Automated Functional Testing based on the Navigation of Web Applications

Boni García - Juan Carlos Dueñas
Universidad Politécnica de Madrid (Spain)
1. Introduction
   – Research Statement
2. Background
3. Approach
4. Implementation: ATP
5. Case Study
6. Conclusions
1. Introduction

• Web has become in one of the most influential instrument in the history of mankind
• Software testing is the main technique to ensure quality and find bugs, but it is time-consuming
• Testing is often poorly performed or skipped by practitioners
• **Test automation** can help to avoid this situation
1.1. Research Statement

• Problem:
  – How to achieve automated functional testing in web applications?

• Proposal:
  – Automated evaluation of the correct navigation of web applications (assessment of the specified functional requirements) by using real browsers
Table of Content

1. Introduction
2. Background
   - Automated Software Testing
   - Web Modelling
   - Graph Theory
3. Approach
4. Implementation: ATP
5. Case Study
6. Conclusions
2. Background (I)

• Automated Software Testing (AST): “Application and implementation of software technology throughout the entire Software Testing Lifecycle (STL) with the goal to improve efficiencies and effectiveness”

• AST Frameworks (http://www.automatedtestinginstitute.com/):
  – 1\textsuperscript{st} generation: Record & Playback (R&P)
  – 2\textsuperscript{nd} generation: Data-driven
  – 3\textsuperscript{rd} generation: Model-based
2. Background (II)

• Web Modeling:

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>UI</th>
<th>Personalization</th>
<th>Transactional</th>
<th>Non-functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>UWE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W2000</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>WebML</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>NDT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

• Graph Theory:
  – Graph = Set of vertices (nodes) connected by edges (links)
  – Digraph = Graph in which edges have orientation
  – Multidigraph = Digraph in which multiples edges and loops are allowed
  – Path = Sequence of vertices such from each node there is an edge to the next vertex
# Table of Content

1. Introduction
2. Background
3. Approach
   - Finding Paths in a Digraph
   - Navigation Modelling
   - Data-Driven View
   - Automated Verification
4. Implementation: ATP
5. Case Study
6. Conclusions
3. Approach

• High-Level Description:
  1. To use graph theory to represent and work with the navigation of web applications
  2. To model correct navigation with different models (*pre-automation*)
  3. To develop framework to perform automated test generation, execution and reporting
  4. To provide data-driven testing by compiling test data and oracles in tabular files, i.e. Excel spreadsheets (*post-automation*)
3.1. Finding Paths in a Digraph (I)

- Possible methods/algorithms:
  - Graph Traversal: Breadth-First Search (BFS) and Depth-First Search (DFS)
  - Traveling Salesman Problem (TSP)
  - The Shortest Path Problem (SPP): Dijkstra, Bellman-Ford, A*, and Floyd-Warshall
  - The Chinese Postman Problem (CPP)
  - The Node Reduction (NR) algorithm
3.1. Finding Paths in a Digraph (II)

- Alternatives: CPP or NR

- **CPP:** \(E_1 \cdot E_3 \cdot E_4 \cdot E_5 + E_1 \cdot E_7 + E_1 \cdot E_2 \cdot E_6 + E_0\)
- **NR:** \(E_1 \cdot E_7 + E_0 \cdot E_5 \cdot E_6 + E_1 \cdot E_3 \cdot E_4 \cdot E_5 \cdot E_6 + E_1 \cdot E_2 \cdot E_6\)

*Is really better CPP than CPP?*
3.1. Finding Paths in a Digraph (III)

• CPP vs. NR: Laboratory Experiment:
  – Comparison between these algorithms, using random multidigraphs with incremental number of links (from 1 to 50).
  – For each digraph NR and CPP will be executed, comparing its cost (number of links employed in the resulting set of paths), and the computation time (milliseconds to achieve the solution).
  – Repeated 100 times, and the mean of the values (cost and time) will be displayed by charts.
3.1. Finding Paths in a Digraph (IV)

CPP has better behaviour in cost and time resolution!
3.2. Navigation Modelling (I)

- Modelling navigation by testers/developers (pre-automation):
  - UML: Based on NDT, created with Enterprise Architect (in XMI format):
    - Use cases
    - Activity diagrams
    - Presentation diagrams
  - R&P: HTML Scripts, recorded with Selenium IDE
  - XML: Self-defined XSD Schema
3.2. Navigation Modelling (II)

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<website xmlns="http://www.dit.upm.es/atp"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    base="http://localhost:8080/WebAdmin/">
    <home id="login">
        <data locator="username">
            <value>Administrador</value>
        </data>
        <data locator="password">
            <value>admin</value>
        </data>
    </home>
    <transition from="login">
        <action target="frmDatos_0" event="click" />
        <to state="init" />
        <to state="login" />
    </transition>
    <state id="init">
        <assert locator="texto-entrada" type="text">
            <value>Welcome</value>
        </assert>
    </state>
</website>
```
3.3. Data-Driven View

- Data-driven testing (post-automation): Test data and oracles in tabular files (Excel):

<table>
<thead>
<tr>
<th>Test data (input)</th>
<th>Oracles (expected output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( in_{\text{element}}<em>1 ) ... ( in</em>{\text{element}}_n )</td>
<td>( out_{\text{element}}<em>1 ) ... ( out</em>{\text{element}}_m )</td>
</tr>
<tr>
<td>( data_{1_1} ) ... ( data_{1_n} )</td>
<td>( outcome_{1_1} ) ... ( outcome_{1_m} )</td>
</tr>
<tr>
<td>( data_{2_1} ) ... ( data_{2_n} )</td>
<td>( outcome_{2_1} ) ... ( outcome_{2_m} )</td>
</tr>
<tr>
<td>... ... ... ...</td>
<td>... ... ... ...</td>
</tr>
</tbody>
</table>

- The elements are located in the DOM document using looking for:
  - \( \text{id, name, value, text attributes} \)
  - XPath expression
3.4. Automated Verification

• While automated browsing is performed, the following assessment is done:
  – State verification:
    • The existence of each defined data field is ensured
    • Each test oracle is assessed
    • Each state component in the DOM is checked (e.g. frames, images, …)
    • Each state is validated as the aggregation of the defined locators (data fields and oracles)
    • JavaScript notifications are captured
  – Transition verification:
    • Each transition locator is assessed and executed
3.5. Summary
Table of Content

1. Introduction
2. Background
3. Approach
4. Implementation: ATP
   - Architecture
   - Commands
5. Case Study
6. Conclusions
4. Implementation: ATP

• The proposed approach has been implemented in an open-source testing framework: Automatic Testing Platform (ATP):

http://atestingp.sourceforge.net/
### 4.1. ATP Architecture

- **Open-source components:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Library</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Framework</td>
<td>JUnit</td>
<td><a href="http://www.junit.org/">http://www.junit.org/</a></td>
</tr>
</tbody>
</table>
4.2. ATP Commands

• ATP has been implemented as a command-line (shell) tool:

```plaintext
> atp
[INFO] ATP (Automatic Testing Platform) v2.0
[INFO] [http://atestingp.sourceforge.net]
[INFO] Copyright (c) 2011 UPM. Apache 2.0 license.
[INFO]
[INFO] Use one of these options:
[INFO] atp create
[INFO] atp run
[INFO] atp clean
[INFO] atp list
[INFO] atp set <key> <value>
[INFO] atp report
```
# Table of Content

1. Introduction
2. Background
3. Approach
4. Implementation: ATP
5. Case Study
   - Pre-Automation
   - Post-Automation
   - Results
6. Conclusions
5. Case Study

• System Under Test: “Factur@”, which is an electronic invoice web management system which has been developed using a Model Driven Engineering (MDE) approach

• Research Questions (RQ):
  – RQ1: Does the Factur@ application accomplish its functional requirements?
  – RQ2: Is ATP capable of finding defects in a finished web application?
  – RQ3: What are the advantages and disadvantages of different types of input (UML, XML, and R&P) to ATP?
5.1. Pre-Automation (I)

- Use cases:
5.1. Pre-Automation (II)

- Activity diagrams:
5.1. Pre-Automation (III)

- Presentation UML diagrams:
5.2. Post-Automation

- Post-Automation: Test data and oracles in Excel files:
5.3. Results (I)

- Verdicts in the report:
  - 6 functional errors (due to broken links, not navigational issues)
  - 8 functional warning (due to JavaScript notifications)
5.3. Results (II)

• Answers to RQs:
  – RQ1: Factur@ has no navigation errors
  – RQ2: ATP can discover failures in web applications
  – RQ3: ATP input pros and cons:

<table>
<thead>
<tr>
<th>Input</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| UML   | • Reuse model for testing  
  • Models describes every possible path | • Test data and oracles are not attached in the models.  
  • Post-automation step is mandatory. |
| XML   | • Every possible path can be depicted.  
  • Data and oracles can be attached | • XML files must be coded and maintained by hand. |
| R&P   | • Scripts creation using Selenium IDE against the real application.  
  • Data and oracles can be attached to HTML scripts | • There is always a single path by HTML script (linear recording).  
  • Error paths should be defined in different scripts. |
# Table of Content

1. Introduction
2. Background
3. Approach
4. Implementation: ATP
5. Case Study
6. Conclusions
6. Conclusions (I)

• This piece of research has presented a method to assess functional requirements of web applications based on its navigation.

• Correct navigation structure must be defined (pre-automation): UML (NDT), XML, R&P (HTML).

• Data-driven approach (post-automation) based on generated Excel files with the data extracted from the model and easily extensible.

• Graph theory has been employed to model the navigation and CPP is the algorithm to find the paths.
6. Conclusions (II)

• Implementation of this approach in a open-source tool: Automatic Testing Platform (ATP)

• ATP is a mixed testing framework:
  – 1st generation: It can use R&P linear scripts
  – 2nd generation: It is data-driven
  – 3rd generation: It can use UML models

• Future work:
  – Extension of the approach to non-functional requirements: Performance, Security, Compatibility, Usability, and Accessibility
  – Release of the implementation, ATP v2.0 (currently only available ATP v1.0 to download)
Thank you!

Boni García

bgarcia@dit.upm.es