Automated Acceptance Tests

Evaluation of some available Frameworks

- **Cucumber** http://cukes.info/
- **jbehave** http://jbehave.org/
- **Spoke Framework** http://jbehave.org/
- **Concordion** http://concordion.org/

Introduction

Automated acceptance tests is a kind of a more general concept called *Behavior-driven development*. **BDD** revolves around the concept of a **Story**, which represents an automatically executable increment or update of business functionality. At its core a Story comprises of one or more **Scenarios**, each of which represents a concrete example of the behavior of the system. Each Scenario comprises of a number of executable **steps**:

```
==========================================================================
A story is a collection of scenarios

Narrative:
In order to communicate effectively to the business some functionality
as a development team I want to use Behavior-Driven Development

Scenario:  A scenario is a collection of executable steps of different types

Given step represents a precondition to an event
When step represents the occurrence of the event
Then step represents the outcome of the event

Scenario:  Another scenario exploring different combination of events

Given a precondition
When a negative event occurs
Then a the outcome should be captured
==========================================================================
```

The above story with several scenarios is often called a **fixture.** This means that such stories are fixed (very stable!) specifications of the desired behavior of the product or system. It is very important to realize that a fixture should be understandable for people having very different roles in the development of a complex product. For example, such a fixture should be clear for Product Owner, for Architect, for Developers and, in some situations, for Customers. Therefore, **a fixture language** is the most important parameter in the evaluation of the above BDD Frameworks.

The *second* important evaluation parameter is a **fixture implementation**. In fact, such an implementation itself can be considered as a matter of taste, but the crucial point is fixture **independence** from its implementation. For instance, if fixture semantics explicitly or implicitly forces some aspects of its implementation, then eventual changes or updates in product implementation can provoke changes in the fixture. The latter is *unacceptable* feature of a BDD framework.

The *third* evaluation parameter is the **report**. A report is generated as a result of the evaluation of the fixture with some hidden implementation. It is also extremely important aspect of a BDD framework. A complete fixture of a complex enterprise-scale product can contain many pages of the above scenarios and steps. Its evaluation can produce failures or other results for different steps. Therefore, the corresponding report should represent possible very large result in a clear structured form.

Next, the *forth* evaluation parameter is an **integration with a build-server,** for instance with **Jenkins**. There are two aspects: how the evaluation report is processed in the build-server and how the complete *Continuous Integration server* can be constructed using the BDD framework.

Finally, the *fifth* evaluation parameter is the **developer acceptance**. This refers to the basic skills and standards that are used by the developers. This includes such aspects as Maven and the corresponding Maven-Plug-ins, Eclipse PlugIns, clear and simple documentation and comfortable development process.

Cucumber Evaluation

Fixture Cucumber used the most standard and widely accepted approach. The fixture semantics is completely independent from its implementation. Moreover, Cucumber itself is a platform-independent framework. This creates a large world-wide community and reliable support. Moreover, it a special variant called **Cucumber-JVM** that is specially developed for the Java-platform.

Fixture implementation provides most of standard tools for test-development such as Pico-Container, CDI, Spring, EJB, WEB, etc. An implementation is annotation-driven and can be easily used with existing JUnit-code.

Report can be generated in many different formats such plain ASCII, HTML, JSON, XML, etc. Moreover, a report provides a very clear indication of errors and pending steps.

Integration with a build-server is aimed at **Jenkins** and it is provided with a special cucumber-jvm plugin. This plug-in insures a very good representation of the evaluation of large enterprise-sclale fixtures. Moreover, the plug-in has a good documentation that allows different customizations and features

Developer acceptance should be considered as an acceptable. The framework is Maven-based and is supported with numerous resources and Maven-plugins. However, it needs good skills in Maven development. For instance, installation from scratch of non-trivial demos such CDI. Spring or WEB needs significant Maven-corrections and experiments.

jbehave Evaluation

Fixture exploits just the same standard and widely accepted approach as *Cucumber* does. The fixture semantics is also completely independent from its implementation.

Fixture implementation also provides most of standard tools for test-development such as Pico-Container, CDI, Spring, EJB, WEB, etc. An implementation is also *annotation-driven* and can be easily used with existing JUnit-code. However, in comparison with *Cucumber-JVM,* the implementation is no longer exploits JUnit as the basic run-time engine and reference. Instead, it basically uses annotated POJO-classes. This is an important concept since development of a fixture evaluation should not be considered as a collection of *independent* unit-tests (the latter is the backbone of the good JUnit-test development). Moreover, annotations seems to be more comfortable than that in *Cucumber*

Report can be generated in many different formats and allows a flexible configuration. More its standard (non-configured) quality is much better that that in *Cucumber-JVM*.

Integration with a build-server provides Jenkins-plugin. However, it seems that the main stream is directed towards a special Continuous *Integration server.* This means that integration with Jenkins is not that reliable as *Cucumber* has.

Developer acceptance should be considered as **perfect** (see some interesting experience in Automatisierte Akzeptanz-Tests mit JBehave). The framework is Maven-based and it has very clear Maven-support. Moreover, there is the **Eclipse Plugin.**

Spoke Framework Evaluation

Fixture exploits groove as its language. This immediately turns the framework into a **specification**-aimed framework. In other words, it is NOT acceptable as an Acceptance Tests framework.
Fixture implantation provides very detailed tests of the product-components. It also is very good for the so-called Test-Driven development in agile projects under SPRINT.
Report ?
Integration with a build-server with Jenkins and Puppet Labs provide a complete Continuous Integration framework.

Concordion Evaluation

Fixture exploits HTML as its language. Its syntax and semantics are not standard and not widely accepted. Moreover, a fixture uses references to Java-beans used in the fixture implementation. This is unacceptable in a BDD environment
Fixture implementation is simple but too primitive. An implementation is always dependent on JUnit.
Report has a normal quality but not acceptable for large enterprise-scale fixtures.
Integration with a build-server Jenkins is provided
Developer acceptance is limited since the *framework is too primitive compared with Cucumber-JVM and jbehave.*

CONCLUSION

The **jbehave** *should be recommend as the basic framework for development of acceptance tests*. It provides most of modern development tools and it conforms most of widely accepted standards. Moreover, it provides a very *comfortable* environment for developers with different skills. However, there are some **unclear** aspects. As it has already been mentioned, the *jbehave-team* **doesn't** consider Jenkins as the main-stream build-server. Instead, they consider another more modern server. In this respect, it is not clear usage of the Puppet Labs framework. In comparison with *Cucumber-JVM,* it is known that the *Cucumber-JVM/Jenkins/ Puppet Labs* can used to create a complete reliable CI-Server. This topic needs some research and some experiments

References

Checkout git@git.ityx.de:ityx-all/ityx-test.git/BDD and look at README.xmind
There is a lot of run-able examples with comments