Vejovis: Suggesting Fixes for JavaScript Faults

Frolin S. Ocariza, Jr.
Karthik Pattabiraman, Ali Mesbah

University of British Columbia
Problem and Motivation

JavaScript in web applications has plenty of reliability issues

JS faults are not trivially fixed [ESEM’13] issues [ESEM’13]
Problem and Motivation

JavaScript in web applications has plenty of reliability issues, these JavaScript faults matter and these JavaScript faults are non-trivial to fix.
Faults in JavaScript Code

- Study of JS bug reports [ESEM’13]

- **Key Insight:** Most (65%) mistakes programmers make in JS propagate to **parameters of DOM API method calls**
  - DOM API methods: `getElementById`, `getElementsByTagName`, jQuery’s `($)`, etc.
  - We also found that such faults are the most impactful, and take the longest to fix

**DOM-RELATED FAULTS**
DOM-Related Fault Example

```javascript
var x = "yes";
var elem = document.getElementById(x);
```
DOM-Related Fault Example

```javascript
var x = "no";
var elem = document.getElementById(x);
```

MISTAKE!
DOM-Related Fault Example

```javascript
var x = "no";
var elem = document.getElementById(x);
```

**MISTAKE!**

ID parameter evaluates to “no”, which is not in the DOM
Goal

Facilitate the process of fixing DOM-related faults
Fault Model

- Suggest repairs for DOM-related faults
- Only one mistake made
Common Developer Fixes

- Study of 190 fixed bug reports from 12 web apps

```javascript
elem = getElementById(param)
elem.innerHTML = "…"
```
Common Developer Fixes

- Study of 190 fixed bug reports from 12 web apps

Modify the parameter

elem = getElementById(new_param)

elem.innerHTML = “…”

Ways Programmers Fix Faults
- Parameter Modification
Common Developer Fixes

- Study of 190 fixed bug reports from 12 web apps

```javascript
elem = getElementById(param)
if (elem)
    elem.innerHTML = "..."
```

Ways Programmers Fix Faults

- Parameter Modification
- DOM Element Validation
Common Developer Fixes

- Study of 190 fixed bug reports from 12 web apps

```
elem = querySelector(param)
attr.innerHTML = “…”
```

Ways Programmers Fix Faults
- Parameter Modification
- DOM Element Validation
- Method Modification

Modify the method
Common Developer Fixes

- Study of 190 fixed bug reports from 12 web apps

```javascript
elem = getElementById(param)
elem.innerHTML = "…"
```

Ways Programmers Fix Faults

- Parameter Modification 27.2%
- DOM Element Validation 25.7%
- Method Modification 24.6%
Structure in DOM Method Parameters

<table>
<thead>
<tr>
<th>WRONG</th>
<th>RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getElementById(&quot;no&quot;)</code></td>
<td><code>???</code></td>
</tr>
</tbody>
</table>

**Question**: How do we know that we should replace “no” with “yes”?

**Answer**: We need to infer programmer *intent*
- Very difficult to do in general, but…
- We have the DOM!
Structure in DOM Method Parameters

**WRONG**

```javascript
getElementById("no")
```

**RIGHT**

```javascript
getElementById("yes")
```

**Question**: How do we know that we should replace “no” with “yes”

**Answer**: We need to infer programmer *intent*
  - Very difficult to do in general, but…
  - We have the DOM!
CSS Selectors

\[ \text{div} \# \text{sample} \text{ > table tr.hello} \]

- Tag name “div”
- ID “sample”
- Tag name “table”
- Tag name “tr”
- Is child
- Is descendant
- Class name “hello”

Input to \texttt{querySelector()}, \\$(\), etc. to retrieve list of elements
In this section, we describe our approach for assisting web developers in repairing DOM-related faults satisfying the fault model described in the previous section. Figure 3 shows a block diagram of our design.

1. **Data Collector** (box a): This component collects data about the symptoms and possible sicknesses associated with DOM-related faults from the web application or URL.
2. **Symptom Analyzer** (box b): This component analyzes the data collected by the Data Collector to identify potential sicknesses related to the symptoms.
3. **Treatment Suggester** (box c): This component suggests possible treatments or workarounds for the identified sicknesses, generating a list of suggestions.

The design includes modules for direct DOM access, symptom data, symptoms, data collector, symptom analyzer, supplementary information, possible sicknesses, and list of workaround suggestions.
Running Example

1. `firstTag = "div";`
2. `prefix = "pain-";`
3. `suffix = "elem";`
4. `level1 = firstTag + "#" + prefix + suffix;`
5. `level2 = "span.cls";`
6. `e = $(level1 + " " + level2);`
7. `e[0].innerHTML = "new content";`
Running Example

1 `firstTag = "div";`
2 `prefix = "pain-";`
3 `suffix = "elem";`
4 `level1 = firstTag + "#" + prefix + suffix;`
5 `level2 = "span.cls";`
6 `e = $(level1 + " " + level2);`
7 `e[0].innerHTML = "new content";`

Lines to set up the CSS selector passed to `$(())`
Running Example

1 firstTag = “div”;
2 prefix = “pain-”;
3 suffix = “elem”;
4 level1 = firstTag + “#” + prefix + suffix;
5 level2 = “span.cls”;
6 e = $(level1 + “ “ + level2);
7 e[0].innerHTML = “new content”;

Constructed selector: div#pain-elem span.cls
Running Example

1. `firstTag = "div";`
2. `prefix = "pain-";`
3. `suffix = "elem";`
4. `level1 = firstTag + "#" + prefix + suffix;`
5. `level2 = "span.cls";`
6. `e = $(level1 + " " + level2);`
7. `e[0].innerHTML = "new content";`

Constructed selector: `div#pain-elem span.cls`
## Running Example

1. \texttt{firstTag = "div"};
2. \texttt{prefix = "pain-"};
3. \texttt{suffix = "elem"};
4. \texttt{level1 = firstTag + "#" + prefix + suffix};
5. \texttt{level2 = "span.cls"};
6. \texttt{e = $(level1 + " " + level2)};
7. \texttt{e[0].innerHTML = "new content"};

**Constructed selector:** \texttt{div#pain-elem span.cls}
Main Idea

- **Parameter Analysis**: What portion of the parameter do we replace?
- **Context Analysis**: How do we perform the replacement in the code?
Parameter Analysis: Dividing Components

Invalid selector: div#pain-elem span.cls

Divide into components

div | # | pain-elem | | span | . | cls
Parameter Analysis: Dividing Components

Invalid selector: div#pain-elem span.cls

Divide into components

div  |  #  |  pain-elem  |  |  span  |  .  |  cls

tag  |  has-id  |  id  |  has-descendant  |  tag  |  has-class  |  class
Parameter Analysis: Dividing Components

Invalid selector: div#pain-elem span.cls

Subdivide each component according to dynamic backward slice

```
div | # | pain- | elem | | span | . | cls
```

Line 2

Line 3
Parameter Analysis: Dividing Components

```
1 firstTag = "div";
2 prefix = "pain-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2);
7 e[0].innerHTML = "new content";
```

Invalid selector: `div#pain-elem span.cls`

Diagram:
```
  div
     |      |
     v      v
  {Id = "main-elem"}
        |      |
        v      v
  div    span
     |      |   |
     v      v   v
  {Id = "wrapper"}  {class="cls"}
        |      |
        v      v
  span    span
          |      |
          v      v
        {class="cls"}
```
Parameter Analysis: Dividing Components

Invalid selector: \texttt{div\#pain-elem span.cls}

Subdivide each component according to dynamic backward slice

\texttt{div | \# | pain- | elem | | span | . | cls}
Parameter Analysis: Finding Valid Selectors

Invalid selector: \texttt{div\#pain-elem span.cls}

Construct VALID selectors from current DOM that are “sufficiently close” to the erroneous one.

```
div
  Id = “main-elem”
  div
    Id = “wrapper”
    span
      class=“cls”
  span
    class=“cls”
  span
    class=“cls”
```

30
Parameter Analysis: Finding Valid Selectors

Invalid selector: div#pain-elem span.cls

List of valid selectors:

div#main-elem span.cls

div#wrapper span.cls

Construct VALID selecters from current DOM that are “sufficiently close” to the erroneous one
Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: div#pain-elem span.cls

div | # | pain- | elem | | span | . | cls
Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: div#pain-elem span.cls

div | # | pain- | elem | | span | . | cls

Assumed incorrect
Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: \texttt{div\#pain-elem span.cls}

\begin{itemize}
  \item \texttt{div
  \item \texttt{#}
  \item \texttt{elem
  \item \texttt{span
  \item \texttt{.}
  \item \texttt{cls}
\end{itemize}

\textbf{List of valid selectors:}

- \texttt{div\#main-elem span.cls}
- \texttt{div\#wrapper span.cls}

Use as pattern
Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: `div#pain-elem span.cls`

```
div | # | elem | span | . | cls
```

List of valid selectors:

1. `div#main-elem span.cls` – MATCHES PATTERN!
2. `div#wrapper span.cls`
Context Analysis

1. firstTag = “div”;
2. prefix = “pain-”;
3. suffix = “elem”;
4. level1 = firstTag + “#” + prefix + suffix;
5. level2 = “span.cls”;
6. e = $(level1 + “ “ + level2);
7. e[0].innerHTML = “new content”;

Invalid selector: div#pain-elem span.cls
Replacement selector: div#main-elem span.cls
Context Analysis

1 firstTag = "div";
2 prefix = "main-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2);
7 e[0].innerHTML = "new content";

Invalid selector: div#pain-elem span.cls
Replacement selector: div#main-elem span.cls
Context Analysis

```javascript
1  firstTag = “div”;
2  prefix = “main-”;  // String literal replaced
3  suffix = “elem”;
4  level1 = firstTag + “#” + prefix + suffix;
5  level2 = “span.cls”;
6  e = $(level1 + “ “ + level2);
7  e[0].innerHTML = “new content”;
```

Invalid selector: `div#pain-elem span.cls`
Replacement selector: `div#main-elem span.cls`

Message:
REPLACE STRING LITERAL “pain-” in line 2 with string literal “main-”
Context Analysis: Non-“Replace” Messages

- Loops – “replace” may be unsafe
- String value doesn’t originate from string literal

**Analyze the context!**

<table>
<thead>
<tr>
<th>MESSAGE TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE</td>
</tr>
<tr>
<td>REPLACE AT ITERATION</td>
</tr>
<tr>
<td>OFF BY ONE AT BEGINNING</td>
</tr>
<tr>
<td>OFF BY ONE AT END</td>
</tr>
<tr>
<td>MODIFY UPPER BOUND</td>
</tr>
<tr>
<td>EXCLUDE ITERATION</td>
</tr>
<tr>
<td>ENSURE</td>
</tr>
</tbody>
</table>
Implementation

- **Vejovis**


- Data collection: Rhino and Crawljax
- Pattern matching: Hampi
Usage Model

INPUTS

URL

VEJOVIS

DOM METHOD LOCATION

AUTOFLOX

OUTPUT

LIST OF ACTIONABLE REPAIR MESSAGES
RQ1: What is the accuracy of Vejovis in suggesting a correct repair?

RQ2: How quickly can Vejovis determine possible replacements? What is its performance overhead?
RQ1: Accuracy of Vejovis

<table>
<thead>
<tr>
<th>Subjects</th>
<th>JS Code Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal</td>
<td>213</td>
</tr>
<tr>
<td>Ember.js</td>
<td>745</td>
</tr>
<tr>
<td>Joomla</td>
<td>434</td>
</tr>
<tr>
<td>jQuery</td>
<td>94</td>
</tr>
<tr>
<td>Moodle</td>
<td>352</td>
</tr>
<tr>
<td>MooTools</td>
<td>101</td>
</tr>
<tr>
<td>Prototype</td>
<td>164</td>
</tr>
<tr>
<td>Roundcube</td>
<td>729</td>
</tr>
<tr>
<td>TYPO3</td>
<td>2252</td>
</tr>
<tr>
<td>WikiMedia</td>
<td>160</td>
</tr>
<tr>
<td>WordPress</td>
<td>197</td>
</tr>
</tbody>
</table>

- 22 bug reports (2 per app, and randomly chosen)
- Replicated bug and ran with Vejovis

**Recall and Precision**

**RECALL**: 100% if correct fix appears; 0% otherwise

**PRECISION**: Measure of extraneous suggestions
## RQ1: Recall

<table>
<thead>
<tr>
<th>Subject</th>
<th>Bug Report #1</th>
<th>Bug Report #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Ember.js</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Joomla</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>jQuery</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Moodle</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>MooTools</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Prototype</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Roundcube</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>TYPO3</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>WikiMedia</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>WordPress</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

**Overall Recall: 91%**
## RQ1: Precision

<table>
<thead>
<tr>
<th>Subject</th>
<th>Bug Report #1</th>
<th>Bug Report #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal</td>
<td>3%</td>
<td>25%</td>
</tr>
<tr>
<td>Ember.js</td>
<td>50%</td>
<td>33%</td>
</tr>
<tr>
<td>Joomla</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>jQuery</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>Moodle</td>
<td>3%</td>
<td>3%</td>
</tr>
<tr>
<td>MooTools</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>Prototype</td>
<td>17%</td>
<td>50%</td>
</tr>
<tr>
<td>Roundcube</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>TYPO3</td>
<td>1%</td>
<td>100%</td>
</tr>
<tr>
<td>Wikimedia</td>
<td>4%</td>
<td>1%</td>
</tr>
<tr>
<td>WordPress</td>
<td>3%</td>
<td>1%</td>
</tr>
</tbody>
</table>

**Avg. Precision:** 2%

**49 suggestions** per bug on average!

**Improvements**

1. Edit distance bound
2. Ranked suggestions
## Alternative: Ranking

<table>
<thead>
<tr>
<th>Subject</th>
<th>Bug Report #1</th>
<th>Bug Report #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal</td>
<td>31 / 40</td>
<td>1 / 4</td>
</tr>
<tr>
<td>Ember.js</td>
<td>1 / 2</td>
<td>1 / 3</td>
</tr>
<tr>
<td>Joomla</td>
<td>1 / 88</td>
<td>1 / 88</td>
</tr>
<tr>
<td>jQuery</td>
<td>2 / 108</td>
<td>-</td>
</tr>
<tr>
<td>Moodle</td>
<td>2 / 37</td>
<td>1 / 37</td>
</tr>
<tr>
<td>MooTools</td>
<td>2 / 2</td>
<td>1 / 2</td>
</tr>
<tr>
<td>Prototype</td>
<td>1 / 6</td>
<td>1 / 2</td>
</tr>
<tr>
<td>Roundcube</td>
<td>4 / 79</td>
<td>-</td>
</tr>
<tr>
<td>TYPO3</td>
<td>1 / 187</td>
<td>1 / 1</td>
</tr>
<tr>
<td>Wikimedia</td>
<td>6 / 24</td>
<td>1 / 71</td>
</tr>
<tr>
<td>WordPress</td>
<td>13 / 30</td>
<td>1 / 170</td>
</tr>
</tbody>
</table>

#1 Ranking in 13 out of 20 bugs

Conservative ranking

Ranking seems to be beneficial
RQ2: Performance

- Takes average of 44 seconds to find correct fix
- Worst case: 91.1 seconds (Joomla)
Threats to Validity

- **External**: Evaluated on 11 web apps
- **Internal**: Took bugs from earlier empirical study
Conclusion

- Vejovis: replacement suggestor for DOM-related faults
  - Project Link: http://ece.ubc.ca/~frolino/projects/vejovis
- Evaluated on 22 real-world bugs
  - Good recall – 91%
  - Correct fix ranked #1 in 13/20 cases
  - Average 44 s to complete