Automating Presentation Changes in Dynamic Web Applications via Collaborative Hybrid Analysis

Xiaoyin Wang*
Lu Zhang
Tao Xie
Yingfei Xiong
Hong Mei

UC Berkeley
Peking University
NC State University
Peking University

* This work was conducted when Xiaoyin Wang was at Peking University.
Dynamic Web Application

- Server code generates HTML page according to user inputs
Presentation Changes

• A common task in web application development
  ✓ Correcting display error or HTML syntax error
  ✓ Adding interface decorations
  ✓ Changing appearance styles

• 7% of 600 bug reports investigated are presentation changes
Challenges

• Presentation changes are often identified and reported on the generated HTML pages

• Developers have to modify the server-side code accordingly
Challenges

Generated web page:

```html
<p2>
  name: <input id = 1 color = BFFFFF value = “default”></input></div>
country: <input id = 2 color = BFFFFF value = “country”></input>
age: <input id = 3 color = BFFFFF value = “age”></input>
</p2>
```

Code generating the web page

```php
$color = BFFFFF; echo “<p2>”; echo “<tr”;
  echo “name: “; $id++;
  echo “<input id =”.$id.” color = “.”.color.” value = “default”></input>”;
  $id++;
  echo “country: “; $id++;
  echo “<input id =”.$id.” color = “.”.color.” value = “country”></input>”;
  $id++;
  echo “age: “; $id++;
  echo “<input id =”.$id.” color = “.”.color.” value = “age”></input>”;
  $id++;
  echo “</p2>”;
```

Too common for text search

Generation code may scatter

Affect multiple places
Outline

• Motivation
• Approach
• Empirical Study
• Discussion
Usage Scenario

Runtime / Server

Server Code

Our tool

Auto fixed

Needs intervention at code position xxx

HTML page

Identify Change

Developer
Approach Overview: Collaborative Hybrid Analysis

• Dynamic String Taint Analysis
  – Locate the piece of code to change

• Static Unexpected Impact Detection
  – Check whether the change is safe
    Safe: perform the change automatically
    Unsafe: report the location to the user
Dynamic String Taint Analysis

• Based on the idea of trace-based bidirectionalization [Xiong et al., ASE07]
  ➢ Add a position tag to each constant string and input string

  ![Diagram](image.png)

  $x = "<tr>"

  $y = $x

  Copy the tags together with the strings

  ![Diagram](image2.png)

  Propagate through string operations

  ✓ Concatenation
String Operation Handling

- Problem: do we need to reimplemenet all string operations?
- Solution: working with finite state transducer [Wassermann and Su, PLDI’07]

Constant string A, B, C
String variable $x, y$
$y = B.C$
replace($x, A, y$)

Automatically generated FST with position tag output, based on the runtime value of $y$, $T = \Sigma^* / A\Sigma^*$
Unexpected Impacts

• Inner-page impacts
String origin to be changed affects multiple places in the generated page

• Inter-page impacts
String origin to be changed affects other pages, or contents not generated in this execution
Checking unexpected impacts

- Inner-page impacts
  Checking all locations sharing the same string origin are changed consistently

- Inter-page impacts
  Checking whether any unexecuted code data-dependent or control dependent on the changed code
Practical Issues

• Insertion:
  ✓ When a change requires insertion between two variables, human intervention is required
  ✓ Example:

Code:
$title = “contact”; 
echo “<td>”.$title. “</td>”

HTML:
<td>contact</td>

• Non-constant string origin
  ✓ When a string origin is not constant (thus cannot be changed directly), human intervention is required
Outline

• Motivation
• Approach
• Empirical Study
• Discussion
Study on the bug reports of three web applications

- 600 Bug Reports from the early history of 3 popular PHP web projects: SquirrelMail, OrangeHRM, and WebCalendar

<table>
<thead>
<tr>
<th>Project</th>
<th>Start (MM/YY)</th>
<th>End (MM/YY)</th>
<th>KLoc</th>
<th>#Bug Reports</th>
<th>#PC Bug Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>SquirrelMail</td>
<td>04/00</td>
<td>12/01</td>
<td>8-26</td>
<td>200</td>
<td>7</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>06/00</td>
<td>12/02</td>
<td>6-17</td>
<td>200</td>
<td>14</td>
</tr>
<tr>
<td>OrangeHRM</td>
<td>03/06</td>
<td>10/06</td>
<td>96-105</td>
<td>200</td>
<td>22</td>
</tr>
</tbody>
</table>

PC Bug Reports: Presentation Change related Bug Reports
Are presentation changes trivial?

- Comparison of processing days between PC Bug Reports and All Bug Reports
- Presentation changes are not trivial (similar processing days compared with other bug reports)

<table>
<thead>
<tr>
<th>Project / Processing Days</th>
<th>PC Bug Reports</th>
<th>All Bug Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>Range</td>
</tr>
<tr>
<td>SquirrelMail</td>
<td>59.3</td>
<td>0-248</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>44.3</td>
<td>0-230</td>
</tr>
<tr>
<td>OrangeHRM</td>
<td>20.1</td>
<td>1-51</td>
</tr>
</tbody>
</table>
Evaluating our approach

- **Dataset**: 39 presentation change tasks (from 43 reports, in which 4 are duplicate)

- **Evaluation Oracle**: developers’ changes

- **Research Questions**:
  - ✓ How effective is our approach on finding the source locations to change?
  - ✓ How effective is our approach on detecting unexpected impacts?
## Evaluation Results

<table>
<thead>
<tr>
<th>Categories</th>
<th>Number of tasks</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td># Correctly Located</td>
<td>39</td>
<td>100.0%</td>
</tr>
<tr>
<td># Automatically fixed</td>
<td>23</td>
<td>59.0%</td>
</tr>
<tr>
<td># Matched fixes</td>
<td>20</td>
<td>51.3%</td>
</tr>
<tr>
<td># Unmatched fixes</td>
<td>3</td>
<td>7.7%</td>
</tr>
<tr>
<td># Human Intervention Required</td>
<td>16</td>
<td>41.0%</td>
</tr>
<tr>
<td># inner-page impact</td>
<td>1</td>
<td>2.6%</td>
</tr>
<tr>
<td># inter-page impact</td>
<td>3</td>
<td>7.7%</td>
</tr>
<tr>
<td># insertions</td>
<td>6</td>
<td>15.4%</td>
</tr>
<tr>
<td># changing non-constants</td>
<td>6</td>
<td>15.4%</td>
</tr>
</tbody>
</table>

Our approach correctly locates all source origins.
Evaluation Results

<table>
<thead>
<tr>
<th>Categories</th>
<th>Number of tasks</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td># Correctly Located</td>
<td>39</td>
<td>100.0%</td>
</tr>
<tr>
<td># Automatically fixed</td>
<td>23</td>
<td>59.0%</td>
</tr>
<tr>
<td># Matched fixes</td>
<td>20</td>
<td>51.3%</td>
</tr>
<tr>
<td># Unmatched fixes</td>
<td>3</td>
<td>7.7%</td>
</tr>
<tr>
<td># Human Intervention Required</td>
<td>16</td>
<td>41.0%</td>
</tr>
<tr>
<td># inner-page impact</td>
<td>1</td>
<td>2.6%</td>
</tr>
<tr>
<td># inter-page impact</td>
<td>3</td>
<td>7.7%</td>
</tr>
<tr>
<td># insertions</td>
<td>6</td>
<td>15.4%</td>
</tr>
<tr>
<td># changing non-constants</td>
<td>6</td>
<td>15.4%</td>
</tr>
</tbody>
</table>

Most automatic changes match the oracles, yet some do not.
Unmatched Auto-fix

Bug Report No. 1510677 of OrangeHRM
“Feedback information of an operation should be in green when the operation succeeds”

Our approach changed “#FF0000” (red) to “#005500” (green).

Developer change added a check for whether the operation succeeds, and then set different colors.

Other unmatched fixes added similar new behavior to the code
Evaluation Results

<table>
<thead>
<tr>
<th>Categories</th>
<th>Number of tasks</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td># Correctly Located</td>
<td>39</td>
<td>100.0%</td>
</tr>
<tr>
<td># Automatically fixed</td>
<td>23</td>
<td>59.0%</td>
</tr>
<tr>
<td># Matched fixes</td>
<td>20</td>
<td>51.3%</td>
</tr>
<tr>
<td># Unmatched fixes</td>
<td>3</td>
<td>7.7%</td>
</tr>
<tr>
<td># Human Intervention Required</td>
<td>16</td>
<td>41.0%</td>
</tr>
<tr>
<td># inner-page impact</td>
<td>1</td>
<td>2.6%</td>
</tr>
<tr>
<td># inter-page impact</td>
<td>3</td>
<td>7.7%</td>
</tr>
<tr>
<td># insertions</td>
<td>6</td>
<td>15.4%</td>
</tr>
<tr>
<td># changing non-constants</td>
<td>6</td>
<td>15.4%</td>
</tr>
</tbody>
</table>

For the rest of the tasks, our approach correctly identifies the need of human intervention.
Outline

• Motivation
• Approach
• Empirical Study
• Discussion
Limitations

• More suitable for small atomic changes than pervasive or large structure changes

• Currently cannot handle web interface generated with Ajax techniques

• May generate undesirable code changes
Conclusion

• Presentation change being common and non-trivial
• Hybrid approach to presentation changes
  – Dynamic analysis to locate the source code to change
  – Static analysis to ensure the change is safe
• Lightweight approach yet effective
Thanks! Q & A
Evaluation Results

- On locating source code and automatic fixing

<table>
<thead>
<tr>
<th>Project</th>
<th>#PC tasks</th>
<th>#Locating</th>
<th>#matched auto-fix</th>
<th>#unmatched auto-fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>SquirrelMail</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>12</td>
<td>12</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>OrangeHRM</td>
<td>21</td>
<td>21</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>39</td>
<td>39</td>
<td>20</td>
<td>3</td>
</tr>
</tbody>
</table>
Evaluation Results

- On detecting unexpected impacts and practical issues

<table>
<thead>
<tr>
<th>Project</th>
<th>#PC tasks</th>
<th>#inner-page Impact</th>
<th>#inter-page impact</th>
<th>#insert</th>
<th>#non-constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>SquirrelMail</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>OrangeHRM</td>
<td>21</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>39</strong></td>
<td><strong>1</strong></td>
<td><strong>3</strong></td>
<td><strong>6</strong></td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>
Example Task
SquirrelMail ---- Bug #601006: “Rejected e-mail link missing a quote”

Error HTML page:
<br><strike><a href="mailto:mymail@gmail.com?subject=WebCalendar:mycal\>Xiao</a></strike>Rejected";

Buggy Code:
echo "<br><strike><a href="\mailto:" . $tempemail . ">
subject=$subject\>" . $tempfullname . "</a></strike> (" .
translate("Rejected") . "\n";

Result of our tool
1. Locate the “\>” in the code as the data origin of the erroneous place in the error HTML page
2. Determine that there is no unexpected impacts and practical issues, so that the fix can be done automatically
Example Task

SquirrelMail ---- Bug #601006: “Rejected e-mail link missing a quote”

Error HTML page:
<BR><STRIKE><A HREF="mailto:mymail@gmail.com?
subject=WebCalendar:mycal\>Xiao</a></STRIKE>Rejected";

Buggy Code:
echo "<BR><STRIKE><A HREF="mailto:" . $tempemai . ">
subject=$subject\>" . $tempfullname . "</a></STRIKE> (" .
translate("Rejected") . "\ n";

Result of our tool
1. Locate the “\>” in the code as the data origin of the erroneous place
   in the error HTML page
2. Determine that there is no unexpected impacts and practical issues,
   so that the fix can be done automatically
Future Directions

• Empirical studies on more web-based projects

• Handling of more complex presentation techniques, e.g., Ajax

• User study on how much the approach it going to help in real maintenance tasks
Dynamic String Taint Analysis

• Based on the idea of trace-based bidirectionalization [Xiong et al., ASE07]

➢ Instrumentation
  Add a position tag to each constant string and input string

➢ Propagate through string operations
  ✓ Concatenation

✓ Other Operation
  Simulated with Finite State Transducer [Wassermann and Su, PLDI’07]
String Operation Handling

Constant string A, B, C
String variable $x, $y
$y = B.C
replace($x, A, $y)

Consider $A = ‘ts’, $x = ‘abct’(tag1) ‘sdd’(tag2)
Output = ‘abc’(tag1’)B(tagB)C(tagC) ‘dd’(tag2)