

**LTI Automation Testing Framework**

Version: 0.9

15th May 2020

Contents

[1. LTI QA Automation 3](#_Toc40453344)

[1.1 Agile automation approach 3](#_Toc40453345)

[1.2 Regression Automation 3](#_Toc40453346)

[2. Framework Overview 4](#_Toc40453347)

[2.1 Framework Capabilities 4](#_Toc40453348)

[2.2 Tooling 4](#_Toc40453349)

[3. Automation Script Intake Workflow 5](#_Toc40453350)

[3.1 Automation Feasibility Guidelines 5](#_Toc40453351)

[4. Approach 6](#_Toc40453352)

[4.1 Application Pre-requisites 6](#_Toc40453353)

[4.2 Agile Scripting 6](#_Toc40453354)

[4.3 Waterfall Scripting 7](#_Toc40453355)

[4.4 Generic Guidelines 8](#_Toc40453356)

[4.5 Dry run and Review 8](#_Toc40453357)

[5. Standards 9](#_Toc40453358)

[5.1 Naming Conventions 9](#_Toc40453359)

[5.2 Guidelines 9](#_Toc40453360)

[6. Appendix 10](#_Toc40453361)

[6.1 Documentation: 10](#_Toc40453362)

[6.2 Tooling and Setup: 10](#_Toc40453363)

# LTI QA Automation

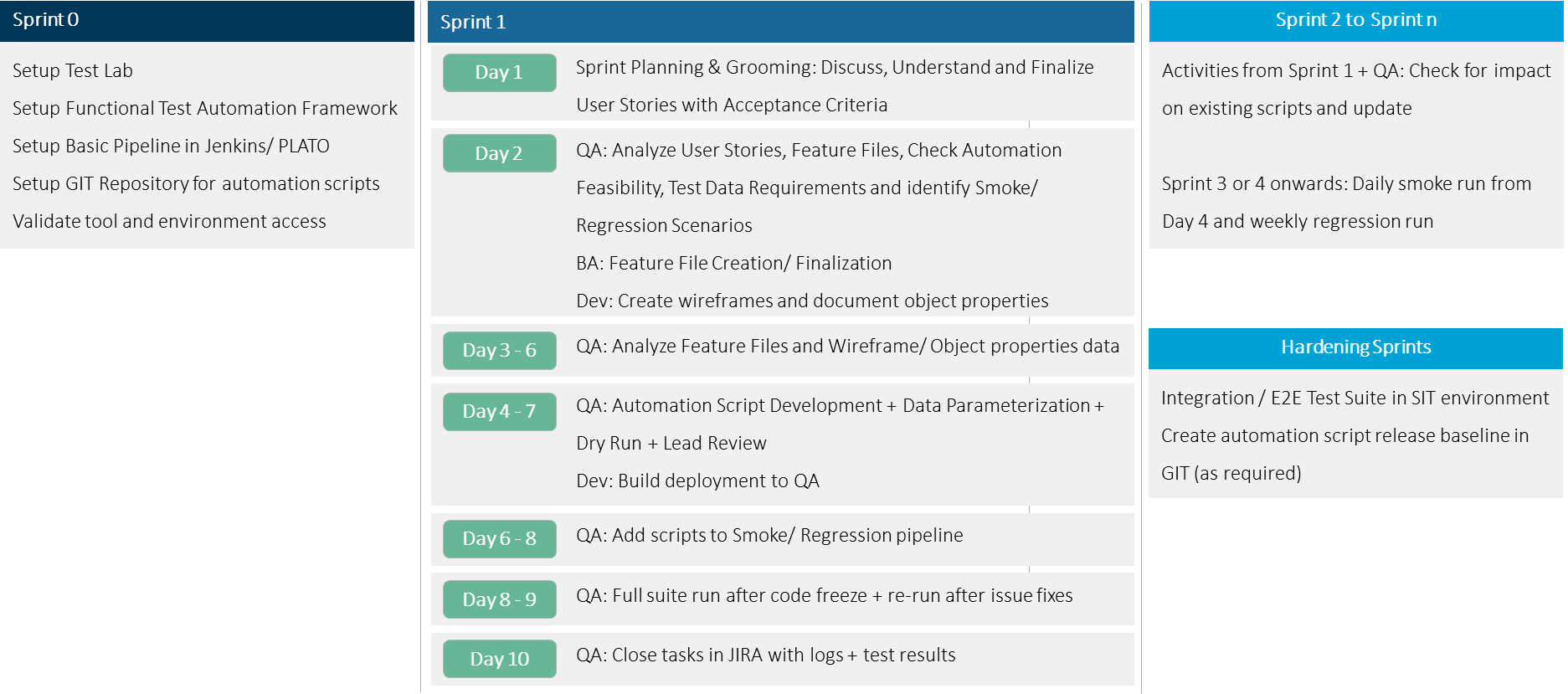
“Automation First” is the approach to all green field agile testing implementations.

This document provides details around the agile and waterfall automation approach, new QA automation framework, guidelines, coding standards and version control.

New framework deployed will cater to all green field implementations delivered in Agile and waterfall mode. Agile automation tooling, approach is aligned to achieve in-sprint automation.

## Agile automation approach

LTI envisioned 2 weeks sprint for Unicorn program. We have aligned all automation activities to align with overall program delivery structure and objectives. Below picture depicts the high-level automation activities at sprint level. Note that the individual implementations in work streams may vary and we could customize the approach to suite respective work streams.



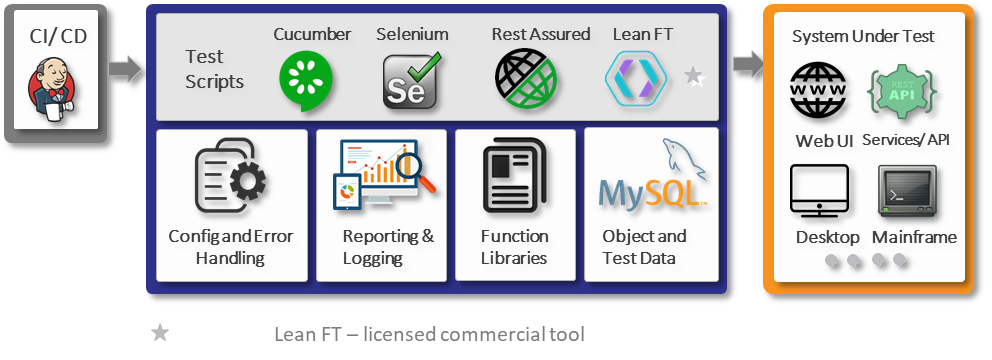
*Picture 1: Agile Automation Activities*

## Regression Automation

* **Sprint level:** select regression for the features updated/ modified during the current sprint should be part of sprint regression. All/ select scripts from these features need to be run at the end of the sprint. Recommendation is to run on Day 9.
* **Release level**: all automation scripts developed till date for the application (s)/ select core regression based on concurrence from the product owner should be run as part of release regression.

Note: tag the JIRA scripts with appropriate tags i.e. automation, regression etc. Create separate JIRA ‘Test Execution’ for regression runs for better tracking.

# Framework Overview

QA Automation framework is built to suite the agile way of delivery and uses the behavioral driven development concepts to achieve in-sprint automation. In-sprint automation is enabled by feature file driven requirement capture. Below is the framework component depiction: 

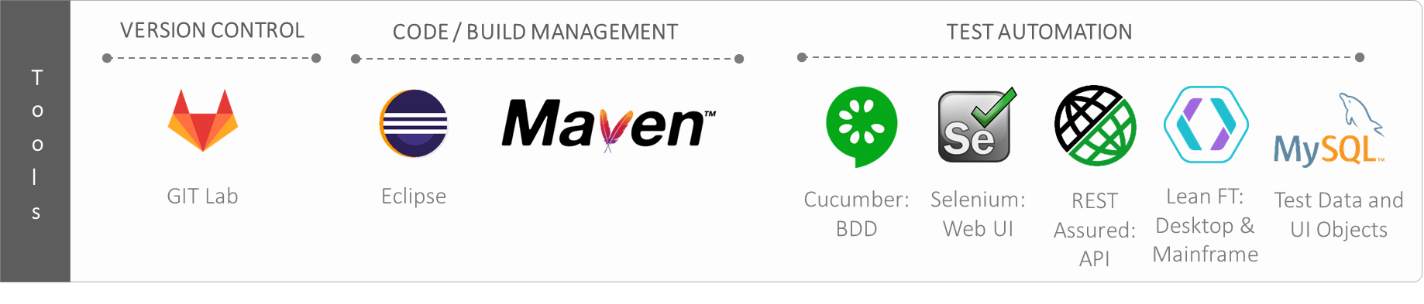
*Picture 2: QA Automation framework overview*

## Framework Capabilities

* Supports automation from Web UI, Web Services (REST), Desktop Apps, and Mainframe terminal applications through integration of various open-source and commercial tools.
* Easier Maintenance through modular approach:
  + Leverages centralized DB to store all objects and data.
  + Page object model (POM) Inspired step definitions.
* Faster and reliable with built-in re-usable functions to handle all generic actions including error handling.
* Quick script development using feature files – step definitions.
* Extensible: can be extended to support mobile platforms and cloud-based browser/ device labs
* CI/ CD: **Orchestration through Jenkins**

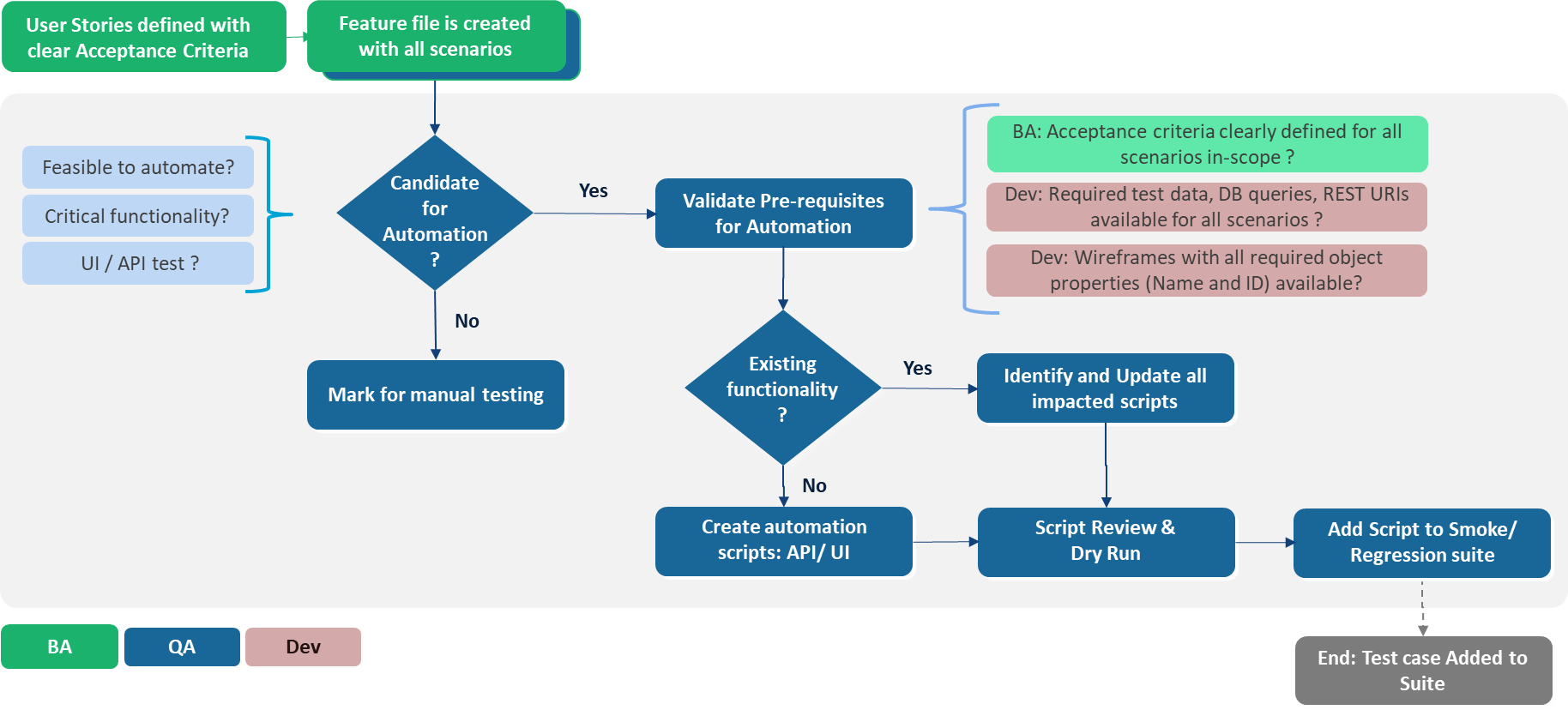
## Tooling

Below is list of tooling used in the current framework:

*Picture 3: QA Automation framework tooling*

# Automation Script Intake Workflow

Feature files will be the single source of requirements for all automation scripts. Below workflow depicts the overall flow for scripts intake:



*Picture 4: Automation script intake workflow*

## Automation Feasibility Guidelines

### Automate:

* More frequently run tests: System Integration, Regression, and Smoke scenarios
* Scenarios with more data variations e.g. APIs with data validation

### Examples of not feasible scenarios:

* CAPTCHA Scenarios
* Usability and visual validations e.g. UI layouts, print and multi-media (video clips)
* Hardware testing and interactions e.g. biometric scanner, card swiping, QR code reading, barcode reading
* UI Objects where no unique properties exist OR Custom widgets not supported by the tools
* Complex integrations where data flows through many systems and some systems are not feasible to automate
* Certain PDF validations

### Do not automate for:

* Tests which are run less than 5 times in entire lifecycle – Focus on most frequently used tests
* Functionalities which are expected to change drastically in near future e.g. UI tests where UI is expected to change.

# Approach

## Application Pre-requisites

* Feature files with scenarios and detailed acceptance criteria for Agile are approved and ready for QA consumption.
* Approved manual test cases for waterfall projects
* Test Environment access [Target URLs, Credentials for each user role]
* Target URLs should be accessible from automation tester machines
* For desktop applications (.NET, Java) – application client setup should be locally available from automation tester machine and application client should be able to connect to the app server
* Wire-frames and unique object properties [ID, Name]
* REST API:
  + API URIs and resource details
  + API Documentation covering authentication, available resources, implemented operations, use case flows, sample JSON files etc.
  + URI end points should be accessible from automation tester machines
  + If end points are HTTPS, ensure you have the relevant SSL certificates

## Agile Scripting

### Create Step Definitions:

* *‘Application.properties’* file will have application URL, User ID and Password
* Open Eclipse and refresh the codebase to get the latest updates
* Create the feature files under respective work stream folder: *src/test/resources/<workstream>/<application>/features*
* Add respective tags to:
  + Feature: tag for fetching test data: @td\_<Feature ID>
  + Scenario: use any of the following standard tags as appropriate: *@StreamShortName\_ApplicationShortName\_Sanity – e.g. @STR\_APP\_Santity*

*@StreamShortName\_ApplicationShortName\_Regression*

*@StreamShortName\_ApplicationShortName\_Web*

*@StreamShortName\_ApplicationShortName\_API*

*@StreamShortName\_ApplicationShortName\_Desktop*

*@StreamShortName\_ApplicationShortName\_Mainframe*

*@StreamShortName\_ApplicationShortName\_Sprint<number>*

* Create step definitions for the feature file: *src/test/java/qa.projectname.<workstream>.<application>.stepdefs*
* Create step definitions at page level for UI tests and API level for API tests.
* For UI tests, page name should be Page ID from PAGES table
* Step definitions must go to respective page/ API level step definition files
* Check all existing step definitions to avoid creating duplicate step definitions

### Write code in Step Definitions:

* Instantiate Action class with relevant parameters for each page (for UI)
* Write code in respective page classes and call the respective methods in step definitions to achieve the step objectives
* Add UI elements in the data base. e.g. DB Name: unicorn\_qa\_db, Table Name: ELEMENTS

|Page ID| Element Name | Value| Value Type |

* + Unique key is combination of Page ID and Element Name
  + For value type, please refer to ‘Value Type’ table
* Add test data required in the database. e.g. DB Name: unicorn\_qa\_db, Table Name: TESTDATA

| Feature ID | Test Data Key | Test Data Value |

* + Unique key is combination of Feature ID and Test Data Name
* For API: store the test data – request and response JSON files under resources folder: *src/test/resources/<work stream>/<application>/<API – service name>*

Store the request and response JSON files in separate folders:

*src/test/resources/<work stream>/<application>/<API – service name>/request*

*src/test/resources/<work stream>/<application>/<API – service name>/response*

* Static data like column names should be in feature files, use step level data table - with ‘|’ separated values for multiple values

## Waterfall Scripting

Unicorn has some projects running in waterfall methodology where manual test cases are available or newly created. Automation framework has the capability to automation test cases as well, where feature files are not available.

Test NG – Page object model is integrated into existing framework to support test case automation.

### Scripting

* Create pages for each UI page and write all methods related to the page in this class
* Add UI elements in the data base e.g. DB Name: unicorn\_qa\_db, Table Name: ELEMENTS

|Page ID| Element Name | Value| Value Type |

* + Unique key is combination of Page ID and Element Name
  + For value type, please refer to ‘Value Type’ table
* Create tests at module/ functionality level and create all tests *(@Test)* under this class.
  + This class should extend *TestNGBase* class.

Note: for mainframe automation, extend *MainframeTestNGBase* class.

* + Ensure extent test is created for each test

*test = extent.createTest("<Test Case Name>");  
GlobalVariables.test = test*

* + Reporting steps should be added for each action and verification step

*test.log(Status.PASS, "<verification message>");*

Note: report logging for failure is handled in *TestNGBase* class.

* *testng.xml* should be created for execution using framework class – *CreateTestNGSuite* at runtime.
* All functionality related code should be written in respective page classes. Test should only contain calls to these methods, assertion and reporting steps.

Note: for mainframe we will have screens instead of pages

* For Mainframe, add fields with id and NOT with x, y positions

## Generic Guidelines

* Use Action class from the framework to perform any re-usable action e.g. click, select, set
* Check for available re-usable functions from the framework library before writing any new code
* Page Name in ELEMENTS table should be page title
* Pages will be added to PAGES table by work stream leads and applications by framework team
* Use TESTDATA table to parameterize all the dynamic data
* API request/ response data should be in excel
* Add/ Edit data in DB using Heidi SQL interface
* Apply required filters before adding/ editing any data e.g. Page level filter in ELEMENTS table and Feature level filter in TESTDATA table
* Use standard naming convention for Workstream and Application Names. Workstream Legend Below:
  + <provide application list with designated short names here>
* Write detailed comments and write Java Doc for all methods
* Do not touch any files under framework folder: *src/main/java/qa. framework*

## Dry run and Review

### Dry Run:

* Save all code files and fix any compile errors
* Set the configuration and environment variables for the current run
* Ensure all required object properties and test data is updated
* Run the script (s) and fix any run-time errors
* Validate the report generated shows all required steps and validation points

### Stream Lead Review:

* Review the code base with Stream Lead
* Stream Lead Validation:
  + Code written follows the defined navigation and actions in the Feature Files
  + All validation points are covered as per the acceptance criteria
  + Elements and Step definitions are not duplicated
  + All dynamic data is parameterized and is read from respective Feature level test data
  + Run results report contains all the required steps and validation points
  + Coding standards are followed including comments
  + Appropriate tags are used e.g. *@STR\_APP\_Sanity, @STR\_APP\_Regression, @STR\_APP\_Sprint<number>*
  + JIRA Test ID is populated in feature file against scenario and in examples for scenario outlines.
  + JIRA test is marked as automation test. Regression tests are marked with appropriate JIRA tag.
* Fix any review comments and commit the code to GIT Dev branch

# Standards

## Naming Conventions

* Package Name: all lower case, no underscores e.g. qa.framework
* Class Name: Upper CamelCase - start with upper case then start with upper case for every inner word e.g. TempEmployee
* Method Name: lower camelCase - start with lower case then start with upper case for every inner word e.g. setValue
* Variable and Parameter Names: lower camelCase - start with lower case then start with upper case for every inner word e.g. customerName
* Constants: use all capital letters. Separate multiple words with underscore: \_

## Guidelines

* Refer to GIT Lab documentation for code commit guidelines
* Folder Structure of the framework, database table columns and scripting guidelines are detailed in respective sections.

# Appendix

## Documentation:

Following documentation is available in framework at: *src/test/resources/docs*

* GIT Lab documentation
* Eclipse proxy setup for Maven
* Feature file writing guideline

## Tooling and Setup:

* Environment Variables should be set up JAVA\_HOME, MAVEN\_HOME, % JAVA\_HOME %\bin and % MAVEN\_HOME %\bin
* TestNG (http://beust.com/eclipse/) and Cucumber Plugin for Eclipse (http://cucumber.github.com/cucumber-eclipse/update-site) should be installed in Eclipse using Help->Install New Software.
* GIT Bash setup in Broadridge VDI
* GIT Access with check-in, check-out permissions for right framework folders
* JDK 8, Eclipse 2019-3, Lean FT 14.5x, Heidi SQL setup is available in Broadridge VDI
* Configure Maven Installations and User Settings in eclipse under Windows ->Preference. Maven Installations should have maven path and User Settings should be path of setting.xml file. Note: setting.xml file should have m2 repository mirror pointing to Nexus repository.
* Configure Installed JRE in eclipse under Windows ->Preference. Windows ->Preference > Java ->Installed JRE ->Add->Standards VM -> select JDK path.
* Proxy setup in eclipse to ensure Maven build works [Refer to appendix for details]
* For Desktop/ Mainframe applications:
  + Lean FT 14.5x should be setup
  + Copy Lean FT drop-ins from *C:\Program Files (x86)\Micro Focus\LeanFT\IDE\Eclipse\dropins* to *<eclipse installation path>\eclipse\dropins*
  + Run *Lean FT Runtime engine* before using Lean FT
  + Run *C:\Program Files (x86)\Micro Focus\LeanFT\bin\TeConfigurator.exe* to configure Lean FT to use IBM PCOM emulator configuration - for mainframe green screen testing.
  + *Terminal Emulators* add-in should enabled in Lean FT settings
* Get the latest code base from the Dev branch to your local folder – refer to GIT documentation.
* Verify the code base by opening the project from Eclipse