Model-based Testing
Theory and Application

Franz Wotawa
Technische Universität Graz
Institute for Software Technology
8010 Graz, Inffeldgasse 16b/2, Austria
wotawa@ist.tugraz.at
Point of departure
Open questions

• Have we built the right system?
  [VALIDATION]

• Have we built the system right?
  [VERIFICATION]
V&V activities

• Check correctness and expectations!
  – Formal verification

  – Testing
    (for verification and validation)
Testing framework

- Requirements
- Specification
- Source code
- Tests are characterized by:
  - Input values
  - Expected output values
- Test suite = set of tests
- Program is “correct” iff all tests are fulfilled!
Test characterization

- **Which information is available?**
  - **Black-box** vs. **White-box** testing

Examples: Model-based testing, Equivalence-based methods, Combinatorial testing, Coverage-based methods, Random-Testing (Monkey Testing, Fuzz Testing,..)
Testing – a program-centric view

• Which part of the program to be tested?
  – Unit-Tests
  – Component tests
  – Integration tests
  – System tests
  – User-interface Testing
Testing – a process-oriented view

- At which part of the development process testing is done?
  - Verification (Unit-Tests, regressions tests,...)
  - Validation
What should I test?

• Functionality
• Robustness
• Usability
Test automation

• 2 Levels:
  – Automated test case generation
    • From models or the source code (Oracle problem)
  – Automated test execution (e.g. JUnit)
    • Challenges because of different interfaces (Web, different OS platforms, databases, GUIs, ...)
    • Hardware In the Loop (HIL) testing
Model-based testing
Model?

Finite automata

Qualitative models

Constraints

1. o = i1 && i2;
2. e = if (i1 > 0) then i2 else 0;
3. ...

UML diagrams
Test case generation

• Directly from the model
  – Equation solving (constraints)
  – Traversing a graph
  – Combination of solving and graph traversal

• Feasible (at least for smaller models)
• Orthogonal to manual testing
• Focus (but not necessarily) system testing
TWO CASE STUDIES
GUI Savvy End-to-End Testing with Smart Monkeys

Birgit Hofer, Bernhard Peischl and Franz Wotawa

Technische Universität Graz
Institute for Software Technology
Example

TEST APPLICATION

NUMBER

N*4  DISPLAY

DISPLAY

OK

Wien, 10. November 2009
Example
(cont.)

State Properties
0  -
1 $\text{TextFieldProperty(number, 0)} \land \text{NotIsHungProperty()}$
2 $\text{NotIsHungProperty()}$
3 $\text{TextFieldProperty(number, num)} \land \text{NotIsHungProperty()}$
4 $\text{WindowCaptionProperty(DISPLAY)} \land \text{NotIsHungProperty()}$
Windows Calculator Case
Found faults

• Event sequenz 1\textsuperscript{st} fault:
  – The monkey produces a division by zero (e.g. $65 / 0$),
  – then it opens the menu item \texttt{?/Help}.
  – The value in the text field changes from the error message \textit{‘Division by 0 not possible’} to a number.

• Event sequence 2\textsuperscript{nd} fault:
  – The monkey produces a division by zero,
  – then it opens the menu item \texttt{?/Info}.
  – The info menu does not appear
FileZilla Case

• Open Source FTP client (www.filezilla.de)

• 3 Models:
  – Connecting to server (quickconnect bar)
  – Test of menu items (offline test)
  – File operations (transfer, delete,...)

• Models have in sum 113 states and 301 transitions
Model coverage

State coverage

Transition coverage
Code coverage

• **Function coverage**
  – Up to 55 % after 1 hour of testing

• **Condition coverage**
  – Up to 26 % after 10 hours of testing

• **Reasons / Explanations:**
  – Models do not cover the whole functionality
  – Not all GUI elements used in models
  – Not all parts of the code can be tested using the GUI
Fault detection capabilities

- 3 faults introduced in original source code
- All faults found (after 10 hours)
- On average 30 minutes to detect a fault
Coverage Based Testing with Test Purposes

Gordon Fraser  Martin Weiglhofer
Franz Wotawa

Institute for Software Technology
Graz University of Technology

QSIC 2008
LTS Model

- Labeled Transition Systems (LTS)
Test case generation

• Test purpose based
• Traversing the model
• Result: Sequence of inputs and outputs

• Case study SIP registrar (VoIP telephony)
## Results test generation

<table>
<thead>
<tr>
<th>C.</th>
<th>No. TP</th>
<th>ok</th>
<th>Regular</th>
<th>Minimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>∞</td>
<td>time</td>
</tr>
<tr>
<td>A</td>
<td>27</td>
<td>25</td>
<td>2</td>
<td>2h49m</td>
</tr>
<tr>
<td>D</td>
<td>78</td>
<td>72</td>
<td>6</td>
<td>8h28m</td>
</tr>
<tr>
<td>C</td>
<td>98</td>
<td>94</td>
<td>4</td>
<td>11h13m</td>
</tr>
<tr>
<td>CD</td>
<td>176</td>
<td>166</td>
<td>10</td>
<td>19h39m</td>
</tr>
<tr>
<td>Σ</td>
<td>379</td>
<td>357</td>
<td>22</td>
<td>42h9m</td>
</tr>
<tr>
<td>C.</td>
<td>OpenSER</td>
<td>commercial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>x</td>
<td>?</td>
<td>✓</td>
</tr>
<tr>
<td>A</td>
<td>40</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>100</td>
<td>16</td>
<td>28</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>124</td>
<td>25</td>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td>CD</td>
<td>224</td>
<td>41</td>
<td>67</td>
<td>3</td>
</tr>
<tr>
<td>∑</td>
<td>488</td>
<td>83</td>
<td>143</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C.</th>
<th>OpenSER</th>
<th>commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>CD</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>∑</td>
<td>45</td>
<td>6</td>
</tr>
</tbody>
</table>
WHAT’S ABOUT SECURITY TESTING?
Applications to security testing

• Test case generation based on models of attack patterns!

• Literature:
Vulnerability Detection

SQLI: `x' OR 'x'='x`

Success!
Vulnerability Detection

**XSS:** `<script>alert(document.cookie)</script>`

**Reflected**

What's your name?

```html
<script>alert(document.cookie)</script> Submit
```

**Stored**

```
Name *
Entry

Message *
<script>alert(document.cookie)</script>
```

Sgn Guestbook
Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

**reflected**

What's your name?

```html
<script>alert(document.cookie)</script>
```  
Submit

**stored**

Name *

Entry

Message *

`<script>alert(document.cookie)</script>`

Sign Guestbook

security=low; PHPSESSID=50d88629b1c35158e63be55e8948d67b

OK
Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

reflected

stored

> Success!

`<pre>Hello <script>alert(document.cookie)</script></pre>`
Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

**reflected**

**stored**

> Failure!

`<pre>Hello &lt;script&gt;alert(document.cookie)&lt;/script&gt;</pre>`
Model-based security testing
Evaluation

- Five applications: NOWASP (Mutillidae) [8], Damn Vulnerable Web App (DVWA) [9], Bodgelt [10], Wordpress [11] and Anchor CMS [12].
- First three contain several security levels with every one having more sophisticated filtering mechanisms.
- Other programs are tested only for the second type of XSS because these are blog software, where posts are stored inside a database.
- All applications have been deployed on an Apache Server and comprise a MySQL database.
- Collection of 33 custom SQLI and 107 XSS input strings.
## Evaluation

<table>
<thead>
<tr>
<th>Application</th>
<th>Type of vulnerability</th>
<th>Security Level</th>
<th>Average execution time (s)</th>
<th># of successful injections</th>
<th>% coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVWA</td>
<td>SQLI</td>
<td>low</td>
<td>8.47</td>
<td>8</td>
<td>24.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>medium</td>
<td>10.55</td>
<td>2</td>
<td>6.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RXSS</td>
<td>SQLI</td>
<td>low</td>
<td>23.00</td>
<td>15</td>
<td>14.02</td>
</tr>
<tr>
<td>SXSS</td>
<td>SQLI</td>
<td>low</td>
<td>26.60</td>
<td>15</td>
<td>14.02</td>
</tr>
<tr>
<td>Mutillidae</td>
<td>SQLI</td>
<td>low</td>
<td>15.69</td>
<td>5</td>
<td>15.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>medium</td>
<td>17.94</td>
<td>5</td>
<td>15.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RXSS</td>
<td>SQLI</td>
<td>low</td>
<td>42.20</td>
<td>40</td>
<td>37.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>medium</td>
<td>52.60</td>
<td>40</td>
<td>37.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SXSS</td>
<td>SQLI</td>
<td>low</td>
<td>53.30</td>
<td>17</td>
<td>15.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>medium</td>
<td>78.10</td>
<td>17</td>
<td>15.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Bodgelt</td>
<td>SQLI</td>
<td>-</td>
<td>8.50</td>
<td>3</td>
<td>9.09</td>
</tr>
<tr>
<td></td>
<td>RXSS</td>
<td>-</td>
<td>27.20</td>
<td>13</td>
<td>12.15</td>
</tr>
<tr>
<td></td>
<td>SXSS</td>
<td>-</td>
<td>26.30</td>
<td>26</td>
<td>24.30</td>
</tr>
<tr>
<td>Wordpress</td>
<td>SXSS</td>
<td>-</td>
<td>33.5</td>
<td>7</td>
<td>6.54</td>
</tr>
<tr>
<td>Anchor</td>
<td>SXSS</td>
<td>-</td>
<td>30</td>
<td>8</td>
<td>7.48</td>
</tr>
</tbody>
</table>
Evaluation

• Both attack patterns have been slightly adapted.
• Wordpress was tested while our application was authenticated so all inputs were submitted after that step.
• Anchor CMS is similar to Wordpress with the difference that all posts have to be approved by the administrator.
• It was impossible to detect vulnerability on the hardest security level of the first three apps, which means that a more sophisticated test case generation strategy has to be adapted for this purpose.
• In Mutillidae, HttpClient enables communication on medium and hard level.
What’s next?

• Modeling of attacker

• Idea:
  – attack = sequence of actions = plan
  – use A.I. planning for attack generation
  – more flexible
Conclusion

• Model-based testing finds faults that have been previously undetected (using manual tests)
• Completely automated generation
• Requires model (+ test purposes)
• Complementary to manual testing
• Can be used for security testing too!
General Chair:
- Franz Wotawa
  (TU Graz, Austria)

PC Chairs:
- Gordon Fraser
  (Univ. Sheffield, UK)
- Darko Marinov
  (Univ. of Illinois, Urbana-Champaign, USA)
Thank you for your attention!

“What I cannot create, I do not understand.”

Richard Feynman
(1918-1988)