1. ## Installing Software for Alps Developers:

    a. Install JDK 1.7.0+ . Install Java SE (not Java EE). .
        i. Here is one possible place you can get it from.
        (Please note that you must get the appropriate jdk for the 64 or 32 bit version of OS you may be running)
        ii. Run the installer accepting the defaults.
        iii. Set JAVA_HOME environment variable to your java directory (e.g. C:\Program Files\Java\jdk1.7.0_55).
        iv. if you are running OSX Mountain Lion (10.8) the you will likely already have Java installed. If not you can get it here
            1. One likely place to look for the installation is /System/Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home
            2. It's also possible that your $JAVA_HOME directory is not set. This may cause problems with maven when compiling. Here are some known locations
                a. /System/Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home
            3. Once you find your java home it's a good idea to set it in your bash profile if it is not already set. Like so
                a. export JAVA_HOME=/System/Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home
        v. To ensure you have things working correctly you can open a command line and run java -version. You should see 1.7.+ something.
    b. Install JCE jars (This is separately licensed and therefore needs an additional download):
        i. Download from here.
        ii. Unzip the download. Copy local_policy.jar and US_export_policy.jar to your Java security folder (e.g. C:\Program Files\Java\jdk1.7.0_55\jre\lib\security) after first renaming US_export_policy.jar to US_export_policy.jar.save and rename local_policy.jar to local_policy.jar.save.
        iii. Note that if you have a JDK **and** a JRE installed (which is likely), then you should update these files in the jre that is **with in** the jdk, not the external one because your $JAVA_HOME will most likely be the jdk one
    c. Eclipse: SpringSourceToolSuite recommended. Ensure that you have the m2e plugin installed by selecting Help | Install New Software... and clicking on the "what's already installed" link.
    d. Install Maven. Make sure you add the maven bin directory to your PATH environment variable (e.g. C:\Program Files\apache-maven-3.1.0\bin). Try running mvn to confirm that the JAVA_HOME environment variable has been correctly set (see above step).
    e. Install Git. For Windows, during the installation, *do not* select Windows Explorer Integration, select "Run Git from the Windows Command Prompt" and later "Checkout Windows-style, commit Unix-style line endings."
    f. Install the SourceTree git client. This makes it easy to follow git-flow which is the git workflow used within Lucas. Start SourceTree and you should be prompted with something to the effect that Git is not installed. Select "Use existing Git installation" and select "Don't use Mercurial". When prompted for user info, enter your name and Lucas email address.

2. ## Install and starting MySQL

    a. Download and install MySQL (5.6.10+).
    b. For Mac and *nix Users:
        i. cd /user/local/mysql (This is where MySQL gets installed on a Mac)
        ii. sudo ./bin/mysqld_safe
        iii. [Press ^Z and then bg]
        iv. When you initially install MySQL, you will only have user 'root' with no password (unless you specified one during install). Use that to log in by issuing: ./mysql --user=root --password= This will get you to a sql prompt
    c. For Windows Users:
        i. Unzip the download
        ii. Install the MySQL service by running the following at the command prompt: $MYSQL_HOME\bin\mysqld --install, where $MYSQL_HOME represents your local MySQL install directory (.e.g. C:\Users\sy\Downloads\ALPS\MySql\mysql-5.6.13-winx64\mysql-5.6.13-winx64\). Go to "Services" and start that service, of not already started.
        iii. From the same directory, in a different command window, run the MySQL.exe file like so:
        $MYSQL_HOME/bin/MySQL.exe --user=root --password=
        This gets you to a SQL prompt
        iv. Navigate to Windows Services applet and start the MySQL service. If desired, set to Automatic so that it starts automatically after each reboot.
    d. Download MySQL Workbench which is a slightly better user experience than the command prompt. Install the msi and follow default prompts. Bring up the workbench and click on the 'house-like' icon on the upper left. Then 'new connection' and fill in lucas/password.
    e. Then create a database (NOTE: These steps can be performed in the MySQL Workbench in Windows or in the command window)
    Linux users input: create database lucasdb --lower_case_table_names=1
    Windows users input: create database lucasdb
    NOTE: MacOSX to 2 and Unix/linux to 0. 1 guarantees that table names are stored in lowercase on disk and name comparisons are not case sensitive.
    To see the current value: SHOW VARIABLES LIKE 'lower%';
    f. Then create a user
    CREATE USER *'lucas'@'localhost'* IDENTIFIED BY 'password'. If this results in an error, ensure that you have an entry in /etc/hosts for localhost.
    g. Then assign privileges:
    GRANT ALL PRIVILEGES ON *.* TO 'lucas'@'localhost' WITH GRANT OPTION

h. Then from the command line:
./mysql --user=lucas --password=password
should get you to a sql prompt.

3. ## Installing RabbitMQ

   a. Please follow instructions here.

4. **Installing Protractor Dependencies**
   a. Please follow instructions here.

5. ## Checking out source

   a. First create an account on bitbucket.org. Use your lucas email address.
   b. Ensure that the bitbucket admin has added your account to the lucasware repo. Then log into bitbucket.org where you should see a lucasware repo
   c. Login to bitbucket.
   Click on the 'lucasware' repository (somewhere on the lower right of the page, says 'lucasware/lucasware')
   Change branch to 'develop' (not master) using the drop down list.
   Click on the "Clone" button on upper right and copy the url that looks like the below url.
   git clone https://<<yourBitBucketUserNameHere>>@bitbucket.org/lucasware/lucasware.git
   d. Make a directory where you would like to checkout the code. Lets call that LUCAS_HOME Typically this would be your IDE workspace.
   e. From that dir type the above url .This should check out the source into your local repo.
   f. The above steps would have required you to enter your password. To prevent you from doing this everytime, follow instructions here. After this you should be able to checkout without supplying a password.
   g. Switch to the 'develop' branch
   cd $LUCAS_HOME
   git checkout develop
   h. If you plan on using git bash to perform source control activities on BitBucket follow these instructions to create an ssh identity and have it pass your authentication credentials automatically. https://confluence.atlassian.com/display/BITBUCKET/Set+up+SSH+for+Git

6. ## Building the code base

   a. This step will be required to be carried out ONE time per developer instance. It creates a parent project that is used by all other projects. Once carried out, builds should be carried out from lucas-parent directory as outlined from step 2 below. Once the corporate repository is in place, this step will not be necessary.
      i. cd $LUCAS_HOME
      ii. Issue **mvn clean install -f lucas-parent/pom.xml**
   b. cd $LUCAS_HOME
   c. Issue **mvn clean install -DskipTests=true -f lucas-parent/pom.xml -Pall**
   This will build the app in the corresponding target directories

7. ## Running Tests

   a. For end-to-end tests in Angular, we need Firefox. Install Firefox.
   b. For running e2e tests on the angular stack, the jetty plugin deploys both the amd war and the api war to port 9020 and then runs the e2e tests against them. To enable both to run, we need to up the heap in which the maven process runs. To do so, issue

Upping heap in the bash shell

```
export MAVEN_OPTS=-Xms500m -Xmx500m -XX:MaxPermSize=256m
```

or

Upping heap on a DOS prompt

```
SET MAVEN_OPTS= -Xms500m -Xmx500m -XX:MaxPermSize=256m
```

Also, we need to change the **DNS** entries so that localhost and local-lucas-api are aliases:

Edit your /etc/hosts file and add the following entry:
127.0.0.1 local-lucas-api.com
The above is done so that you can simulate a cross domain access behavior in your local environment (Remember that lucas-amd and lucas-api are both hosted locally).
Windows users will find this file, most likely in C:\windows\system32\drivers\etc

c. cd $LUCAS_HOME
d. To build the code with unit tests, you must specify the encryption key as below:
mvn clean install -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f lucas-parent/pom.xml -Pall
e. To run unit tests and functional tests, issue:
mvn clean install -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f lucas-parent/pom.xml -Pall,alltests
f. To run unit tests and functional tests AND run liquibase scripts, issue:
mvn clean install -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f lucas-parent/pom.xml -Pall,alltests -Dskip.liquibase=false -Dliquibase.password=password
g. To run ***only*** tests on the Angular stack:
cd $LUCAS_HOME/lucas-amd
i. Only unit tests:
mvn clean test
ii. Unit tests + e2e tests
mvn clean install -Palltests -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk

8. Testing Code Coverage

Almost all tests are concentrated in the lucas-services module, however lucas-amd also contains unit and functional tests. Given that, the following commands can be used to generate code coverage reports for server side and client side tests you may have just written. There is advantage in running just one test (as against running the whole suite) because the whole suite may take long. This way, you run one test and view the line level test coverage in the class under test and then attempt to cover lines that are not yet covered by tests. *Please note* the ability to run only *one* test, is only currently possible for server side tests (lucas-services) and not client side tests (lucas-amd).

lucas-services

To run a single test, issue the command below:

```
cd $LUCAS_HOME
mvn clean cobertura:cobertura
-DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f
lucas-services/pom.xml -Dtest=<NameOfTestClass>
```

For example:

```
mvn clean cobertura:cobertura
-DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f
lucas-services/pom.xml -Dtest=LocationServiceFunctionalTests
```

When you run the above command, the maven-cobertura-plugin will run the ONE test that you have specified using the `-Dtest` option and generate the report in `$LUCAS_HOME/lucas- services/target/site/cobertura/index.html`. When you load up that html page, navigate down to the class that you are testing and ensure that you have **> 80%** test coverage.

In the event you want to **run the cobertura coverage report for ALL tests**, issue the command below:

```
mvn clean cobertura:cobertura
-DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f
lucas-services/pom.xml -Palltests
```

lucas-amd

Generating reports when building all lucas projects (Note: it's important to include the `alltests` profile here, otherwise the lucas-services report will not be generated). This will produce coverage reports in lucas-amd/target and lucas-services/target

```
cd $LUCAS_HOME
mvn clean install -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f
lucas-parent/pom.xml -Pall,alltests
```

Generating coverage report from within amd project for *only* the lucas-amd tests (unit tests):

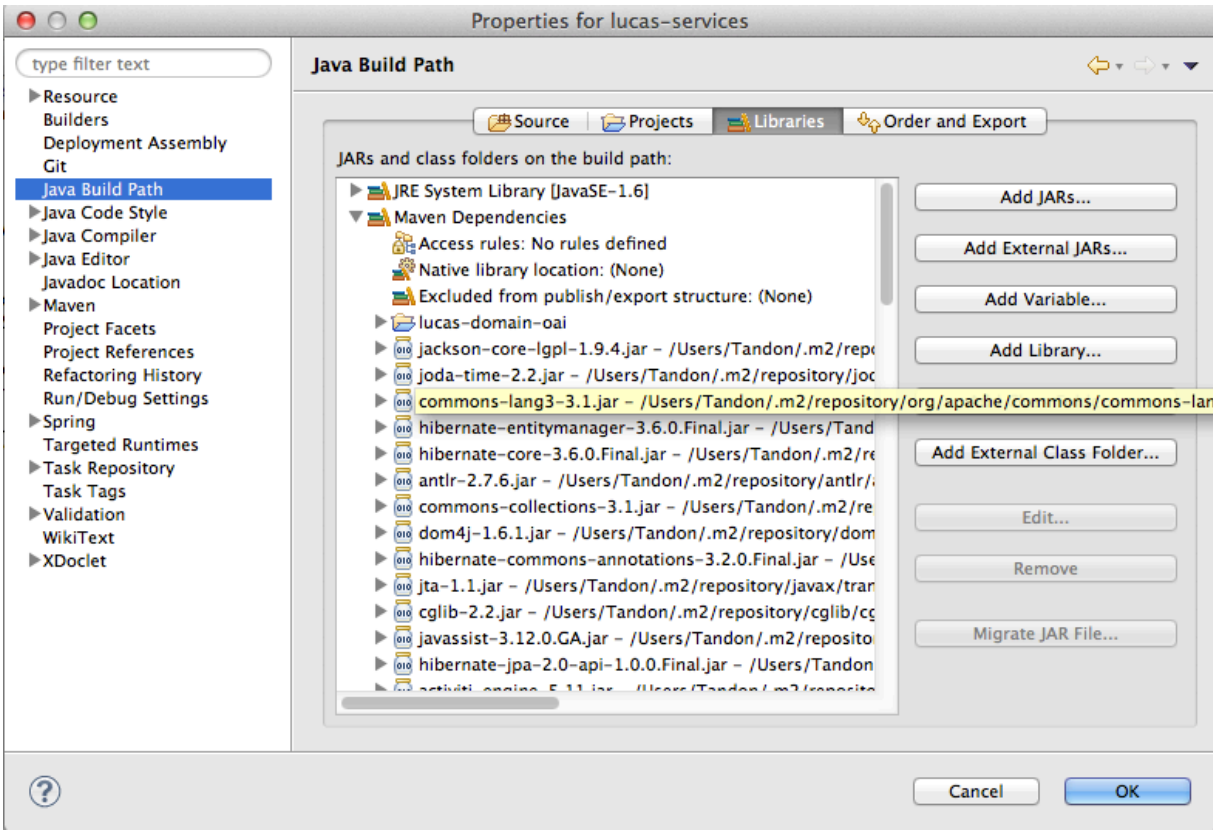
```
cd $LUCAS_HOME/lucas-amd
mvn clean test
```

When you've run the above command, the maven-saga-plugin will run the tests and generate the report in `$LUCAS_HOME/lucas-amd/target/reports/saga/coverage/total-report.html`. When you load up that html page, navigate down to the class that you are testing and ensure that you have **> 80%** test coverage.
Note that this report covers *only* coverage of unit tests, *not* e2e tests. We must strive for 80% coverage via unit tests only.

9. ## Generating Site Documentation

Site Documentation includes static analysis reports, test coverage reports and javadocs amongst others:
cd $LUCAS_HOME
mvn clean site:site -Palltests -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -f lucas-services/pom.xml
Access $LUCAS_HOME/lucas-services/target/site/index.html to view the reports

10. **Pulling your code into Eclipse**
The code checked into the source code repository is independent of any IDE specific files.
Therefore **any** IDE can be used for development. SpringSourceToolSuite (eclipse under the covers) is recommended because it has several plugins that align with the tech stack used by Alps.
Eclipse projects have a .project and .classpath files. These are NOT a part of the code base.
What we have checked out of source control are maven projects, not eclipse ones. Maven promotes consistency in directory structure across most open source projects amongst other things.
Note that a maven project is defined by the existence of a pom.xml whereas an eclipse project is defined by the existence of .project and/or .classpath files. Maven projects can be nested whereas Eclipse projects cannot. Therefore we will have to set up each maven project separately without nesting.
    a. Open up eclipse
    b. File | New | Java Project. Uncheck the Use Default Location checkbox and navigate to the location where you checked out your code.
    c. Select the first directory (should be lucas-amd), and hit "Finish"
    d. Repeat above for each project (lucas-batch, lucas-designer-extensions, through lucas-service-clientxyz or whatever is last)
    e. Then from the Navigator or Project Explorer or Package Explorer views in eclipse, you should see around 10 or so projects.
    f. For each project:
        i. Right click on the project, then | Configure | Maven Project. This adds the Maven nature to the project.
        ii. Right click the project again and then Maven | Update Project. Doing above should allow Eclipse to pull dependencies of your project from the pom.xml.
    g. To confirm above, Right click project | properties | Java Build Path | Libraries should show a node that says 'Maven Dependencies' with content like below:



h. Repeat step e onward for each project.
i. At the end of the above steps, if you navigate down to any *.java file, the errors (red squiglies) should not exit. You can also confirm by looking at the 'Problem' view in Eclipse which should be clean.

11. Running lucas-amd and lucas-api on your local environments

i. Download Tomcat 7.x.
ii. Extract the zip and copy the Tomcat folder to your desired destination (e.g. C:\Program Files\apache-tomcat-7.0.42).
iii. Go to $TOMCAT_HOME/conf/Catalina/localhost (make folders if they don't exist) and make a file called amd.xml with the following content:
(Replace path portion to suit the location on your machine where maven builds the war)

TC context

```
Windows
<Context docBase="C:/<code root>/lucas-amd/src/main/webapp/app" />

Linux or MAC
<Context docBase="/<code root>/lucas-amd/src/main/webapp/app" />
```

Please note that because AMD is developed using Angular, there is no need to deploy a war file to tomcat. Therefore the context points directly to the angular app directory. The other context path is left there for consistency with other (true) webapp modules, where the context will be as below (for example, for lucas-rest-api module):

```
Windows
<Context docBase="C:/<code root>/lucas-api/target/lucas-api" />

Linux or MAC
<Context docBase="/<code root>/lucas-api/target/lucas-api" />
```

iv. Go to $TOMCAT_HOME/bin and add an setenv.sh (or setenv.bat) that has the following settings (Change suitably for Windows):
For Mac

```
export JAVA_OPTS="-Xms500m -Xmx500m -XX:MaxPermSize=256m -server
-Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=8001,suspend=n
-Dlog4j.debug=true
-DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk"
```

For windows

```
set JAVA_OPTS=-Xms500m -Xmx500m -XX:MaxPermSize=256m -server -Xdebug
-Xrunjdwp:server=y,transport=dt_socket,address=8001,suspend=n
-Dlog4j.debug=true
-DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk
```

v. Edit your /etc/hosts file and add the following entry:
127.0.0.1 local-lucas-api.com
The above is done so that you can simulate a cross domain access behavior in your local environment (Remember that lucas-amd and lucas-api are both hosted locally).
Windows users will find this file, most likely in C:\windows\system32\drivers\etc

vi. If you are deploying lucas-amd from the target directory (in step iii above) then the following command needs to be run from $LUCAS_HOME before starting tomcat:
mvn clean install -DskipTests=true -f lucas-parent/pom.xml -Pall

vii. Start Tomcat as you would normally (by running startup.bat or startup.sh from your Tomcat bin directory). A time saver for Mac users could be:

```
#!/bin/shps -eaf | grep tomcat | grep dt_sock | grep 7\.0\.35 | awk
'{print $2}' | xargs kill -9 && ps -eaf | grep tomcat | grep -v grep |
wc
ps -eaf | grep tomcat
./startup.sh && tail -f ../logs/catalina.out
```

Change the grep to suit your tomcat minor version

viii. Browse to localhost:8080/amd to see the home page or to localhost:8080/lucas-api/home for the rest api webapp.

Changing log file location (Linux only)

a. By default, the dev environment is configured to send logs from the amd web app to /tmp/log/lucas-amd.log. and from the lucas-processes project to /tmp/logs/lucas-processes.log.
To override that location, pass the following flag on the cli when building the project

```
-Dlogfile.location.amd=<<AbsolutePathToYourAMDLogFile>>
```

or

```
-Dlogfile.location.processes=<<AbsolutePathToYourProcessesLogFile>>
```

12. Running the Benchmark

This is for creating an assembly that runs the Activiti process engine via a shell script/bat file and invokes a service
It was built to carry out a proof-of-concept with a vendor that creates JVMs for Windows CE.

a. cd $LUCAS_HOME/lucas-processes
b. Ensure that JAVA_HOME is pointing to JRE 6+
c. Issue **mvn package assembly:single -Dmaven.test.skip=true**
This will place a file in LUCAS_HOME/lucas-processes/target called lucas-processes-<<version>>-jar-with-dependencies.tar
d. Untar this file in any location on your machine. Let's call that BENCHMARK_HOME
e. cd $BENCHMARK_HOME/scripts
f. ./run.sh or run.bat depending on linux or windows env. This will output logs to stdout/console. You may have to change the $JAVA_HOME to suit your environment in these script files.

13. Running lucas-batch

Lucas-batch is a war file (Webapp) that can be used to launch jobs on demand, or pre-cofigure them using spring application-contexts.
One job exists called **logPersistJob** is set up to read a file(s) from a location and dump it's contents into a relation db.
This job uses spring-integration inbound-channel-adapter to poll a configured location for incoming file(s).
Another way to launch the job is to use the spring-batch-admin UI that can be used to launch pre-configured jobs.
If a job were to be launched in this manner (as against listening for a file being dropped), then bring up the UI as a web application.

a. Go to lucas-batch/target and deploy batch.war. This can be done inline by pointing your context to lucas-batch/target/batch like has been done in the case of lucas-amd and luacs-api (point 7 above)
b. Navigate to http://localhost:8080/batch and then to jobs and then to the logPersistJob.
c. In the field that asks for JobParameters enter: input.file=file:////C:\path\to\the\actual\file\with\extension
For example: input.file=file:///Users/Tandon/myTemp/batchStuff/testLogFile016.log
d. Hit the Launch button

14. **Maven Profiles**

All these profiles are specified in lucas-parent/pom.xml and should continue to be specified there (as against child modules).

a. The following *module profiles* exist to build different **modules**.
i. **all** - builds all modules including client extensions.
ii. **amd** - only builds the lucas-amd module which requires no other dependency
iii. **restapi** - builds lucas-domain, lucas-services, lucas-rest-api modules
iv. **batch** - builds lucas-domain, lucas-services, lucas-batch modules
v. **process** - builds lucas-domain, lucas-services, lucas-processes and lucas-designer-extensions modules
b. These profiles manage the environments. They are:
i. **loc** - for local development
ii. **dev** - for running as part of CI on Amazon EC2 instances
iii. **qas** - For QA. Not yet implemented
iv. **prd** - For production. Only implemented for lucas-batch
c. There is an **alltests** profile that configures the surefire plugin to invoke functional (long-running) tests
d. There is a **beanstalk** profile to explicitly deploy to a Elastic Beanstalk instance.

Typically you can choose one profile from each of the above categories separated by commas (no spaces).

Examples:
To run the build for a specific project, here is an example:

mvn clean install -f lucas-parent/pom.xml -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -Pbatch,dev,alltests
The above will build the *lucas-batch* project and run *all it's tests* for the *dev* environment.

Similarly:
mvn clean install -f lucas-parent/pom.xml -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -Pamd
Will build *only* lucas-amd and will *only* run unit tests for the loc envirnment (default), whereas
mvn clean install -f lucas-parent/pom.xml -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -Pamd,alltests
Will build and run all unit and functional tests for lucas-amd for the loc environment.

Note that lucas-parent project now serves as an inherited project plus an aggregator (earlier there used to be a lucas-build for aggregation).

Consequently, all modules and profiles are maintained in lucas-parent.

15. Feature Development using the Git Flow Process

This section describes how git should be used during dev, release planning and CI Integration. We will follow the process outlined here.
If you are not using SourceTree, then please follow the cli commands or the commands in any equivalent Git client.
There are some common scenarios that are recorded in a child page called Git Scenarios

a. **For Developers**
   i. **Setup**
      1. Open up SourceTree to point to the lucas repo on your local machine.
   ii. **Starting a new story/feature**
      1. Sync local branches with origin by issuing the command: git fetch
      2. Switch to Develop branch: git checkout develop
      3. Bring local develop branch up-to-date with origin/develop: git rebase origin/develop
      4. Create feature branch: git checkout -b feature/PHX-nnn-short-description-of-ticket
      5. Commands are shown below:

```
$> git fetch
$> git checkout develop
$> git rebase origin/develop
$> git checkout -b feature/PHX-nnn-short-description-of-ticket
```

   iii. **Story/feature development**
      1. Daily git process while working on a story/feature
         a. Make sure that you've rebased any new changes from the repo to your develop branch first (git checkout develop, git fetch, git rebase origin/develop)
         b. Switch to feature branch: git checkout feature/PHX-nnn-short-description-of-ticket
         c. Merge develop into feature branch, and resolve any conflicts: git merge develop or origin/develop
         d. Commands:

```
$> git fetch
$> git checkout feature/PHX-nnn-short-description
$> git merge origin/develop //resolve conflicts
```

      2. Develop your code as necessary
      3. Commit to your feature branch often (recommendation is when you have finished a logical unit of work, and name that commit accordingly. Unlike other Source control systems, it is **not** necessary to prefix all commits with the ticket name (PHX-nnn)).
      4. Create experimental branches off of this feature branch as necessary. Include tests as a part of your feature development
      5. Build and run tests. If you usually run tests from the IDE, please ensure you run tests from the maven cli at least once before pushing.
         mvn clean install -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQIFyby1BAs7ZQVmBm+NRk -Palltests
      6. Once done, **push your feature branch to remote**. Note **push ONLY the feature branch, NOT the develop** branch.
         a. git push origin feature/PHX-nn-a-short-description
      7. Go to bitbucket.org and issue a pull request against the feature branch that you have just pushed. **Ensure that the pull request is issued from your feature branch to lucasware/develop (not master or some other branch).** This is done by selecting the appropriate drop downs on the pull reuest page.

Include the email addresses of team members. Definitely include the tech lead.

8. Other devs need to review all open pull requests and annotate with comments. Please set aside some time (~15 mins) to do this everyday. This is for you to get to know what is going on in other parts of the code base, as well as try to improve code quality.

b. **For Tech leads**

i. **Resolve Pull Requests:** Once a day, have a team meeting no longer than 30 minutes, to discuss pull requests.

ii. **Merge feature branches into develop:** Applicable to those branches whose pull requests have been resolved. There could be merge conflicts but those will need to be resolved as this stage. Tests passing are a good indication of success. Steps:

1. Sync the remote tracking branches with origin
   git fetch
2. Sync the feature branch with remote
   git checkout feature/PHX-nn-a-short-description
3. git rebase origin/feature/PHX-nn-a-short-description //Gets the local feature branch in sync with the remote feature branch
4. Sync the develop branch with remote
   git checkout develop
5. git rebase origin/develop //gets the local develop branch in sync with the remote develop branch
6. Perform the merge
   git merge --no-ff feature/PHX-nn-a-short-description //At this point the feature branch is merged into develop (don't forget the --no-ff, if there's some way to make that default behavior, please share).
7. Make sure all is good, tests pass, after the merge (run all tests, rather than only unit tests at least once before pushing)
   cd $LUCAS_HOME
   mvn clean install -DLUCAS_ENCRYPTION_PASSWORD=Hn4UKcorcdoQlFyby1BAs7ZQVmBm+NRk -f lucas-parent/pom.xml -Pall,alltests
8. If tests pass
   git push origin develop
9. If tests don't pass and for some reason the merge was not good, do the following:
   git log //This will show the last commit SHA before the merge
   git reset --hard <<commit-sha>> //This should restore to commit corresponding to the SHA

iii. **Creating a release:** The release is created spanning several sprints.
Create a release branch off of develop called release-<<major.minor.bug>>. Steps:

1. git checkout -b release-1.1.0 develop
2. Bump up the version in some kind of release/version.txt file to 1.1.1
3. git commit -a -m "Bumped up version to 1.1.1"

16. ## Identifying and Working on a Bug

Often developers, tech leads or qa users/testers will identify a bug (not a feature) that needs to be addressed. Please follow the process below to initiate work on a bug

a. Go to the plan page of the mountainRange agile board.
b. Click "Create Issue"
c. Select the Issue Type as Bug
d. Enter a meaningful summary line
e. Enter a description that preferably has the exception stack/error message. Please try to do this as it will help troubleshooting searches as our code base grows. Also, if possible, add a "How to reproduce bug" section.
f. Assign it to the tech lead
g. Create the issue and note it's key (PXH-nn)
h. Do not move this ticket into the sprint but email the scrum master/tech lead with a link to this ticket. It is his call, if this bug goes into the sprint or not.
i. Once the ticket is moved into the sprint, the Tech Lead/scrum master will assign the ticket appropriately. At that time, pull it off the "ToDo column" into the "In Progress" column, and start work on it. Do not start work till this ticket is moved to the sprint.
j. Please work on it as though it is a feature branch. Except that it is a bug branch. So please create a branch in the format: bug/PHX-nn..... So, same process as outlined for feature development (estimate, work, push branch, log work)

17. ## Dev Process JIRA interaction

This section describes interaction with the JIRA Issue tracking system using the mountainRange agile board.

The statuses that exist for this workflow are shown below with the column mapping that each status maps to.

	Status	Column Mapping
1	Create Issue	ToDo
2	Open	ToDo
3	InProgress	InProgress
4	Reopened	ToDo

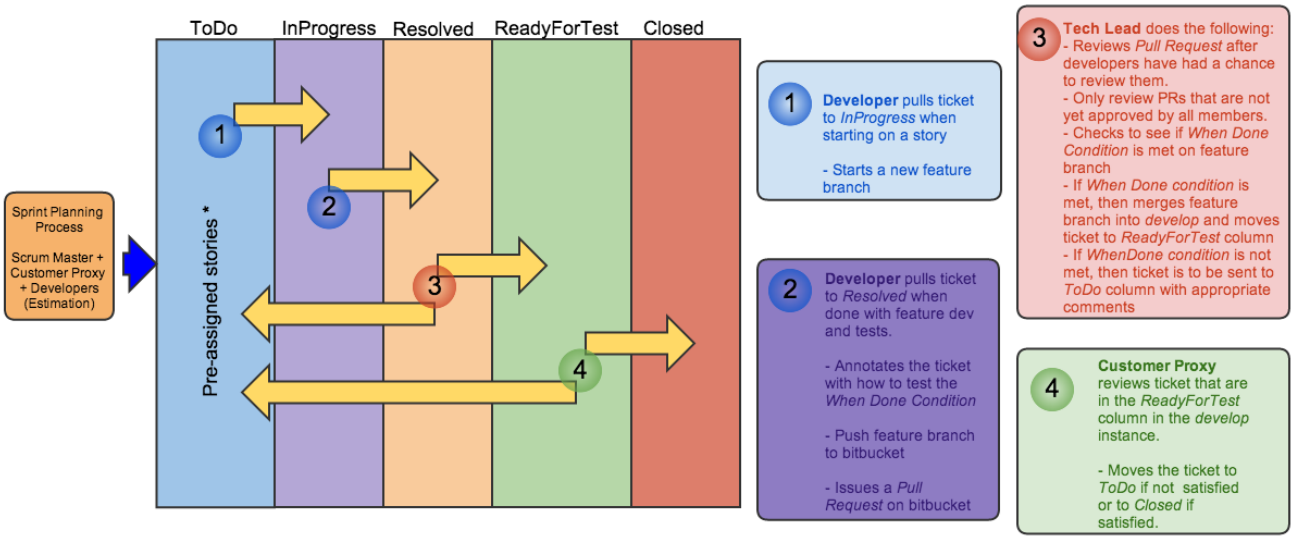
5	Resolved	Resolved
6	ReadyForTest	ReadyForTest
7	Closed	Closed
8	WontFix	Closed
9	Duplicate	Closed

We have also switched to the Greenhopper Scrum IssueType Scheme. The issuesTypes that are supported are:

	IssueType	Purpose and branch mapping
1	Epic	Equates to feature in Release Planning
2	Feature	Equates to feature in Release Planning
3	Story	maps to a feature branch in git
4	Bug	maps to a bug branch in git
5	Technical Task	Is a subtask of a story or an improvement as shown here

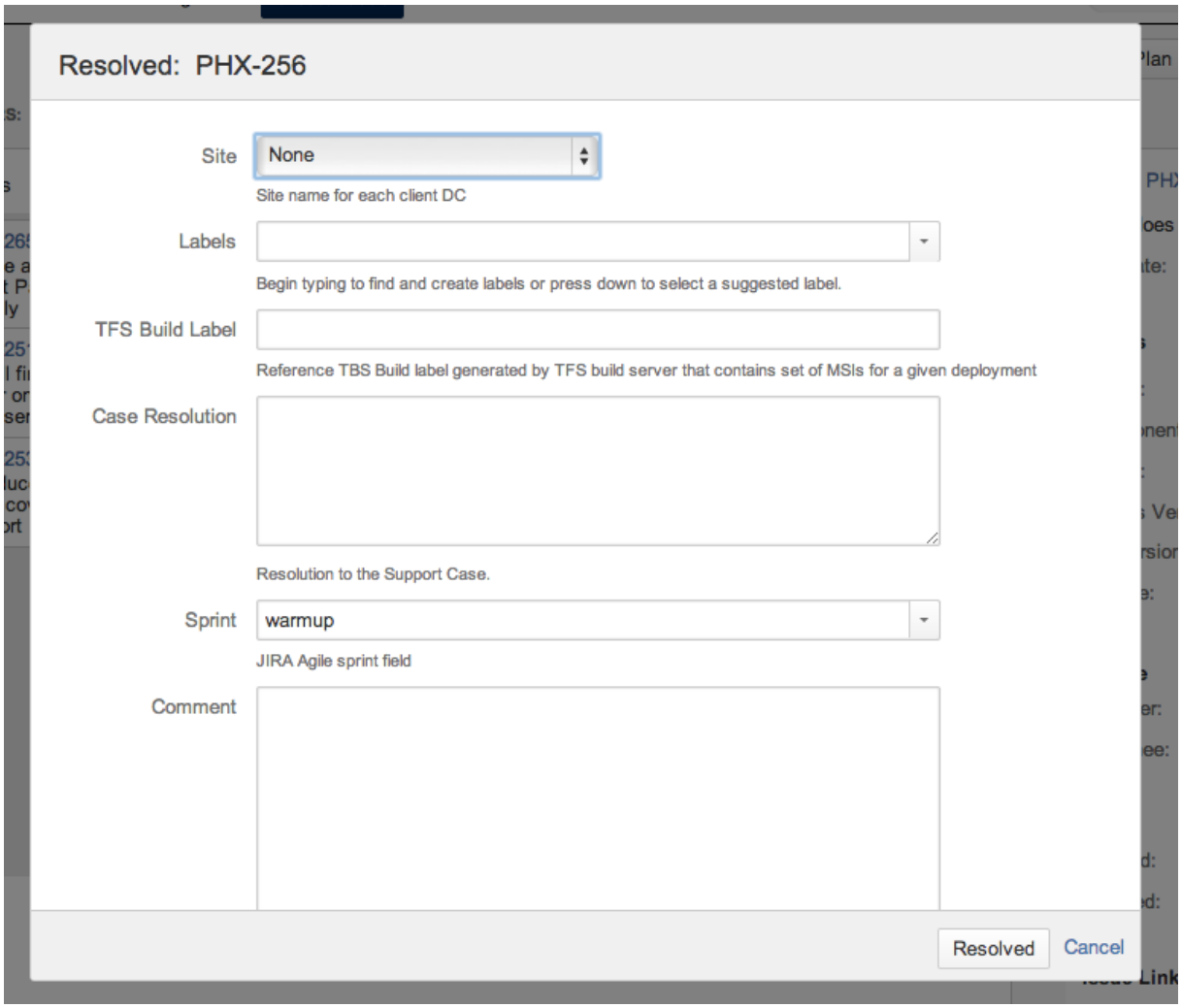
Please see picture here that describes some of the above terms in the context of release management.

The picture below describes the workflow:



a. Once the ticket has made it into the sprint, it should be assigned to a certain developer and should appear in the "ToDo" section of the scrum board.
   Till such time that everyone in the team is a 'T-Shaped Person', tasks will be pre assigned. (More info here).
b. The developer must drag it over to the "InProgress" section when s/he starts work on it. Please note that the "InProgress" section should have it in, as few stories as possible, so that merge issues do not e-merge.
c. When work is done and the developer is satisfied with his work, follow the git process under "Feature Development using the Git Flow Process" to resolve the ticket.

Please ensure that when the ticket is resolved, a comment should be placed on the ticket by the developer, in the pop-up box that appears when the ticket is dragged over to the Resolved column.



Use the Comment field, NOT the Case Resolution field.

Among other pertinent info please indicate how to test the "When Done" condition. If it is not possible to test the ticket result, please mention that too, with a short explanation, if not obvious.

d. The tech lead reviews pull requests and moves ticket to ReadyForTest and merges into develop. The nightly build process deploys the merged code into the develop branch.
e. The business owner/customer-proxy will test all resolved issues in the dev environment (http://lucas-dev-amd.elasticbeanstalk.com) and if satisfied, move the ticket over to "Closed" column. If not, then it will be sent back to the "InProgress" column with appropriate comments. Note to business owner: There is a 24 hour lag between when a story is resolved and when it makes it to the dev environment. (We will manage this manually, for now).
f. Note that in this process, there is always a chance that code that makes it into develop does not represent the tickets that are closed. This will happen when the customer proxy re-opens a ticket that he felt was not resolved (step 4 above), yet the code is already checked into develop. Because this is an incremental development process, these situations will have to manually

monitored and minimized.

16. Using liquibase against the database

a. After you have a database set up locally please follow the steps below to use it (Only for the 'loc' profile, loc for local)
    1. Set up db values as described in "Starting MySQL" above.
    2. cd lucas-services
       mvn liquibase:update -Dliquibase.password=yourLocalDbPassword (Should be password, unless you changed it)
       Note that all other values are specified in the parent pom's loc profile
    3. The above should create all objects as defined in lucas-services/database/sql files.

b. Since liquibase is managing your db, do not change the files that are have already been run against a db or you will get some checksum errors. You can create new ones to alter the behavior of previously run scripts/DDL/DML.
   Sometimes you have to relax that rule. To override, please issue
   mvn liquibase:clearCheckSums liquibase:update

c. To rollback to a certain rollback
   mvn liquibase:rollback -Dliquibase.password=password -Dliquibase.rollbackCount=n
   where n is the number of rollbacks that you want to go back.
   Since mvn liquibase:update will re-run all changes, it is ok to experiment with liquibase:rollback to see how far back you want to go.

d. Sometimes the following scenario may play out:
    i. **Developer1** writes a liquibase script to INSERT rows in table T1. He adds one INSERT statement to insert K1, say.
    ii. **Developer1** runs liquibase scripts against his local db. This records the checksum on this script file on is local db.
    iii. **Developer2** checks out that branch and runs tests (or runs the liquibase scripts). This records checksum on this file in his (Developer2's) db.
    iv. **Developer1** modifies the same liquibase script by adding another INSERT statement (to insert K2, say) in the same script file and checks it into the branch.
    v. When **Developer1** runs liquibase again, it will complain of a bad checksum, as the file was modified. So Dev1 issues a mvn liquibase:clearCheckSums followed by a mvn liquibase:update.
       But since, the db has recorded that the insert script was run, liquibase will silently NOT run the insert script again. At this point, Dev1 can issue a liquibase:rollback to this insert script, or, if the rollback script is not ready (likely, since this is still being developed), he will delete the row corresponding to this script from the DATABASECHANGELOG table and rerun mvn liquibase:update.
    vi. When **Developer2** checks out the branch, and clears checksums, he will either need to do a rollback to this script, or delete the row from the DATABASECHANGELOG table and **then** run liquibase:update.

       The above scenario has used an INSERT script as an example. However, it can happen with any script (table creation, alter table etc).

17. Rules to follow when writing Liquibase scripts:

1. Do not use the IF NOT EXISTS clause in any object that is created via a liquibase script.
Since liquibase guarantees that scripts will be not re-run if they are already run, such a clause results in a false-positive when the script needs to be rerun using a liquibase:update goal of the liquibase maven plugin after deleting a row from the DATABASECHANGELOG table. (The script runs but does nothing because the object already exists)

If liquibase:update needs to be issued to cause liquibase scripts to be re-run, then it must be preceded with a mvn liquibase:rollback -Dliquibase.rollbackCount=<n> so that the corresponding object is deleted via the rollback script and re-created via the "create" script.

2. If something like the Primary Key column of an existing table needs to be changed/renamed, it is better to edit the original script (in which the table was created) instead of creating another script with an ALTER TABLE ... ddl. For all other stuff, like adding or renaming a column to a table, adding a FK or changing the datatype of a column, it makes sense to add a new script to the changelog.

In this case (when the original script is edited) it should be clearly communicated to the tech lead that a rollback to the script that corresponds to this CREATE statement needs to be run so that a subsequent liquibase:update can re-create the object. Secondly, since the checksum on the original file would have changed, a liquibase:clearCheckSums must be issued before liquibase:update

3. For stored procedures, use CREATE OR REPLACE <<sp_name>> in the liquibase script but use the *runOnChange* directive on the changelog.

4. Do not use *runOnChange* or *runAlways* directive on any scripts other than those that are idempotent.

5. For INSERTS include column names in the INSERT. This is because if an ALTER TABLE is issued in a subsequent script, and a column is added, then the previous INSERTs will fail.

6. Typically liquibase script files should include one user story. So it is expected to see an ALTER TABLE, ADD COLUMN followed by INSERT INTO T1 (theNewColumn) VALUES (aValue) in the same script file.