JavaScript Errors in Web Applications: Understanding, Fixing and Prevention

Karthik Pattabiraman
Frolin Ocariza, Saba Alimadadi, Kartik Bajaj, Sheldon Sequira, Ali Mesbah

University of British Columbia (UBC)
Overall Goal & approach

- **Goal**: Make JavaScript Applications easy to program
- **Challenge**: JavaScript is a difficult language to analyze
- **Approach**: Perform dynamic analysis of JavaScript applications

Can we make JavaScript-based web applications robust and easy to program?
Talk Outline

• Our Prior Work: DOM-Related Faults [ESEM’13]

• AutoFlox: Automatically Localizing JavaScript Faults [ICST’12]

• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]

• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award

• Dompletion: Code completion for JavaScript [ASE’14]

• Conclusions & Ongoing work

• Open Challenges
Our Prior Work [ESEM’13]

• Bug report study of 12 applications: JavaScript faults
  – Over 300 bug reports analyzed; only fixed bugs considered

• DOM-related faults dominate JavaScript faults
  – Responsible for nearly two-thirds of all faults
  – Responsible for 80% of highest impact faults
  – Take 50% longer time to fix for developers

• Need low-cost solutions for DOM-related faults
  – Focus of this part of the tutorial
DOM-Related Faults

Code accesses non-existent element – returns null (bug)
Talk Outline

• Our Prior Work: DOM-Related Faults [ESEM’13]

• AutoFlox: Automatically Localizing JavaScript Faults [ICST’12]

• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]

• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award

• Dompletion: Code completion for JavaScript [ASE’14]

• Conclusions & Ongoing work

• Open Challenges
AutoFlox: Motivation

- What to do after we find errors? Need to fix them
- Fault localization: Find the root cause of the error
  - Focus on DOM-related JavaScript errors
AutoFlox: Example (Tumblr)

- Show a banner that cycles through four images every 5s

```javascript
function changeBanner(bannerID) {
  clearTimeout(changeTimer);
  changeTimer = setTimeout(changeBanner, 5000);

  prefix = "banner_";
  currBannerElem = document.getElementById(prefix+currentBannerID);
  bannerToChange = document.getElementById(prefix + bannerID);
  currBannerElem.removeClassName("active");
  bannerToChange.addClass("active");
  currentBannerID = bannerID;
}

currentBannerID = 1;
changeTimer = setTimeout(changeBanner, 5000);
```

- Passed with no argument (even though changeBanner needs one argument)
- BannerID will be set to undefined
- Would return null
- Async exception

Copyright: Karthik Pattabiraman, 2014
AutoFlox: Approach

• Identify faulty DOM access through dynamic tracing
• Focus on errors due to **DOM-JavaScript** interactions

![AutoFlox Flowchart]

1. **Web App**
2. **Instrument JS code**
3. **Run Web Application**
4. **Generate Traces**
5. **Extract backward slice**
6. **Construct relevant sequence**
7. **Partition into sequences**
8. **Faulty DOM access**

Trace collection

Trace analysis

Copyright: Karthik Pattabiraman, 2014
AutoFlox: Experimental Setup

RQ1: Is AutoFlox effective at finding DOM-related JS errors?

Experiment: Injected faults by mutating DOM accessor methods. Compared AutoFlox output with the injected line – match equals success

RQ2: How fast is AutoFlox at finding DOM-related JS faults?
### AutoFlox: Results - RQ1

<table>
<thead>
<tr>
<th>Web Application</th>
<th>Successful Detections</th>
<th>Failed Detections</th>
<th>Percent Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskFreak</td>
<td>39</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>WordPress</td>
<td>14</td>
<td>3</td>
<td>82.4%</td>
</tr>
<tr>
<td>ChatJavaScript</td>
<td>10</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>TUDU</td>
<td>9</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>JSScramble</td>
<td>6</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>JavaScript-Todo</td>
<td>2</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>80</strong></td>
<td><strong>3</strong></td>
<td><strong>96.4%</strong></td>
</tr>
</tbody>
</table>
AutoFlox: Results - RQ2

• **Approach:** Measure trace collection overhead
  – Tumblr website to localize example fault
  – Successfully localized the fault

• **Results**
  – Trace collection incurred 35% overhead
  – Trace analysis took 0.115 seconds to complete
AutoFlox: Summary

• Fault localization for JavaScript is challenging

• About 67% of JavaScript bugs occur due to DOM interactions – DOM-related JS Fault

• AutoFlox uses dynamic backward slicing to successfully isolate > 90% of injected faults
  – Real error from tumblr.com localized
Talk Outline

• Our Prior Work: DOM-Related Faults [ESEM’13]
• AutoFlox: Automatically localizing JavaScript Faults [ICST’12]
• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]
• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award
• Dompletion: Code completion for JavaScript [ASE’14]
• Conclusions & Ongoing work
• Open Challenges
Vejovis: Motivation

- **Goal:** Automatically “fix” DOM-related faults

- **Challenges:**
  1. Dynamic nature of JavaScript
  2. Dynamic nature of DOM
  3. Interaction between two languages

- **Approach:** Find symptoms to determine problem
  - Suggest workarounds to get rid of symptoms
  - Workaround patterns based on “common fixes”
Vejovis: Common FIX Patterns

- Parameter Modification: 27%
- DOM Element Validation: 26%
- Method Modification: 25%
- Major Refactoring/Other: 22%

What we support

Copyright: Karthik Pattabiraman, 2014
**Vejovis: Main Idea**

**WRONG**

getElementById("no")

**RIGHT**

generateElementById("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 can use the DOM’s structure
Vejovis: 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";

Would return empty set!

Constructed selector: div#pain-elem span.cls
Vejovis: Approach

- Divide selector components according to backward slice
- Find valid and selectors “sufficiently close” to the erroneous one, using the DOM

List of valid selectors:
- `div#main-elem span.cls`
- `div#wrapper span.cls`

- Match each valid selector with pattern based on divided selector components using an SMT Solver
Vejovis: Fix Classes

- Determine **action** for programmer to replace wrong selectors with valid selectors in the JS code
- Use constraint solver (hampi) to find the best action

### Fix Patterns

<table>
<thead>
<tr>
<th>Fix Patterns</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>

**Example:** REPLACE string literal “pain-” in line 2 with string “main-”
# Vejovis: Evaluation

<table>
<thead>
<tr>
<th>Web Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal</td>
</tr>
<tr>
<td>Ember.js</td>
</tr>
<tr>
<td>Joomla</td>
</tr>
<tr>
<td>jQuery</td>
</tr>
<tr>
<td>Moodle</td>
</tr>
<tr>
<td>MooTools</td>
</tr>
<tr>
<td>Prototype</td>
</tr>
<tr>
<td>Roundcube</td>
</tr>
<tr>
<td>TYPO3</td>
</tr>
<tr>
<td>WikiMedia</td>
</tr>
<tr>
<td>WordPress</td>
</tr>
</tbody>
</table>

- 22 bug reports (2 per app)
- Replicated bug and ran Vejovis on the application

**Recall and Precision**

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

**PRECISION**: Measure of extraneous suggestions
### Vejovis: 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: (20/22) = 91%**

**NOTE:** We consider a fix to be correct if and only if it matches the programmers’ fix for the bug.
# Vejovis: Precision and 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 (#2 in 3)

Conservative ranking

Many suggestions provided for some cases (average: 40)
Vejovis: Summary

• Vejovis: replacement suggestor for DOM-related faults

• Suggests fixes based both on DOM structure and common fix patterns used by programmers

• Evaluated on 22 real-world bugs
  – Good recall > 90%
  – High ranking of correct suggestions
  – Average 44 s to complete
Talk Outline

• Our Prior Work: DOM-Related Faults [ESEM’13]

• AutoFlox: Automatically localizing JavaScript Faults [ICST’12]

• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]

• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award

• Dompletion: Code completion for JavaScript [ASE’14]

• Conclusions & Ongoing work

• Open Challenges
Clematis: Motivation

• **Goal:** Understand and visualize dependencies between JavaScript events and the DOM

• **Challenge:** Difficult to understand the dynamic behavior and the control flow of events
  – Event propagation due to the DOM
  – Asynchronous events (e.g., timeouts)
  – DOM state changes due to events

• **Approach:** Dynamically capture execution of JavaScript applications and convert it to a model
Clematis: Approach

JavaScript Transformation

Trace Collection

Model Visualization

Behavioral Model
Clematis: Model Episodes

Copyright: Karthik Pattabiraman, 2014
Clematis: Model Links

Temporal

Causal
Clematis: Visualization
CLEMATIS: VISUALIZATION

Zoom Level 1
```javascript
function ss_update() {
    ss_cur = Math.max(ss_cur, 0);
    if (ss_cur >= ss_date.length) {
        hideElem('ss_link2');
        showElem('ss-theend');
        ss_cur = ss_date.length;
        var a = dg('ss_n');
        a.innerHTML = "Final";
        if (ss_play)
            ss_playpause();
    }
}
```
Clematis: User Experiment

• Participants
  – 20 software developers from a large software company in Vancouver (they were all well versed in web development)
  – Experimental group: Clematis
  – Control group: Chrome, Firefox, Firebug (any tool of choice)

• Procedure
  – Tasks: control flow, feature location, DOM mutations, ...

• Data collection: Task completion duration & accuracy
# Clematis: Results

## Duration

<table>
<thead>
<tr>
<th>Task</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>(39%↑)</td>
</tr>
<tr>
<td>T2</td>
<td>(48%↑)</td>
</tr>
<tr>
<td>T3</td>
<td>(68%↑)</td>
</tr>
<tr>
<td>T4</td>
<td>(32%↑)</td>
</tr>
</tbody>
</table>

## Accuracy

<table>
<thead>
<tr>
<th>Task</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>(67%↑)</td>
</tr>
<tr>
<td>T2</td>
<td>(41%↑)</td>
</tr>
<tr>
<td>T3</td>
<td>(20%↑)</td>
</tr>
<tr>
<td>T4</td>
<td>(68%↑)</td>
</tr>
</tbody>
</table>

## Task Description

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Following control flow in presence of asynchronous events</td>
</tr>
<tr>
<td>T2</td>
<td>Finding DOM mutations caused by a DOM event</td>
</tr>
<tr>
<td>T3</td>
<td>Locating the implementation of a malfunctioning feature</td>
</tr>
<tr>
<td>T4</td>
<td>Detecting control flow in presence of event propagation</td>
</tr>
</tbody>
</table>
Clematis: Summary

• Freely available:
  – https://github.com/saltlab/clematis

• Ability to visualize JavaScript events and DOM states
  – No changes to server side or client side code
  – Causal dependencies between events incl. AJAX requests
  – DOM state changes and event propagation in the DOM

• Significantly improved task duration and accuracy compared to other state-of-the-art tools
Talk Outline

• Our Prior Work: DOM-related Faults [ESEM’13]

• AutoFlox: Automatically localizing JavaScript Faults [ICST’13]

• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]

• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award

• Dompletion: Code completion for JavaScript [ASE’14]

• Conclusions & Ongoing work

• Open Challenges
Dompletion: Motivation

- Provide code-completion for DOM-JavaScript interactions

```
var x = document.getElementById("elem");
```

Copyright: Karthik Pattabiraman, 2014
Dompletion: Challenge

Potentially infinite number of DOM states!

User Input / User Action / Server Side
Dompletion: Intuition

DOM states exhibit patterns

**S0**
- HTML
- BODY
- DIV id="container"
  - INPUT class="val1" value="10"
  - INPUT class="val2" value="20"
  - BUTTON id="add" value="Add"
  - SPAN class="result" html="Result:"
Dompletion: Approach

- Webapp URL
- JavaScript Code

1. DOM Analysis (Phase 1)
2. Code Analysis (Phase 2)

Suggestion Generation (Phase 3)

- Executed only once at beginning
- Executed every time
Dompletion: Suggestion Generation

Suggestions

Copyright: Karthik Pattabiraman, 2014
Dompletion: Screenshot

```
1 $a = document.getElementById('maincol').innerHTML;
2 if(a == 'header') {
3     elem = document.getElementById('headerBar');
4 } else {
5     elem = document.getElementById('photoBoxes');
6 }
7 elem.getElementsByClassName('');
```

Implemented in the Brackets IDE

Copyright: Karthik Pattabiraman, 2014
Dompletion: Evaluation

• RQ1: Do DOM element locators for web applications converge, and if so, what is the convergence rate?

• RQ2: How accurate are the code-completion suggestions provided by Dompletion?

• RQ3: How effective is Dompletion in helping the web developers with code completion tasks?
Dompletion: Convergence (RQ1)
Dompletion: Convergence (RQ1)
Dompletion: Accuracy (RQ2)

Recall = \frac{Valid \ Output}{Valid \ Output \cup Invalid \ Output \cup Unsupported \ Selectors}

<table>
<thead>
<tr>
<th>Website</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phormer</td>
<td>83.67</td>
</tr>
<tr>
<td>Gallery3</td>
<td>63.74</td>
</tr>
<tr>
<td>WordPress</td>
<td>80.13</td>
</tr>
<tr>
<td>Average</td>
<td>75.84</td>
</tr>
</tbody>
</table>

Copyright: Karthik Pattabiraman, 2014
Dompletion: Accuracy (RQ2)

\[ \text{Precision} = \max(0, 100 - \text{Rank}_{\text{Actual Suggestion}}) \]
Dompletion: User Study (RQ3)

- 9 Participants
- 4 Tasks

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
<th>No. of Participants</th>
<th>Average Time</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group A</td>
<td>Using Dompleiton</td>
<td>5</td>
<td>1m 43s</td>
<td>90.83%</td>
<td>97.5%</td>
</tr>
<tr>
<td>Group B</td>
<td>Without Dompletion</td>
<td>4</td>
<td>4m 28s</td>
<td>76.25%</td>
<td>47.5%</td>
</tr>
</tbody>
</table>

Copyright: Karthik Pattabiraman, 2014
Dompletion: Summary

- Fully automated DOM-aware code completion technique
- Implemented on Brackets IDE
- Download
  https://github.com/saltlab/dompletion

- Significant recall and high precision with just one or two keystrokes. Benefits to real users. Real-time response.
Talk Outline

• Our Prior Work: DOM-related Faults [ESEM’13]

• AutoFlox: Automatically localizing JavaScript Faults [ICST’12]

• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]

• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award

• Dompletion: Code completion for JavaScript [ASE’14]

• Conclusions & Ongoing work

• Open Challenges
Ongoing Work: Aurebesh - 1

M V C

MODEL
DEFINES
Variables:
alpha beta gamma

VIEW
USES
Variables:
alpha beta gamma
delta

CONTROLLER
USES
Variables:
alpha beta gamma

Undefined!
Ongoing Work: Aurebesh - 2

**AngularJS**

Must match types!

Model

... $scope.display = true;
...

Controller

Must match identifiers!

Controller

$scope.foo = function() {
...
}

HTML

Extract models, views, and controllers

Find model variables and controller functions

Find identifier and type inconsistencies

Find model, view, controller groupings

Copyright: Karthik Pattabiraman, 2014
Ongoing Work: Aurebesh - 3

Fault Injection Experiment

- 95% Recall and 100% Precision

Test on Real Applications

- Detected 13 real-world bugs in 9 applications

Performance

- Ran for ~0.8 s in application with 5K LOC

Copyright: Karthik Pattabiraman, 2014
Ongoing Work: ToChal - 1

DOM element
JS Function
XHR Object
Labeled and Directed Edge

Ongoing Work:

- checkPrice()
- XHR
- updateItem()
- suggestItem()
- getUpdatePrice()
- addTaxToPrice()
- calculateTax()
- displaySuggestion()
Ongoing work: ToChal - 2

**Static Analysis**
- static function dependencies

**Dynamic Analysis**
- function invocations
- DOM accesses
- XHR relations

**Hybrid Analysis**
- Static/dynamic integration
- Finding the impact set
- Ranking the impact set

Copyright: Karthik Pattabiraman, 2014
Ongoing Work: Tochal - 3

- Empirical study: validation of impact through DOM
- User experiment: 10 professional web developers
  - Experimental group: Tochal / control group: Chrome
  - Results: improved developers’ speed and accuracy
    - 44% faster, 70% more accurate answers
Ongoing Work: Camellia - 1

• Lots of work on generating test cases
  – But what happens after a test failure?

• Fault localization crucial during debugging

• No useful stack trace when test case fails
  – Failure is on the DOM, and not on the JS code

• Need to provide developers with a starting point
Ongoing Work: Camellia - 2
Conclusions

• Modern web applications growing in importance
  – Building robust web applications is a challenge

• First part: Characterized the errors in web apps [ESEM’13]
  – Majority of errors are DOM-related (66%)
  – Majority of highest impact errors are DOM-related (80%)

• This part: Techniques to address DOM-related faults
  – AutoFlox: To automatically localize DOM-related faults [ICST’12]
  – Vejovis: To automatically fix DOM-related faults [ICSE’14]
  – Clematis: To understand JS events & DOM-JS interactions [ICSE’14]
  – Dompletion: Code completion for DOM-JS interactions [ASE’14]

Download at: http://blogs.ubc.ca/karthik/software

Copyright: Karthik Pattabiraman, 2014
Talk Outline

• Our Prior Work: DOM-related Faults [ESEM’13]

• AutoFlox: Automatically localizing JavaScript Faults [ICST’12]

• Vejovis: Automatically fixing JavaScript Faults [ICSE’14]

• Clematis: Understanding JavaScript Events [ICSE’14] – Distinguished paper award

• Dompletion: Code completion for JavaScript [ASE’14]

• Conclusions & Ongoing work

• Open Challenges
Open Challenges

• What are techniques to mitigate other kinds of (non-DOM-related) JavaScript faults?

• How can we help programmers write error-free JavaScript code through IDE support?

• What kinds of frameworks/variants of JavaScript are out there to mitigate faults?
Non-DOM-related Faults

• What about faults that do not involve DOM-JS interactions?
  – Currently account for about 35% of faults, but may increase as we mitigate DOM-related faults
  – Three related efforts
    • Non-DOM related API faults [FSE’14]
    • Type-related faults (Typedevil) [Berkeley’14]
    • Race conditions: Webracer [PLDI’14]
IDE Support for JavaScript

• Writing JavaScript is challenging
  – Very poor IDE support for JavaScript
  – Few tools to understand web applications

• Code completion and debugging tools
  – Approximate call graph construction [ICSE’13]
  – Static enforcement of policies [Livshits’09]
  – Record and Replay: Mugshot [NSDI’10]
TypeScript and JavaScript Frameworks

• Type/formalism analysis for JavaScript
  – Verified JavaScript semantics [Guha-ECOOP’10]
  – Gradual Typing [Swamy-POPL’14]
  – Static analysis [Moller-FSE’11]

• Frameworks for construction JavaScript Apps
  – Flapajax [Guha-OOPSLA’09]
  – Arrows [Hicks-DLS’09]