Automated Oracle Comparators for Testing Web Applications

Sara Sprenkle

Lori Pollock

Holly Esquivel

Barbara Hazelwood

Stacey Ecott
Testing Web Applications

- 772 million people online in May 2007*

- Amazon.com had 142K unique visitors, on average visit 3.1 times, in May 2007*
  - Earned $10.7 billion in revenue in 2006

- Need reliable web applications
  - Errors are seen by many, quickly
  - Frequent maintenance/update cycles
  - Need fast, effective oracles

* Source: comScore
Oracles for Web Applications

- Challenge: Low observability of outputs
- Solution: Use “pseudo-oracle” [Weyuker 82] on HTML
  - HTML: Common output for web applications
Overview of Contributions

- Developed 22 automated HTML comparison algorithms
  - Why HTML? Common web application output, users see output, side effects of failures may show up in HTML

- Evaluated comparators on 4 deployed subject applications
  - False positives, false negatives

- Made recommendations to testers about selecting effective oracles
Our Oracle Process

- **Expected results:** use original version of application (assumed to be correct)
  - **Gold Standard**
Our Oracle Process

Comparator

1. Expected HTML Response
2. Actual HTML Response
3. Customized HTML Processor
4. Processed Expected HTML Response
5. Processed Actual HTML Response
6. Pass?

- Pass/No Pass
- Set of Differences
Importance of Accurate Oracle

- Oracle is quality inspector
- Need oracle to be accurate
  - Allow only the applications that behave as expected to be deployed
- Consequences of inaccurate oracle
  - Buggy application is deployed
    - Customers may see errors
  - Correct application is not deployed
    - Not taking advantage of improvements
Evaluating Oracles

<table>
<thead>
<tr>
<th>Oracle Result (Is there a fault?)</th>
<th>Application Under Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Faulty Application</td>
</tr>
<tr>
<td>Positive (fault)</td>
<td>True Positive</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative (no fault)</td>
<td>False Negative</td>
</tr>
<tr>
<td></td>
<td>Missed a fault!</td>
</tr>
</tbody>
</table>

- **False positive**
  - Oracle says “fault” but application does not have a fault
- **False negative**
  - Oracle says “no fault” but application contains a fault
Evaluating Oracles

<table>
<thead>
<tr>
<th>Oracle Result (Is there a fault?)</th>
<th>Application Under Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive (fault)</td>
<td>False Positive</td>
</tr>
<tr>
<td>Missed a fault!</td>
<td>False Positive</td>
</tr>
<tr>
<td>False Positive</td>
<td>Look for a non-existent fault!</td>
</tr>
<tr>
<td>Negative (no fault)</td>
<td>True Negative</td>
</tr>
<tr>
<td>True Positive</td>
<td>False Positive</td>
</tr>
</tbody>
</table>

- HTML comparator algorithms
  - False negatives: for faults that do not show up in the HTML
Example of an HTML Response

```html
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body>
    <table border=0 cellspacing=0 width="580">
      <tr>
        <td rowspace=2>
          <img src="books.gif" width=310></td>
      </tr>
    </table>
    <!-- Sidebar Links -->
    <ul>
      <!-- Content -->
      <li><a href="alumni.html">Alumni</a></li>
    </ul>
    </body>
  </html>
```

- **Start tag**: `<html>`
- **Content**: `<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">` `<title>hiperspace lab</title>`
- **Style**: `<style>…</style>` `<script language="Javascript">…</script>`
- **Layout**: `<table border=0 cellspacing=0 width="580">` `<tr>` `<td rowspace=2>` `<img src="books.gif" width=310></td>` `</tr>` `</table>`
- **Comment**: `<!-- Sidebar Links -->`
- **Content**: `<ul>…</ul>` `<li><a href="alumni.html">Alumni</a></li>`
- **Attribute**: `<a href="alumni.html">Alumni</a>`
- **Close tag**: `</html>`
Comparing the HTML Output

**Expected**

```html
<html>
<head>
<title>My Page</title>
</head>
<body>
<h1>Intro</h1>
<p>Text…
<p>Today is January 10.
<a href="link.html">link</a>
<h2>Subsection</h2>
<form method=post …>…</form>
</body>
</html>
```

**Actual**

```html
<html>
<head>
<title>My Page</title>
</head>
<body>
<h1>Intro</h1>
<p>Text…
<p>Today is January 11.
<a href="link2.html">link</a>
<h2>Subsection</h2>
<form method=post …>…</form>
</body>
</html>
```
Comparison Algorithm: Document

- Diff entire document
  - Cheap, thorough

Packet 1:

```html
<html>
<head><title>My Page</title></head>
<body>
<h1>Intro</h1>
<p>Text…
<p>Today is January 10.
<a href="link.html">link</a>
<h2>Subsection</h2>
<form method=post ...
</form>
</body>
</html>
```

Packet 2:

```html
<html>
<head><title>My Page</title>
</head>
<body>
<h1>Intro</h1>
<p>Text…
<br>Today is January 11.
<a href="link2.html">link</a>
<h2>Subsection</h2>
<form method=post ...
</form>
</body>
</html>
```

Appearance fault

Link fault

Not a fault
Comparison Algorithm: Document

- Diff entire document
  - Cheap, thorough

```html
<html>
<head><title>My Page</title></head>
<body>
<h1>Intro</h1>
<p>Text…</p>
<p>Today is January 10.</p>
<a href="link.html">link</a>
<h2>Subsection</h2>
<form method=post ...>…</form>
</body>
</html>
```

```html
<html>
<head><title>My Page</title></head>
<body>
<h1>Intro</h1>
<p>Text…</p>
<br>Today is January 11.
<a href="link2.html">link</a>
<h2>Subsection</h2>
<form method=post ...>…</form>
</body>
</html>
```

- **Options**: ignore comments, some tags; collapse white space
Comparison Algorithm: Content

- Diff document’s text

<table>
<thead>
<tr>
<th>My Page Intro Text…</th>
<th>Today is January 10. Subsection</th>
</tr>
</thead>
<tbody>
<tr>
<td>My Page Intro Text…</td>
<td>Today is January 11. Subsection</td>
</tr>
</tbody>
</table>

- Misses link fault
- **Options**: collapse white space, ignore dates
Comparison Algorithm: Structure

- Diff document’s tags

```html
<html>
<head><title></title></head>
<body>
<h1></h1>
<p></p>
<p>
<a href="link.html">
<h2></h2>
<form method=post ...
</a>
<br>
<a href="link2.html">
<h2></h2>
<form method=post ...
</a>
</body>
</html>
```

- Miss errors in content

- Options: ignore closing tags, style, layout, unimportant attributes, all attributes, form options, order of links/imgs
## HTML Comparison Algorithms

False negatives: faults not manifested in HTML

<table>
<thead>
<tr>
<th>Comparison Algorithm</th>
<th>False Positives</th>
<th>False Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Dynamic, real-time changes</td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td>Dynamic, real-time changes</td>
<td>Errors in structure, e.g. forms</td>
</tr>
<tr>
<td>Structure</td>
<td>Display changes that do not affect app behavior</td>
<td>Errors in content</td>
</tr>
</tbody>
</table>
Partial Ordering of Implemented Comparison Algorithms

**Document-based oracles:**
- Document (D)
- DocumentBase (DB)
- DocumentBase-CollapsedWS (DB-W)

**Content-based oracles:**
- Content (C)
- Content-CollapsedWS (C-W)
- Content-CollapsedWS-Dates (C-WD)

**Structure-based classes:**
- Tags (T)
  - Forms (F)
  - TagNames+Impt Attrs (N+I)
    - TagNames (N)
    - UnorderedLinks (U)

More information
Fewer False Negatives
Fewer False Positives
Partial Ordering of Implemented Comparison Algorithms

- **Document-based**
  - **Structure-based classes:**
    - Tags (T)
      - Tags-Closing (T-C)
      - Tags-Style (T-S)
      - Tags-StyleLayout (T-SL)
    - By default: include closing tags, style, layout tags and attributes
  - Forms (F)
    - Forms-Select (F-S)
  - TagNames (N)
    - TagNames-Closing (N-C)
    - TagNames-Style (N-S)
    - TagNames-StyleLayout (N-SL)
  - TagNames+Impt Attrs (N+I)
    - TagNames+ImptAttrs -Style (N+I-S)
    - TagNames+ImptAttrs -StyleLayout (N+I-SL)
  - UnorderedLinks (U)
    - UnorderedLinks-Style (U-S)
    - UnorderedLinks-StyleLayout (U-SL)
Experimental Study

- **Goal**: oracle evaluation and recommendations to testers, researchers

- **Exp 1**: Effect of nondeterministic, real-time behavior
  - False positives

- **Exp 2**: Failure detection
  - Number & types of detected faults
  - Precision: $\frac{\text{# failures correct}}{\text{Total failures reported}}$ (higher means few false +)
  - Recall: $\frac{\text{# failures correct}}{\text{Expected # of failures}}$ (higher means few false -)
Methodology: Failure Detection Study

- Faults seeded manually into application code
- Various categories of seeded faults
  - data, logic, form, appearance, link

Test Suite ➞ Modified Web Application code ➞ requests ➞ responses ➞ HTML responses from Faulty App
Methodology: Evaluating Oracles

For the test suite of each subject application:

1. Generate failure detection reports for each fault class using each oracle.
2. Replay suite on clean and fault-seeded version(s) of code.

Comparator’s Failure Detection Report:

<table>
<thead>
<tr>
<th>Request/Response</th>
<th>Failure? (Yes/No)</th>
</tr>
</thead>
</table>

Subject Applications

Four deployed subject applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Test Cases</th>
<th>Requests</th>
<th>NCLOC*</th>
<th>Faults Exposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masplas</td>
<td>169</td>
<td>1103</td>
<td>999</td>
<td>22</td>
</tr>
<tr>
<td>E-Commerce Bookstore (Book)</td>
<td>125</td>
<td>3564</td>
<td>7791</td>
<td>36</td>
</tr>
<tr>
<td>Course Project Manager (CPM)</td>
<td>890</td>
<td>12352</td>
<td>9300</td>
<td>96</td>
</tr>
<tr>
<td>DSpace</td>
<td>75</td>
<td>3183 (3023 HTML pages)</td>
<td>49513</td>
<td>20</td>
</tr>
</tbody>
</table>

*NCLOC: Non-comment Lines of Code

Total: 174 Faults
How to Interpret Results

Legend:
- Horizontal Bar: Median
- + : Mean
- Box: Inner quartile region (IQR) -- inner 50% of data
- Vertical line: 1.5 * IQR

Point in distribution: comparator’s precision or recall for 1 fault
Failure Detection Results

- Precision: light gray, left
- Recall: dark gray, right
- Across all applications, all fault types
  - Individual graphs in Sprenkle thesis

- Tradeoff between precision and recall
- Trends follow hierarchy
  - Top of hierarchy: higher recall, lower precision
  - Leaves of hierarchy: lower recall, higher precision
Combinations of Oracles

- **Key Insight**: combine best oracles that focus on disparate parts of the HTML document to improve effectiveness over individual comparators
  - Union: improves recall, may affect precision
  - Intersection: improves precision, decreases recall

- Empirically best combination, across all apps: **C-WD ∪ N+I**
Guidance to Testers

- Execute test suite several times
  - Identify responses with nondeterministic, real-time behavior
  - Identify comparators with fewest false positives
- Vary comparator by response’s behavior
  - Deterministic: use Document
  - Nondeterministic: use C-WD $\cup$ N+I
- Use more precise oracle in early testing stages
  - Don’t overwhelm testers with failure reports
  - Flesh out bugs w/o sifting through false positives
- **Document** family for fewest false negatives
- **Forms-Select** for fewest false positives w/ highest recall
Related Work

- Model-based testing of web applications
  - Ricca01, Liu00, Deng04
  - Andrews05: suggests partial oracles
- HttpUnit
  - Testers create oracles manually
- HTML-based comparators
  - Elbaum05, DiLucca02 -- few details of implementation, no evaluation of false positives, negatives
Conclusions and Future Work

- Proposed, evaluated a suite of customized, HTML-based oracle comparators
- Recommendations to testers for selecting oracle
  - Most effective oracle in general: Content-CollapsedWS-Dates ∪ TagNames+ImptAttrs
- Learning effective oracle comparator combinations: STEV2007
  - Train on faults from bug reports
- Future work
  - Fully customizable comparator
  - Evaluate with more applications, faults