Test Automation: An Empirical Perspective.  

Part II – Testing Web Applications

Long Tutorial at the GTTSE Summer School on Generative and Transformational Techniques in Software Engineering, Braga, Portugal, 2011

Arie van Deursen, Delft University of Technology
Joint work with Ali Mesbah (UBC) & Danny Roest (TU Delft)
Invariant-Based Automated Testing of Modern Web Applications

Ali Mesbah (UBC)
Arie van Deursen (TU Delft)
Danny Roest (TU Delft)
GTTSE Overview

• Monday: Reflections on *impact & empirical methods*

• Today: Reflections on *test automation for the web*

• Tomorrow: Reflections on *compositional testing*
Zef Hemel

Posts

Zef Hemel - 2:48 PM - Public
Cool, first StateOfCode post that makes the Hacker News homepage (http://news.ycombinator.com/item?id=2729545)

Zef Hemel - 2:23 PM - Public
Javascript will be the Java bytecode of the 2010s

1 share - Béla Varga
Asynchronous Javascript & XML
Frameworks for Programming Ajax

- Javascript programming
- Using libraries (JQuery)
- Code generators (Google Web Toolkit)

```html
<a href="#"
    onClick="OpenNewsPage();">
</a>

<span onClick="OpenNewsPage();">
</span>

<div class="news">

\$(".news").click(function() {
    \$("#content").load("news.php");
});
```
An Ajax Fault Model

- Stateful client;
- Asynchronous communication;
- Delta updates;
- Untyped JavaScript;
- Client-side DOM manipulation;
- Event handling;
- Timing;
- Back/forward, undo/redo
- Browser dependence (IE, FF, Chrome, ...)

Focus on client side GUI logic.
Different fault models / test strategies for server side / database / business etc. logic.
Testing Web Applications

Traditional:
• Response – Request
• No client side logic

Capture & Playback:
• Selenium
• Manual effort

Can we push these limits?
State: DOM Tree
Edge: Event that is fired and causes a state transition
Steps Involved

• Finding (new) clickables
• Triggering events
• Comparing DOM trees
• Recurse
• Backtrack
• Input data generation
Detecting Clickables

• *Candidate* elements:
  – HTML tagnames: DIV, SPAN, A, …
  – attribute constraints: DIV:{class="menuitem"}
  – Excluded elements: A:{title="logout"}

• Expose each element to real events: (click, mouseover, dbclick, …)

• DOM changed after click? Element clickable; new state.
Detecting State *Changes*

Compare DOM Tree *before* and *after* event

Levenshtein edit distance
Oracle Comparator Pipelining

• Diff does not suffice

• Multiple types of differences (*patterns*) to ignore:
  – styles, attributes, dates, tables, lists, ...

• Every Oracle Comparator strips its own specialized differences from the DOM and passes the stripped version to the next comparator.

• Use a simple string comparison at the end of the pipeline.

• Very fast and scalable.
Manage Data Entry Points

• Default: Enter random data.
• Engineer can provide custom data

```java
// some fields for which random data isn't ok.
InputSpecification input = new InputSpecification();
input.field("email").setValue("steve.jobs@apple.com");
input.field("email").setValue("an-invalid-email-address");
input.field("username").setValue(USER_NAME);
input.field("password1").setValue(PASSWORD);
input.field("password2").setValue(PASSWORD);
crawler.setInputSpecification(input);
```
Implementation: Crawljax

• Open source: http://crawljax.com

• Java, maven, JUnit, Apache libraries
• Relies on webdriver to access the browser
  – IE, Firefox, Chrome

• Plugin-based architecture

• Contributions from NL, Canada, Japan, ...
• Used at Microsoft, Google, Fujitsu, eBay, ...
Example: MirrorGenerator

```java
import java.io.FileWriter;
import com.crawljax.core.CrawlSession;
import com.crawljax.core.plugin.OnNewStatePlugin;

public class MirrorGenerator implements OnNewStatePlugin {

    public void onNewState(CrawlSession session) {
        try {
            String dom = session.getBrowser().getDom();
            String fileName = session.getCurrentState().getName();

            FileWriter fw = new FileWriter(fileName, false);
            fw.write(dom);
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
# Scalability Indicators

<table>
<thead>
<tr>
<th>Case</th>
<th>DOM string size (byte)</th>
<th>Candidate Clickables</th>
<th>Detected Clickables</th>
<th>Detected States</th>
<th>Crawl Time (s)</th>
<th>Depth</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>4590</td>
<td>540</td>
<td>16</td>
<td>16</td>
<td>14</td>
<td>3</td>
<td>A, DIV, SPAN, IMG</td>
</tr>
<tr>
<td>C2</td>
<td>24636</td>
<td>1813</td>
<td>33</td>
<td>34</td>
<td>26</td>
<td>2</td>
<td>A, IMG</td>
</tr>
<tr>
<td>C3</td>
<td>262505</td>
<td>150</td>
<td>148</td>
<td>148</td>
<td>498</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>19247</td>
<td>1101</td>
<td>1071</td>
<td>5012</td>
<td></td>
<td>2</td>
<td>A, TD</td>
</tr>
<tr>
<td>C4</td>
<td>40282</td>
<td>3808</td>
<td>55</td>
<td>56</td>
<td>77</td>
<td>2</td>
<td>A, DIV, INPUT, IMG</td>
</tr>
<tr>
<td>C5</td>
<td>165411</td>
<td>267</td>
<td>267</td>
<td>145</td>
<td>806</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>32365</td>
<td>1554</td>
<td>1234</td>
<td>6436</td>
<td></td>
<td>2</td>
<td>A, DIV</td>
</tr>
<tr>
<td>C6</td>
<td>134404</td>
<td>6972</td>
<td>83</td>
<td>79</td>
<td>701</td>
<td>1</td>
<td>A, DIV</td>
</tr>
</tbody>
</table>
Crawling for *Testing* Purposes?

- Smoke tests (e.g., crashes)
- Structural Invariants
- Performance (e.g., memory leak in JS)
- Browser compatibility
- Regression
- Scalability (e.g., concurrent clients)
- Leverage derived model
Invariants as Oracles

• Generic, on the DOM-tree state
  – validity, error messages in DOM, accessibility, link discoverability, security...

• Generic, between successive DOM states
  – Consistent back-button

• Application-specific (design) invariants:
  – Constraints on specific element and attribute relations in particular DOM states
Expressing Invariants

- Conditions (e.g., on DOM tree, JS variables)
- XPath expressions
- Regular expressions
- JavaScript variables
- DOM Templates

```java
//menu on News state should always have the class attribute newsElement
Condition preCondition = new JavaScriptCondition("document.title == 'News';");
String xpath = "/DIV[@id='menu']/UL/LI[contains(@class, 'newsElement')];
crawler.addInvariant(new XPathCondition(xpath, preCondition));
```
Test Case Generation

• Post-crawling plugin to generate test suite from inferred state machine

• K shortest paths from Index to sinks

• Cycles included only once:
  – “All roundtrip-path” coverage

• Transform each event path into a JUnit test case method.
@Test
public void method1() {
    browser.goToUrl(url);

    Clickable c1 = new Eventable(new Identification("xpath",
        "/DIV[1]/SPAN[4]"), "click");
    assertPresent(c1);
    browser.enterRelatedInputValues(c1);
    assertTrue(browser.fireEvent(c1));
    assertTrue(invariantsChecked(browser.getDom()));
    assertEquals(oracle.getState("S_1").getDom(),
        browser.getDom());

    Clickable c2 = new Eventable(new Identification("xpath",
        "/SPAN[2]/DIV[2]"), "click");
    assertPresent(c2);
    ...
}
Past Executions as Oracles

Application to *Regression Testing*

1. Infer a model of a trusted version
2. Create test suite with the model
3. Run test suite on newer version
4. Check all the links and compare the states
Finding & Visualizing Relevant Differences
Cross-Browser Testing

Step 1: Crawl web app. under different environments and capture behavioral model under each environment.

Step 2: Establish pair-wise equivalence of the generated models.

Web Application + Crawljax

Execution environment (OS + Browser)

Models of crawled behavior
- AdSense 3.0
Example: Accessibility testing
Empirical Evaluation

Goal: assess usefulness of approach in supporting testing of modern web applications

RQ1: What meaningful invariants can be obtained?
RQ2: What is the effectiveness / fault revealing capability?
RQ3: What is the performance and scalability?
RQ4: What is the automation level? How much manual effort is involved?
Study 1: Invariants (RQ1) -- Setup

Four open source cases. Per case

1. Run Crawljax w/ default settings + overview
2. Analyze graph manually;
   Adjust config to cover additional UI states
3. Inspect states for candidate invariants
   (Firebug, Firefinder, JS, Xpath)
4. Encode invariants
5. Run to check invariants
# Example Invariants

## Pointing to Faults

<table>
<thead>
<tr>
<th>Subject System</th>
<th>Inv. Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>THEORGANIZER</td>
<td>The Year View state //img[contains(@src, 'head_yearView')] contains at least one appointment item //input[@id='edit']</td>
<td>XPath expr. state invariant</td>
</tr>
<tr>
<td>THEORGANIZER</td>
<td>Failed to populate the list properly!</td>
<td>Generic ('Fail%') DOM invariant</td>
</tr>
<tr>
<td>THEORGANIZER</td>
<td>Appointment item (view) structure</td>
<td>Reg. expr. state invariant</td>
</tr>
<tr>
<td>THEORGANIZER</td>
<td>Clicking on the logoff button //img[contains(@id, 'logoff')] results in a state with //div[contains(text(), 'You have logged out')]</td>
<td>application-specific SM inv.</td>
</tr>
<tr>
<td>THEORGANIZER</td>
<td>Clicking on //img[@id=&quot;X&quot;] results in a state with //img[contains(@src, 'head_X')], where X is a string</td>
<td>application-specific SM inv.</td>
</tr>
<tr>
<td>TASKFREAK</td>
<td>Top level body contains div[@id='header'], div[@id='container'], and at most one div[@id='calendar']</td>
<td>XPath expr., state invariant</td>
</tr>
<tr>
<td>TASKFREAK</td>
<td>All pages displaying current tasks via table[@id='taskSheet'] match a given template</td>
<td>Reg. expr. state invariant</td>
</tr>
<tr>
<td>TASKFREAK</td>
<td>Reload button in any state found via img[@id='frk-status'] should lead to state displaying current tasks</td>
<td>application-specific SM inv.</td>
</tr>
<tr>
<td>HITLIST</td>
<td>Contact template, shown in Figure 13</td>
<td>Reg. expr. (template) state invariant</td>
</tr>
<tr>
<td>THTUNNEL</td>
<td>global variable alive is true during the game, and false after player fails</td>
<td>JAVA SCRIPT invariant</td>
</tr>
<tr>
<td>THTUNNEL</td>
<td>position of ship must be 32 times higher than the wall ship_x+32 &gt;= right_wall</td>
<td>JAVA SCRIPT invariant</td>
</tr>
<tr>
<td>THTUNNEL</td>
<td>the background value must be between 0 and 20</td>
<td>JAVA SCRIPT invariant</td>
</tr>
<tr>
<td>All systems</td>
<td>Back button</td>
<td>Generic SM inv.</td>
</tr>
</tbody>
</table>
Study 1: Findings

• Invariants capture & require understanding of the design of the web application
• Invariants can be easily expressed:
  – Relations between / attrs of elements: XPath
  – Invariants over structure: Templates
  – Actions + effects: State machine invariants
  – Code-level Javascript invariants
• Faults : Memory leaks, crossbrowser, regressions
• Manual effort < 1 hour per case
Study 2: Tudu / RQ2-4

- Ajax-based open source todo list manager
  - Used by other researchers as well
- J2EE, 12,000 loc Java/JSP, DWR, Scriptaculous
  - Javascript: 11,000 libraries, 580 custom code
5 - Documentation

Share the current Todo List

Share with user: another_user

Current users:
- a_user (Remove)
- another_user (Remove)

Add login

Add some Maven reports

RSS | Backup | Restore

Did you find a bug? Thanks for submitting it
Configuration (I)

• **Settings (property file)**
  – URL to deployed site
  – Included tag elements: `A`, `DIV`
  – Excluded elements: `A:title=“Log out”`
  – Depth level: 2
  – Similarity threshold 0.89
  – Maximum crawling time: 60 minutes

• **Wrote `preCrawling` plugin to log into the web application automatically**
Configuration (II)

• Initial run: find new states and data entry points recursively
• Provide custom input values for data entry points
• Activated 4 generic plugins:
  – DOM Validator, Back button, Error Detector, Test Case Generator
• RQ4: Total configuration time: ~ half an hour
Effectiveness

RQ2: How to assess effectiveness of test suite?

• Approximation I: line coverage
  – Java/Clover, Javascript/JSCoverage
  – 73% server side; 75% custom JS; 35% library JS

• Approximation II: finding (10) seeded faults:
  – Malformed DOM, oversized values, duplicate todo items, removing all items instead of one, ...
  – 8 found; 2 swallowed by JS not affecting DOM

Total time used by runs (RQ3): 6 minutes
Study 3: “Coach Your Self”

• Commercial application supporting ~5000 teachers in self-assessment

• Migration of Java applet to Ajax
  – Synchronized tree-based TOC navigation

• Use to evaluate manual effort (RQ4) and capability to find faults actually occurred during development process (RQ1/2).
CYS Case Study

• Two developers, two person weeks

• Ajax solution:
  – JQuery + treeview, history-remote, listen plugins
  – Custom code: 150 loc JS + CSS + HTML

• Developers were asked to
  – Document design decisions using invariants
  – Turn invariants into plugins
CYS Invariants (I)

• Invariants to document external “treeview” plugin for JQuery library
  – (Un)folded tree based on class attributes in lists
  – Collapsible, hit-area, lastExpandable-hitArea, ...
  – JS intercepts clicks and re-arranges class attrs as needed
  – Constraints: Div-element below li.expandable should have expandable-hitarea as its class.
  – Invariants help to document such design decisions
CYS Invariants (II)

- Invariants for CYS custom code.
- Synchronizing interactive display of table of content with actual content shown
  - Invariant I: at most one path labeled as “current”
  - Invariant II: current page actually displayed
- Invariants effective in finding faults:

<table>
<thead>
<tr>
<th>Failure</th>
<th>Cause</th>
<th>Violated Invariant</th>
<th>Inv. type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images not displayed</td>
<td>Base URL in dynamic load</td>
<td>Dead Clickables</td>
<td>Generic</td>
</tr>
<tr>
<td>Broken synchronization in IE</td>
<td>Invalid HTML id</td>
<td>DOM-validator</td>
<td>Generic</td>
</tr>
<tr>
<td>Inconsistent history</td>
<td>Issue in listen library</td>
<td>Back-Button</td>
<td>Generic</td>
</tr>
<tr>
<td>Broken synchronization in IE</td>
<td>Backslash versus slash</td>
<td>Consistent current page</td>
<td>Specific</td>
</tr>
<tr>
<td>Corrupted table</td>
<td>Coding error</td>
<td>treeview invariants, Consistent current page</td>
<td>Specific</td>
</tr>
<tr>
<td>Missing TOC Entries</td>
<td>Incomplete input data</td>
<td>Consistent current page</td>
<td>Specific</td>
</tr>
</tbody>
</table>
Discussion

• Scope: *client side faults.*
• Invariants: *seem to work!*
• Impact of *generated* Javascript?
• Manual effort: *ok.*
• Performance and scalability – *a real challenge*
• Application size – 10,000+ states
• Testing strategies – beyond invariants
Threats to Validity

• *See discussion, and:*

• External
  – Additional applications: Google, Microsoft, Fujitsu

• Internal
  – Complex system, built on top of many OS libs
  – Countermeasure: well tested, open source

• Repeatability
  – Most cases open source
Aim: Sharp & Critical Case Studies

• Explicit goal
• Explicit plan / setup
• Units of analysis
• Tabularize
• Explain / narrative
• Rival explanations?
• Analytical generalization
• Discussion / interpretation
This article examines five common misunderstandings about case-study research: (a) theoretical knowledge is more valuable than practical knowledge; (b) one cannot generalize from a single case, therefore, the single-case study cannot contribute to scientific development; (c) the case study is most useful for generating hypotheses, whereas other methods are more suitable for hypotheses testing and theory building; (d) the case study contains a bias toward verification; and (e) it is often difficult to summarize specific case studies. This article explains and corrects these misunderstandings one by one and concludes with the Kuhnian insight that a scientific discipline without a large number of thoroughly executed case studies is a discipline without systematic production of exemplars, and a discipline without exemplars is an ineffective one. Social science may be strengthened by the execution of a greater number of good case studies.

**Keywords:** case study; case selection; critical cases; validity in case studies

When I first became interested in in-depth case-study research, I was trying to understand how power and rationality shape each other and form the urban environments in which we live (Flyvbjerg, 1998). It was clear to me that to understand a complex issue such as this, in-depth case-study research was necessary. It was equally clear, however, that my teachers and colleagues kept dissuading me from employing this particular research methodology.

“You cannot generalize from a single case,” some would say. “and social science needs generalizability.” I was also frequently reproached for not having enough theoretical knowledge. But of course, theoretical knowledge is necessary for any form of research. A single-case study can be highly informative as long as it is carefully conducted and analyzed. In fact, the qualitative research paradigm emphasizes the importance of in-depth analysis of a single case. A single-case study can be highly informative as long as it is carefully conducted and analyzed. In fact, the qualitative research paradigm emphasizes the importance of in-depth analysis of a single case.
Case Study = Sum of all your Choices

- Choice of systems
- Choice of faults
- Configuration of tools
- Running of tools
- Observation of people
- Narrative on how used
- Measuring performance, time, scalability info, ...

McGrath
- Generalizability
- Realism
- Precision

In search for “possibilities of action”
Tutorial Overview

• Monday: *impact & empirical methods*

• Today: *test automation for the web*

• Tomorrow: *compositional testing*