1 Smartesting CertifyIt with UFC Security plugin

General information	
Name	Smartesting CertifyIt
Provider	Smartesting
Topic addressed	Model-Based Testing solution
Description	The tool is composed by a Rational Software Architect plugin for modelling activities, a standalone Java application for the test generation and test case management, and dedicated RASEN plugin to manage DASTML specifications, vulnerability test generation from test purposes and RASEN test publisher.
License	RASEN project partner can obtain a license for the project duration.
Website	www.smartesting.com
Technical information	
Download site	www.smartesting.com
OS	Win or Linux
Technology environment	Rational Software Architect v8.0.x and v8.5.x, EMF compliant JAVA 1.6, 1.7 compliant
Other dependencies	N/A
Additional information	
Known issues/risks	N/A
Additional useful information	N/A

2 Security plugins for Smartesting CertifyIt

This section introduces the Security plugins developed by UFC for the tool CertifyIt provided by the company Smartesting. It enables to generate security test cases from vulnerability risk assessment, and to produce test results, which can actively contribute to identify, estimate and finally treat the risk of the tested application

It should be noted that a more detailed presentation of the tool features including tutorial, and tips to use it efficiently, are provided with the tool (modeling guide, UML/OCL reference guide and software documentation).

2.1 Presentation

Smartesting CertifyIt is a tool suite that automatically generates test cases based on a model of system requirements. Manual test design is labor intensive and error prone; this manual work can be avoided for complex applications by modeling the key concepts (abstraction) and allowing Smartesting CertifyIt to automate your test design work. Since the model is more expressive and simpler than the system-under-test, it can more readily be reviewed for correctness and coherency, as well as be updated more easily. Some plugins have been developed during RASEN project for the deployment of the RASEN approach in order to assess its accuracy and precision regarding risk-based objectives. Smartesting CertifyIt including these plugins supports UML/OCL models as the specification modeling language, and generates test cases to cover security test patterns used as test objectives. Finally, the generated test cases can be exported in UML sequence diagrams and can be fully integrated into the RASEN Security Dashboard introduced in section **Error! Reference source not found.**.

More precisely, the Smartesting plugins developed by UFC concern the following features:

1. Test Purpose language extension, enabling sets creation using OCL code evaluated on the initial state of the system
2. Keyword creation to be used by Test Purposes. This mechanism enables to create generic Test Purposes, and to help for maintenance and reuse.
3. Capability to link a Test Purpose to a requirement identifier to ensure the traceability through the all test generation process.
4. Test Purpose catalogue import/export to reuse and apply Test Purposes on several systems under test.
5. Generation and Animation standalone Java API to enable fuzzed test sequences validation regarding the model.
6. A DSML allows the validation engineer to easily create the UML model. This DSML specifically addresses web-based applications.
7. A RASEN dedicated Wizard assist the user during the model creation, and pattern selection regarding the Risk analysis.
8. A RASEN publisher makes it possible to export the test result into the RASEN Security Dashboard.

Those features are presented in a more detailed way in the D4.2.3 RASEN deliverable.

2.2 Installation Guidelines

Smartesting test generation solution works with models edited by an Eclipse-based UML modeler. Smartesting provides a plug-in that checks whether the model fits the Smartesting CertifyIt restrictions on UML, and exports a model to be used by the Smartesting CertifyIt tool to generate the test cases. To use Smartesting test generation solution, both the Eclipse-based modeler plug-in for IBM RSA and the Smartesting CertifyIt software are needed.

IBM RSA Modeler Plug-in installation

1. This plug-in must be installed with the Eclipse update site tool. The following steps describe the installation of a new release.
2. Start RSA modeling tool.
3. From the tool menu, select Help->Software Updates->Find and Install...
4. A dialog opens. Select "Search for new features to install", and click "Next". Select "New Archived Site ...".
5. Specify the path to a .zip file available in the 'install' folder of the Smartesting CertifyIt installation directory. Once the path is selected, the following window should appear. You can use the 'Name' field in order to define your own Local Site name.
6. Click the "Finish" button of the "Install" window to launch the installation.
7. Check the Smartesting CertifyIt plug-in, and click « Next ».

   At this stage, you may have warnings indicating that some features cannot be installed (due to unavailable dependencies). In that case, check the “System Requirements and Supported Software”, and upgrade RSA modeler if required.
8. Accept terms of the license agreement, and click « Next ».
9. Click the « Finish » button to install the plug-in.

If verification messages appear, just accept those in order to complete the installation. It should be noted that write permissions for the install location are needed. Afterwards, restart RSA modeling workbench. The Smartesting CertifyIt plug-in should be successfully installed at this point.

Smartesting CertifyIt software with Security plugin installation

To install Smartesting CertifyIt software and the dedicated Security plugins, simply run the setup program and follow the installation instructions (this implies accepting the terms of the displayed license agreement).

Before using Smartesting CertifyIt, a license needs to be defined. Prerequisites are: Smartesting CertifyIt must be installed and the RSA Eclipse-based modeler plug-in must be installed. There are two types of licenses: floating license server or capacity-based license.

2.3 User Guide

This section introduces each step of the testing process supported by the tool.

2.3.1 Modeling Activity

The first action in the process is Web application modeling. A domain-specific modeling language called DASTML enables to create textual UML models. Therefore, any text editor can be used for model design. Designing a DASTML model does not require any testing skills, and consists of doing a manual crawling of the Web Application under test to gather relevant information that will be used for vulnerability detection purposes.

The first step of this manual process should be about creating a “map” of the Web application, that is to say the page flow. This can be done by clicking on links and filling Web forms, the goal is to identify each unique page that requires testing (e.g., because it provides user interactions that are not purely navigational) or would be part of an attack process (e.g., a result page).

Then, the second step is to do reconnaissance on each page that has been modeled to gather information in order to design a DASTML model, in a text file. Once the model is complete, or a first subset of the Web application has been modeled, the file can be processed by RSA to translate its content in a UML test model.

Figure 1 shows the RSA interface, which has been has been extended with several plugins for testing purposes.

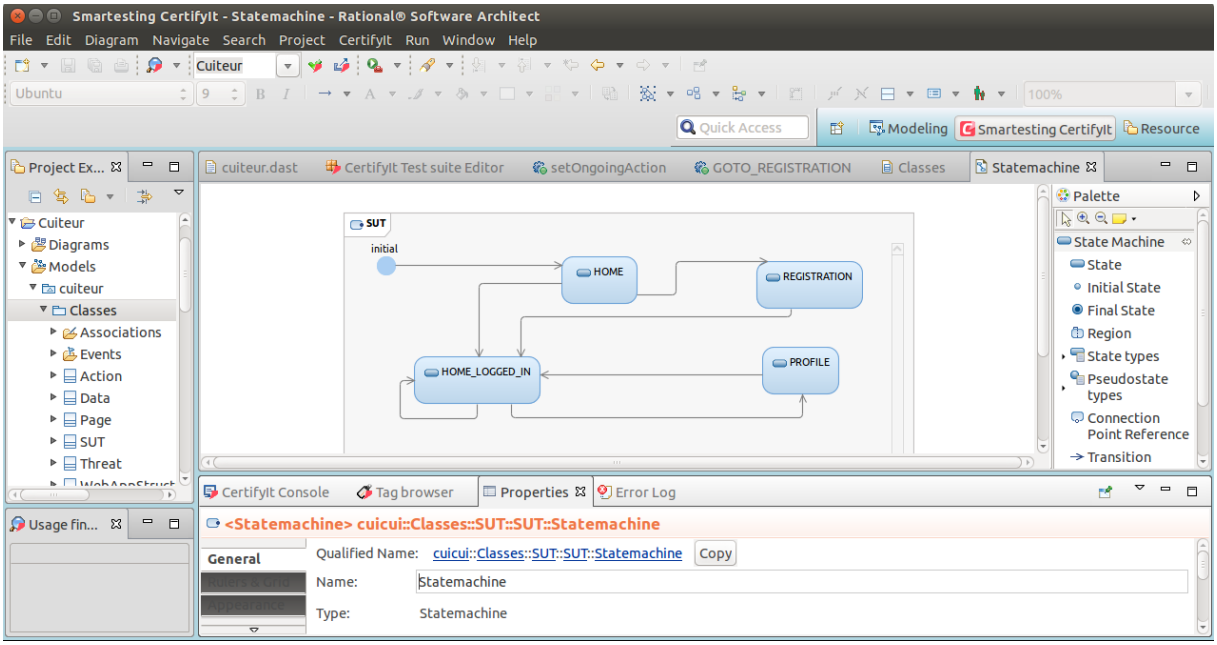

Figure 1 – IBM RSA Modeling Environment

The first plugin, Smartesting for RSA, enables to create UML test models, animate them, check their compliancy with the UML metamodel, and export them in a standardized XML file. It is used to manage generated UML entities and potentially make adjustments in special cases where experimented test engineers need to create new test purposes. It is also possible to animate models to check whether they are accurate regarding the Web application under test.

The second plugin, Test Purposes connector for RSA, provides a graphical interface to manage test purposes: creation, import / export, validity check. It is more discussed in the next section.

The third plugin, PMVT Connector for RSA, is responsible for the translation of DASTML models in UML test models. DASTML models are parsed using an ANTLR3 grammar, which creates an Abstract Syntax Tree (AST) out of it. Then, a dedicated algorithm visits the AST to create the corresponding UML entities on the fly. This is delivered to users of the solution in two ways.

First, test engineers can choose to create a new modeling project. An eclipse wizard has been created and provides two inputs: a DASTML model file, and a test purpose catalog. From these inputs, the wizards automatically creates the corresponding UML test models, and integrates the test purposes from the catalog. Second, it is possible to import a DASTML model in an existing project. However, model evolution is not yet supported at this point and test engineers must remove any previously generated entity before importing a new DASTML model.

Next step of the process is the selection and / or creation of test purposes, as described in the next section.

2.3.2 Test Purpose Activity

Once the model has been designed, test engineers have to proceed to test selection. This is done using test purposes. The design and selection of test purposes is also done through RSA. A graphical interface has been created, which is depicted in Figure 2. The upper part of the interface concerns keywords lists. Such list can contain a set of Enumeration literals, instances, states, behaviors, etc. Lists content is then used by test purposes iterators to compute test cases based on a refined set of model elements, therefore allowing test engineers to generate precise test suites that only address the selected elements. Note that all the visible keywords lists visible in the figure have been generated automatically.

The lower part of the interface is the test purpose editor. The left side lists all the created or imported test purposes, and on the right side is the actual editor. It is shipped with test purposes and OCL expressions on-the-fly syntax check.

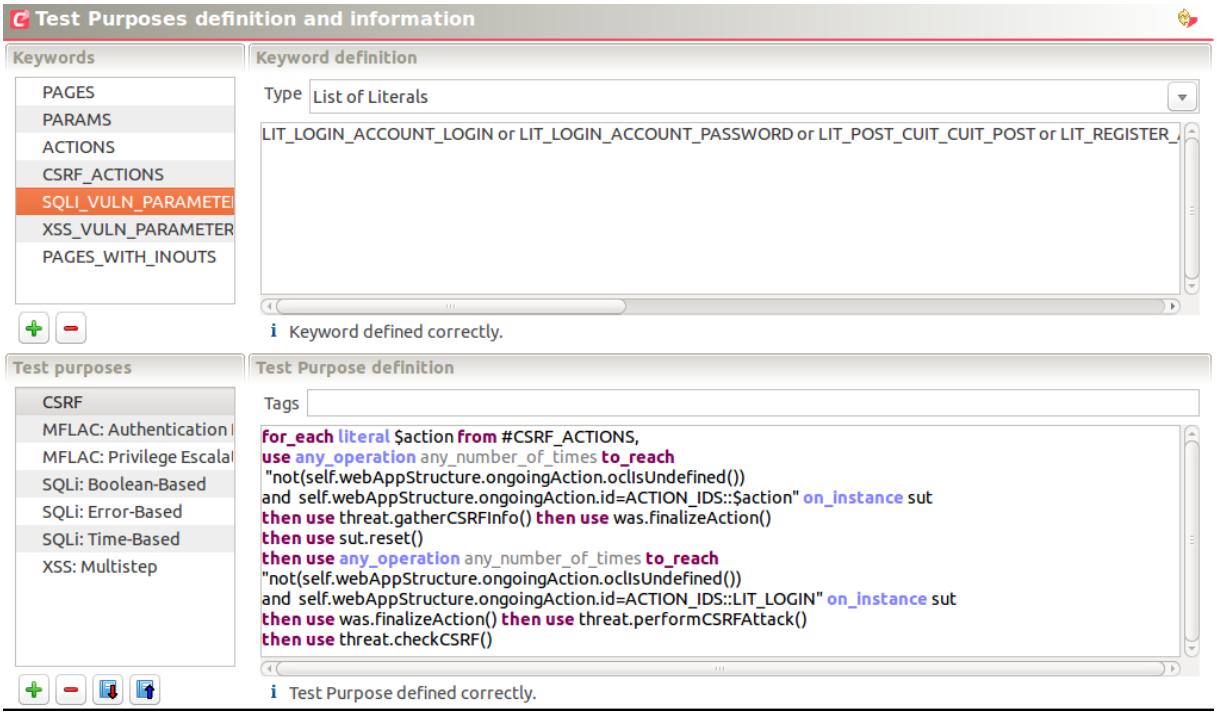

Figure 2 – Test Purpose Editor

Users can add and remove test purposes, and export their test purposes as a catalog in an XML file that is linked with CORAS. This way, they can directly import them from risk assessment. For that, we provide to users of the RASEN approach a predefined catalog containing the existing test purposes. Users can import all or part of this catalog. The test purposes stored this catalog are usable as is to test any Web application, without the need for adaptation. Figure 3 shows the interface for importing test purposes from the catalog in a test project.

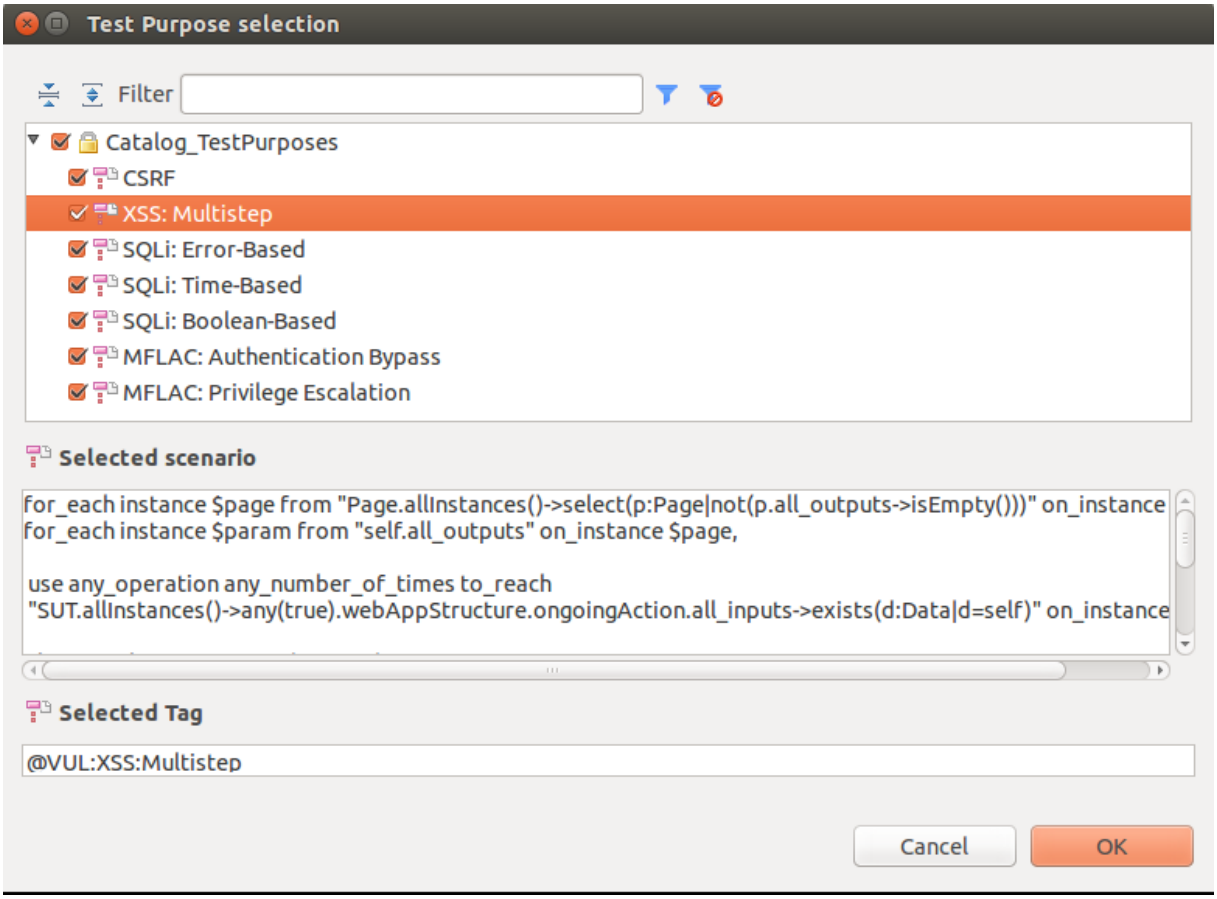

Figure 3 – Test Purpose Catalog

When the model and the test purposes have been defined, Test engineers can check the validity of their project. If it has been successfully checked, the next step is to export this project in a standardized XML file and pass it on to the test generation engine.

2.3.3 Test Generation Activity

The test generation engine that is used as part of the process is supported by the tool CertifyIt provided by Smartesting. This tool generates abstract test cases by finding paths in the test model to reach the test targets that have been derived from the test purposes. Figure 4 shows its interface.

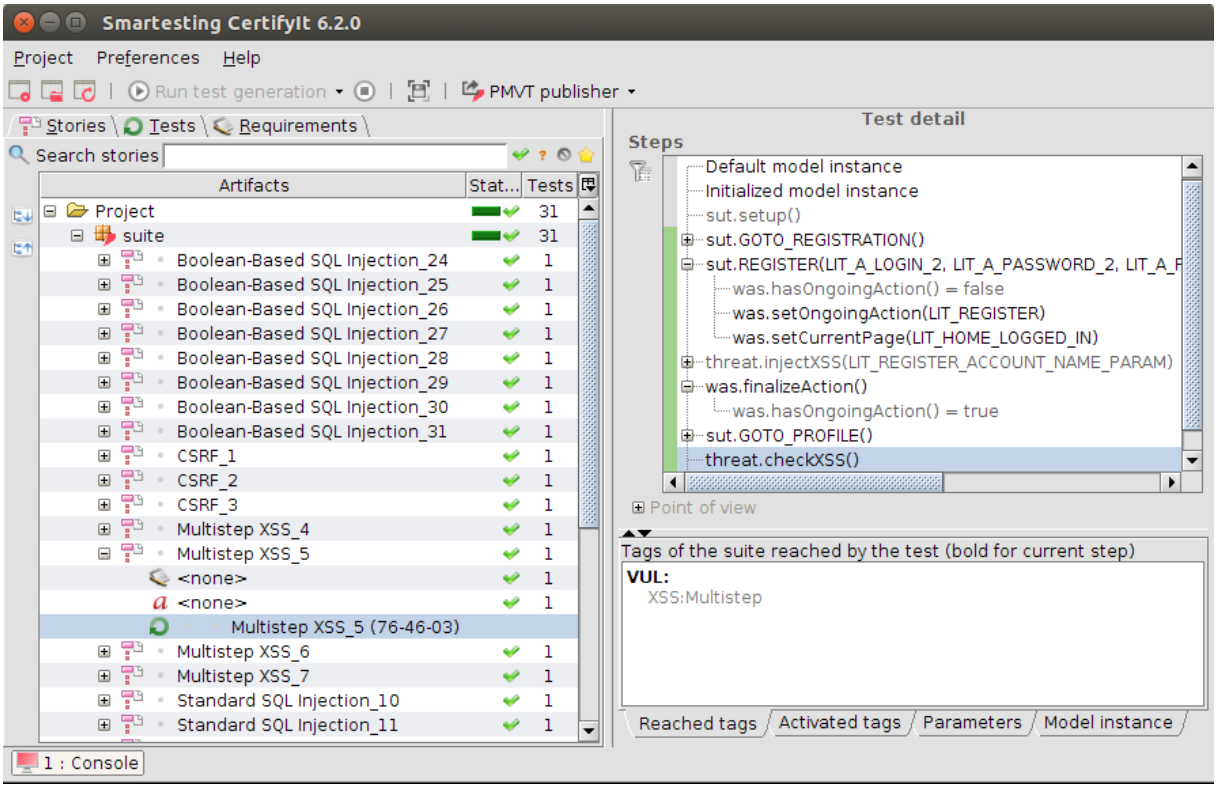

Figure 4 – Test Generation Environment

The left part lists all the test targets that the tool extracted from the model. If a test target is successfully reached, a test case is produced and all its steps are displayed on the right part of the tool.

CertifyIt interprets test purposes using a two-step process. First, during the importation of a test project, the test generation unfolds the test purposes: if a test purpose contains iterators, then the test generation engine creates a test target for each element of the list linked to the iterator. Second, the test generation engine tries to reach each test target by animating the model. When test generation is completed, test engineers must export the generated abstract test cases in executable test scripts, which are introduced in the next section.

2.3.4 Test Concretization and Execution Activities

Abstract test cases are not executable as is on the real system, therefore they must be concretized (adapted) and exported as test scripts. Test engineers must implement each operation from the model, provide concrete data to match each abstract data, and may need to develop a test harness in order to simplify test execution.

In the context of RASEN, generated abstract test cases are exported as JUnit test scripts. These scripts allows to automatically interact with the Web application through its GUI, as a normal user would do. To accomplish this programmatically, test scripts use libraries such as Selenium[1] or HTMLUnit[2], which enable to operate a headless browser using primitives (i.e. methods such as filling form fields and clicking buttons / anchors).

Test concretization activity represents a considerable amount of work to obtain executable test scripts. A dedicated test publisher has therefore been designed in order to automate most of the process. When all abstract data have been matched with concrete data and all methods have been implemented, the obtained test scripts can be executed on the real Web application.

Test execution and verdict assignment are fully automated, thanks to the test harness generated by the publisher. Test engineers execute the test suite, and collect the results.

---

[1] http://www.seleniumhq.org/ [Last visited: August 2015]
[2] http://htmlunit.sourceforge.net/ [Last visited: August 2015]

Both the test concretization and execution activities are performed preferably using an IDE (in our case, IntelliJ Idea), as depicted in Figure 5. Such tool provides a graphical view of the test results, and an "export" function to store the results, for instance in an XML file. For instance, a RASEN exporter allows engineers to produced a XML files including test cases and results that can be imported into the RASEN Security Dashboard presented in section **Error! Reference source not found.**.

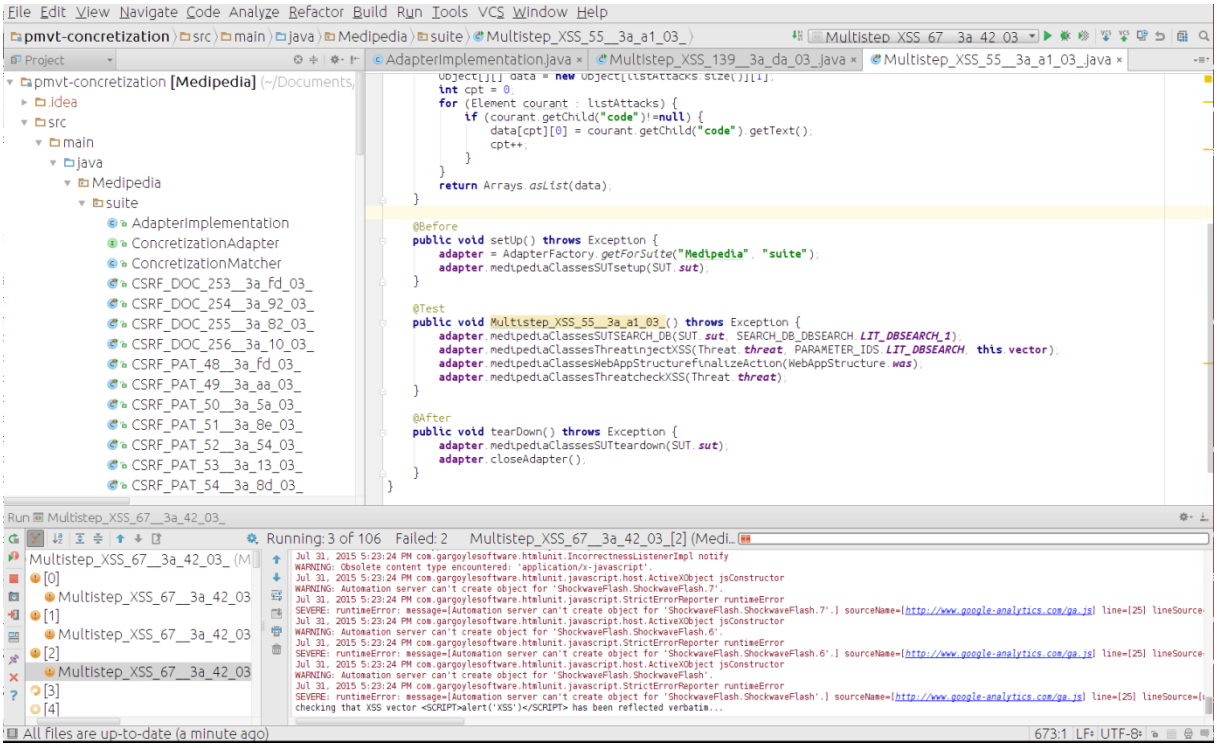

Figure 5 – Test Concretization and Execution Environment