On Requirements for Acceptance Testing Automation Tools in Behavior Driven Software Development

Evgeny Pyshkin
Maxim Mozgovoy
Mikhail Glukhikh

St. Petersburg Polytechnical University
The University of Aizu
Joint Group in Information Retrieval & Software Research
О требованиях к средствам автоматизации приемочных тестов при использовании подхода «разработка, управляемая описанием поведения»

Евгений Пышкин
Максим Мозговой
Михаил Глухих

Санкт-Петербургский Государственный политехнический университет
Университет Айзу

Совместная научно-исследовательская группа «Информационный поиск и программное обеспечение»
Classified

• **Joint Group of Information Retrieval and Software Research**
  - Problems of higher education in field of programming and software engineering
  - Software development and testing
  - Information retrieval, especially MIR
  - Natural language processing and virtual learning labs
  - Software reliability…

Why did we write today’s paper

- Know more about BDD solutions
- Try to use
- Learning from practice:
  - Testing technique vs. development practice
- Maturing approach
  - Debates & definitions
Acceptance Testing Automation in BDD

• What are acceptance tests?
• What are main difficulties about acceptance tests?
• What’s the BDD?
• How it works?
• What did we do?
What are acceptance tests?

- USER REQUIREMENTS DEFINITION
- REQUIREMENT SPECIFICATION DEFINITION
- ARCHITECTURAL DESIGN
- DETAILED DESIGN
- UNIT TESTS
- INTEGRATION TESTS
- SYSTEM TESTS
- ACCEPTANCE TESTS
- Project Vision
- Accepted Software
- User Requirements
- Requirement Specification
- System Architecture
- Interfaces and Classes
- Compiled Modules
- Tested Modules
- Tested Subsystems
- Tested System
- Activity
- Product
- Verification
The main idea of BDD

CUSTOMERS

DEVELOPERS

QA ENGINEERS
What are main difficulties?

• To run tests we have to run code
• To test code we have to know the code
• Customers don’t know anything about the code
• But they want to be sure that the program fits the requirements
What’s the BDD?

• Suppose we create software for Kalah game
What’s the BDD?

- We know the rules
How it works?

Given a default game
When I take stones from cell 1
Then score should look like 0:1
Given a default game
When I take stones from cell 1
Then score should look like 0:1

public class AnnotatedStoryBase {
    protected final static FieldFactory factory = FieldFactory.getInstance();
    protected Field field;

    @Given("a default game")
    public void givenDefaultGame() {
        field = factory.createDefaultField();
    }

    @When("I take stones from $cell")
    public void takeStonesFromCell(int cell) {
        // Implementation of the move
        //...
    }

    @Then("score should look like $lower:$upper")
    ...

JBehave
public class AnnotatedStoryBase {
    ...
    @Given("a default game")
    public void givenDefaultGame() {
        ...
    }
    @When("I take stones from cell $cell")
    public void takeStonesFromCell(int cell) {
        ...
    }
    @Then("score should look like $lower:$upper")
    ...
}

public class StoryBase extends JUnitStory {
    @Override
    public Configuration configuration() {
        ...
    }
    @Override
    public List candidateSteps() {
        ...
    }
}

Running story kalah/jbehave/annotated_story_base.story
...
Scenario:
Given a default game
When I take stones from cell 1
Then score should look like 0:1

Results (console, HTML, XML, IDE)
Integration with JUnit and configuration
How it works?

• Let’s summarize:
  1. Stories
  2. Mapping to test classes
  3. Configuration
  4. Running tests
  5. Reports
What did we do?

• Analysis of the BDD communication schema
• Study of the BDD tools characteristics
  • Tools & features
  • Integration with IDEs
• Requirement analysis for a BDD supporting tool
# BDD tools

<table>
<thead>
<tr>
<th>Toolkit</th>
<th>Supported languages</th>
<th>User stories as plain text</th>
<th>Mapping rules</th>
<th>Automated mapping to the unit tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBehave</td>
<td>Java</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NBehave</td>
<td>.NET</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RSpec</td>
<td>Ruby</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MSpec</td>
<td>C#</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Cucumber</td>
<td>Ruby, Java, Python, .NET, C++, etc.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>StoryQ</td>
<td>.NET</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SpecFlow</td>
<td>.NET</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CBehave</td>
<td>C</td>
<td>Yes</td>
<td>Part.</td>
<td>No</td>
</tr>
</tbody>
</table>
Challenges & Considerations

⚠️ Units tests are easy to automate, conversions are not.

⚠️ Even if acceptance tests don’t change after changing requirements, the conversions may change.

⚠️ Are we always able to define behavior without diving into the code?

⚠️ From unstructured native language to the simplified “automatable” language.
<table>
<thead>
<tr>
<th>Toolkit</th>
<th>Deployment</th>
<th>IDE integration</th>
<th>IDE templates</th>
<th>Debug</th>
<th>Unit tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBehave</td>
<td>jar</td>
<td>No</td>
<td>No</td>
<td>Part.</td>
<td>JUnit</td>
</tr>
<tr>
<td>NBehave</td>
<td>Install</td>
<td>Plug-in for Visual Studio</td>
<td>No</td>
<td>Part.</td>
<td>NUnit, MbUnit, XUnit, MSTest</td>
</tr>
<tr>
<td>StoryQ</td>
<td>dll</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Visual Studio Unit Testing</td>
</tr>
<tr>
<td>SpecFlow</td>
<td>Install</td>
<td>Visual Studio</td>
<td>Yes</td>
<td>Yes</td>
<td>NUnit, Visual Studio Unit Testing</td>
</tr>
<tr>
<td>CBehave</td>
<td>source code</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Own</td>
</tr>
</tbody>
</table>
# Requirements: BDD inspired

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracing and debugging the test executions by marked-up scenarios</td>
<td>✔️</td>
</tr>
<tr>
<td>Test run reporting</td>
<td>✔️</td>
</tr>
<tr>
<td>Back trace to the story from the test run</td>
<td>✔️</td>
</tr>
<tr>
<td>Conversion of narrative stories to the marked-up scenarios</td>
<td>⚠️</td>
</tr>
<tr>
<td>Conversion from the marked-up scenarios to the unit tests</td>
<td>⚠️</td>
</tr>
</tbody>
</table>
### Requirements: BDD inspired

<table>
<thead>
<tr>
<th>“Running” user stories</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion from the unit tests to the marked-up scenarios</td>
<td>!</td>
</tr>
<tr>
<td>Including meta-information to the stories</td>
<td>!</td>
</tr>
<tr>
<td>Marking-up scenarios</td>
<td>!</td>
</tr>
<tr>
<td>More…</td>
<td></td>
</tr>
</tbody>
</table>
Summary

• State of art
  • *BDD tools are still oriented to the developers’ side in a greater degree*
  • *In many published cases the test stories and the marked-up scenarios were composed by the same engineers*
  • *Additional work for engineers*
Summary

• BDD ideas are *great* but implemented at surface level
Summary

• BDD ideas are *agile 😊* but implemented at surface level

• Even modest improvements can greatly increase the overall usability of BDD-supporting instruments
Thank you!

Evgeny Pyshkin
Maxim Mozgovoy
Mikhail Glukhikh

St. Petersburg Polytechnical University
The University of Aizu
Joint Group in Information Retrieval & Software Research