINE- levels and EMMA is reportedly faster than Cobertura ([per Sonar site](http://www.sonarqube.org/pick-your-code-coverage-tool-in-sonar-2-2/)).