How I Learned to Stop Worrying and Love the "DOM" :

Characterizing and Improving the Reliability of JavaScript-based Web Applications

Karthik Pattabiraman

Frolin S. Ocariza, Jr., Kartik Bajaj, and Ali Mesbah

University of British Columbia (UBC)

My Research

- **Building fault-tolerant software applications**

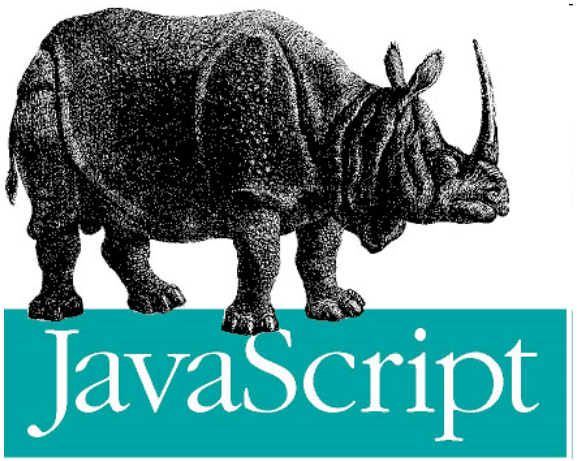
- **Compiler & runtime techniques for resilience**
  - Partitioning data for differential resilience [ASPLOS'11]
  - Error detection in parallel programs [DSN'12]
  - Error detection in soft-computing applications [DSN'13]

- **This talk**
  - Reliability of modern web applications
  - [ISSRE'10] [ISSRE'11] [ICST'12] [ESEM'13] [ICST'13] [ASE'13]

Modern Web Applications: Examples

Modern Web Applications: JavaScript

- JavaScript: Implementation of ECMAScript standard
  - Client-Side JavaScript: used to develop web applications
- Executes in client's browser – send AJAX messages
- Responsible for web application's core functionality
- Not easy to write code in – has many "evil" features



Studies of JavaScript Web Applications

Performance and parallelism:
JSMeter [Ratanaworabhan-2010], [Richards-2009], [Fortuna-2011]

Reliability

?

Security and Privacy:
[Yue-2009], Gatekeeper[Guarnieri-2009], [Jang-2010]

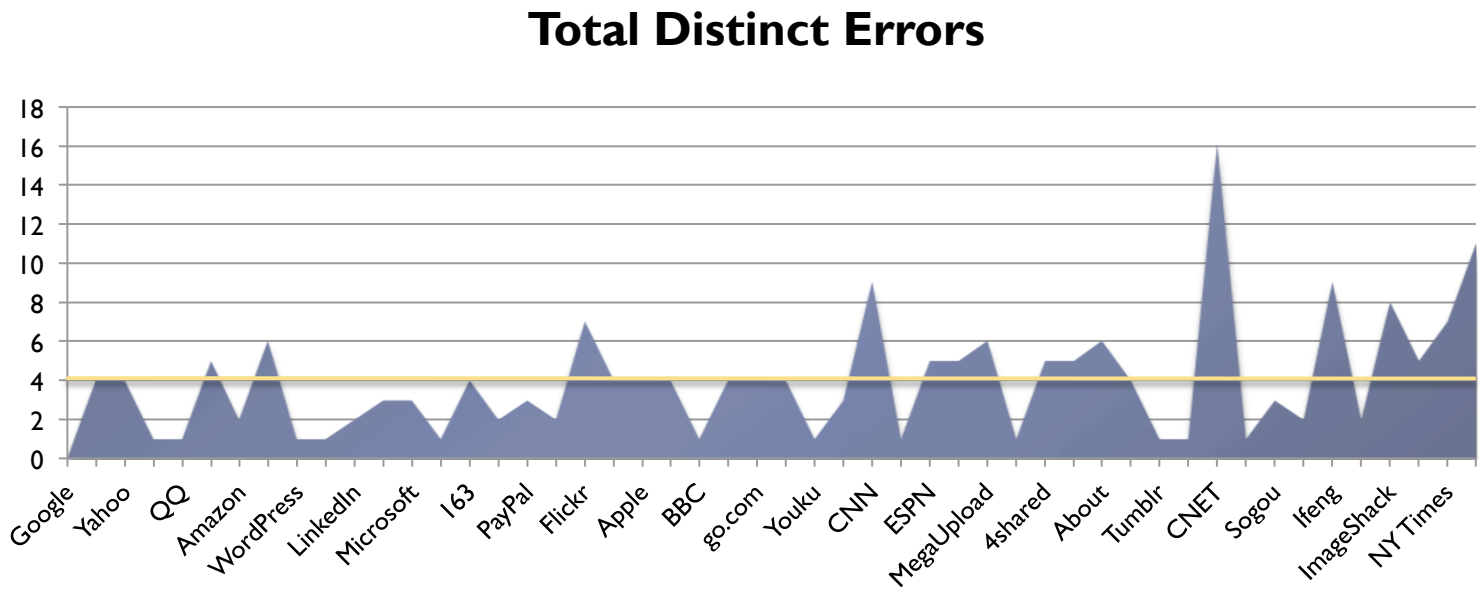
Does Reliability Matter in Web Apps ?

- Snapshot of iFeng.com: Leading media website in China

an error occurred when processing this directive

JavaScript Reliability: Our Prior Work

- Earlier study based on Console Messages: Alexa top 100
- **Popular web applications experience four distinct JavaScript error messages on average [ISSRE'11]**
- Many errors were non-deterministic and it was hard to determine the root cause and impact of these errors



Talk Outline

- Bug Report Study of twelve open source JS applications
  - To understand bug characteristics [ESEM'13]
- AutoFlox: Localizing DOM-related faults in JS applications
  - Based on dynamic backward slice [ICST'12 best paper nominee]
- Vejovis: Automatically fixing JavaScript Faults [in preparation]
- Future Directions & Other work

Bug Report Study: Goals

- What errors/mistakes ***cause*** JavaScript faults?

- What ***impact*** do JavaScript faults have?

Bug Report Study of twelve popular, Open Source JavaScript Applications

Bug Report Study: Experimental Objects

Eight JavaScript Web Applications

Four JavaScript Libraries

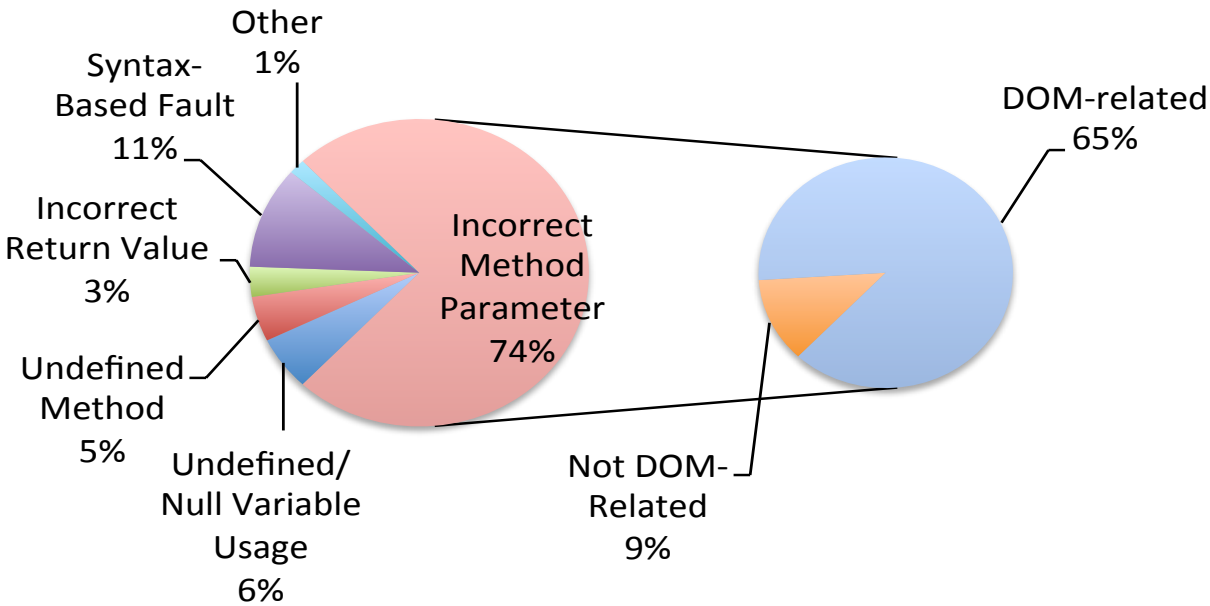
Bug Report Study: Methodology

- Collect bug reports from bug repositories
  - Focus on bugs that are marked fixed to avoid spurious bugs
  - Organized into a uniform format (XML file)

Bug Report Study: Research Questions

- **RQ1**: What types of JavaScript *faults* occur in web apps?
- **RQ2**: What is the nature of *failures* from JS faults?
- **RQ3:** What is the impact of JS faults ?
- **RQ4**: What is the root cause of JS faults?
- **RQ5**: How long does it take to fix a JS fault?

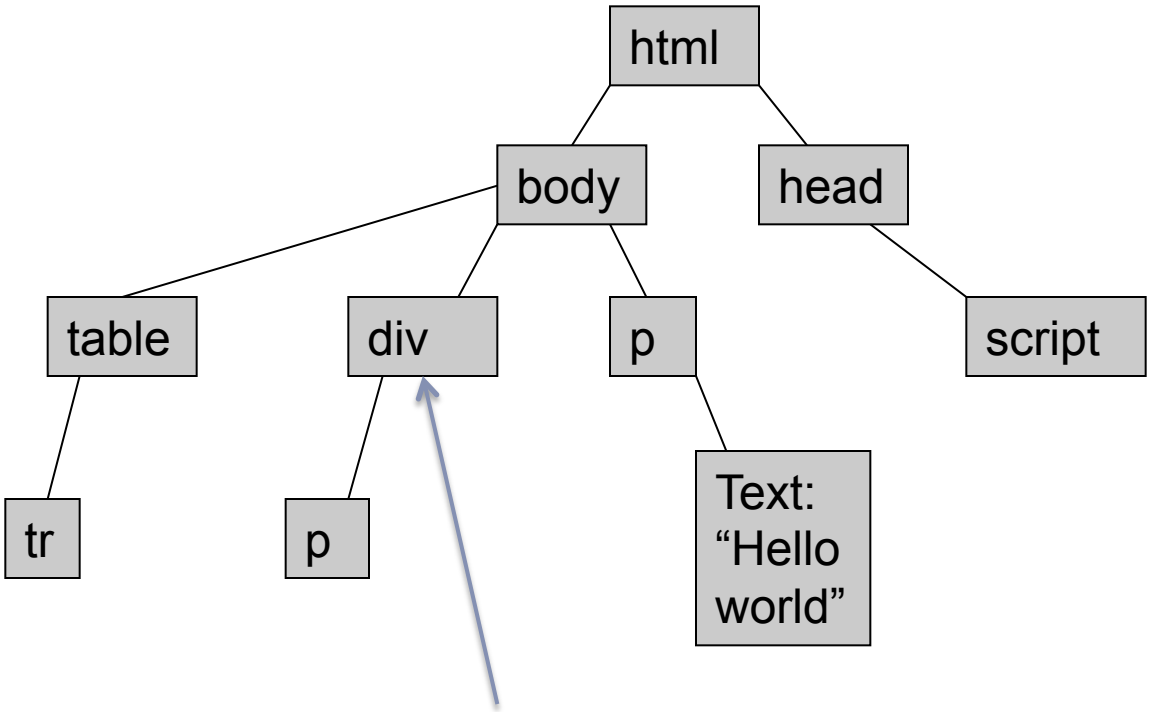
Bug Report Study: Fault Categories



Incorrect Method Parameter Fault: Unexpected or invalid value passed to JS method or assigned to JS property

DOM-Related Fault: The method is a DOM API method
- Account for around two-thirds of JavaScript Faults

Bug Report Study: DOM-Related Faults



Want to retrieve this element

Bug Report Study: DOM-Related Faults

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

Inexistent ID

DOM-related JavaScript fault

DOM:

id: elem

Bug Report Study: DOM-Related Fault

ID of element to retrieve: hello_world

```
var toggle = 1;
var x = “hlelo_”;
var y = “world”;
var elem = document.getElementById(x + y);
var dis = “”;
if (toggle == 1) {
    dis = “block”;
}
else {
    dis = “inline”;
}
elem.style.display = dis;
```

Error: “hello_” is misspelled

Fault: Code would attempt to retrieve the DOM element using wrong ID.
Variable elem becomes **NULL**

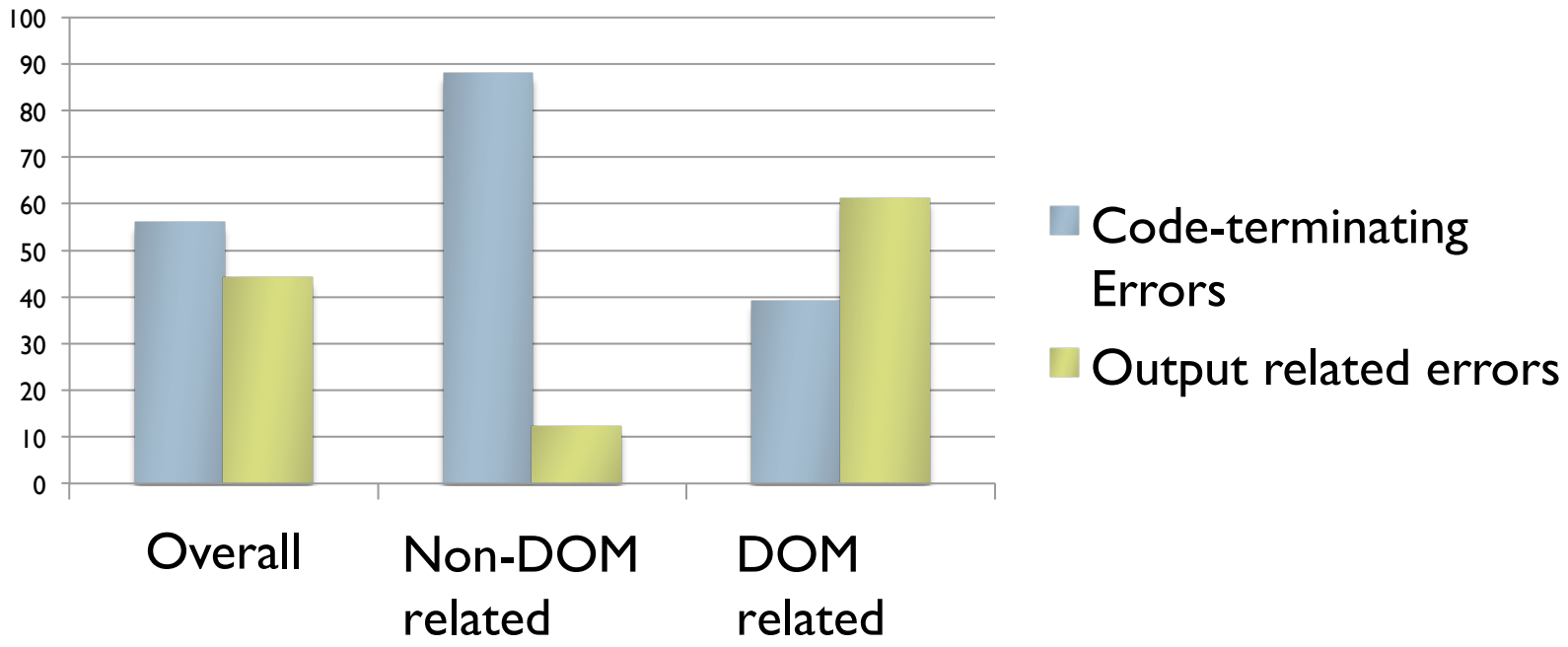
Failure: NULL EXCEPTION!

Bug Report Study: Research Questions

- **RQ1**: What types of JavaScript *faults* occur in web apps?
- **RQ2**: What is the nature of *failures* stemming from JS faults?
- **RQ3:** What is the impact of JS faults ?
- **RQ4**: What is the root cause of JS faults?
- **RQ5**: How long does it take to fix a JS fault?

Bug Report Study: Nature of Failures

- DOM related errors are less likely to be code-terminating
  - 54% of JavaScript faults lead to exceptions
  - 88% of *non-DOM-related faults* lead to exceptions
  - Only 39% of *DOM-related faults* lead to exceptions

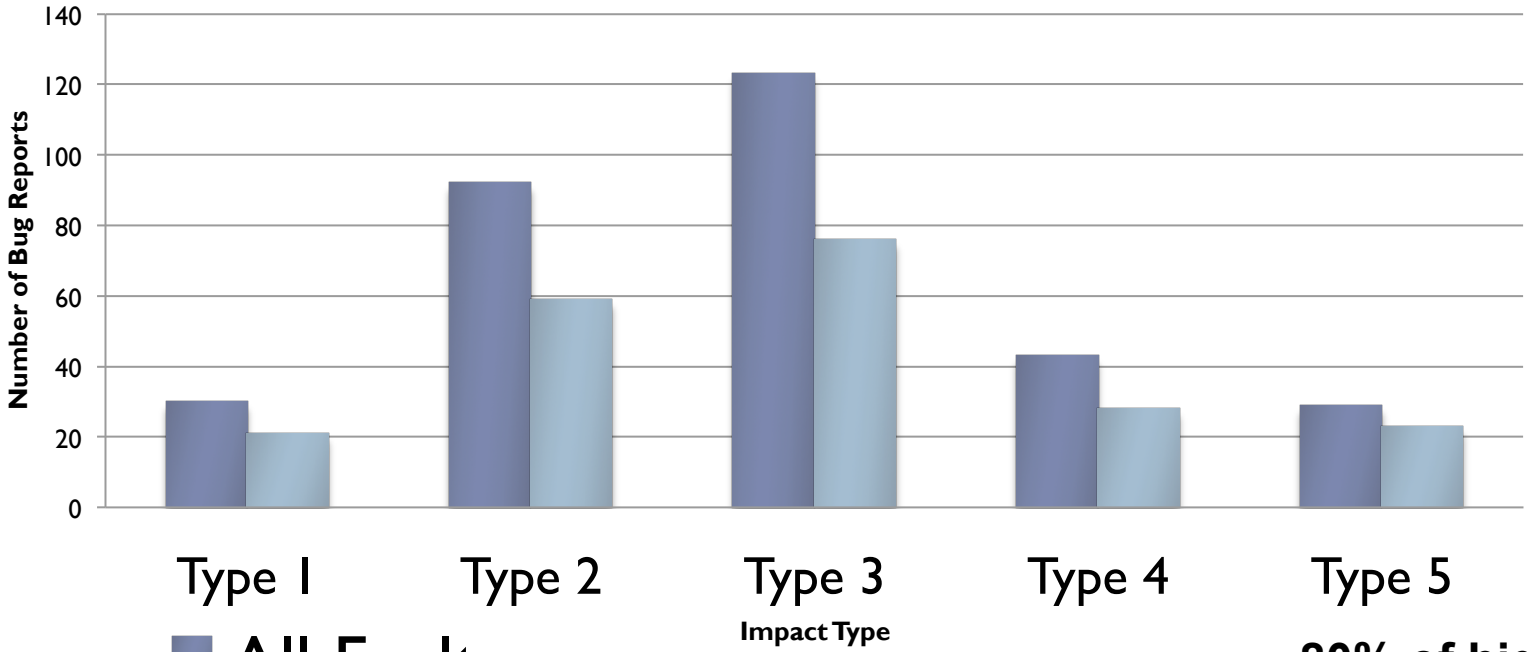


Bug Report Study: Research Questions

- **RQ1**: What types of JavaScript *faults* occur in web apps?
- **RQ2**: What is the nature of *failures* stemming from JS faults?
- **RQ3:** What is the impact of JS faults ?
- **RQ4**: What is the root cause of JS faults?
- **RQ5**: How long does it take to fix a JS fault?

Bug Report Study: Impact of JS Faults

- Impact Types – Based on Bugzilla's classification [ICSE'11]
  - Type 1 (lowest impact), Type 5 (highest impact)



All Faults

DOM-Related Faults Only

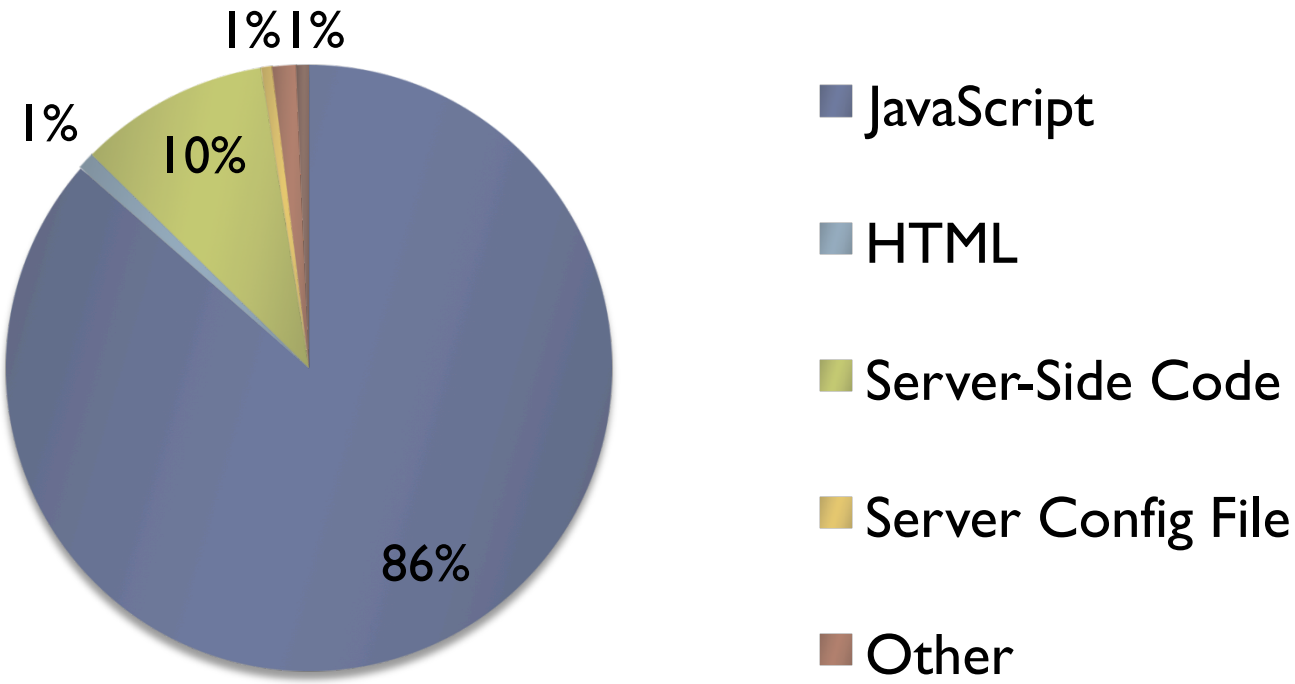
80% of highest impact faults are DOM-related

Bug Report Study: Research Questions

- **RQ1**: What types of JavaScript *faults* occur in web apps?
- **RQ2**: What is the nature of *failures* stemming from JS faults?
- **RQ3:** What is the impact of JS faults ?
- **RQ4**: What is the root cause of JS faults?
- **RQ5**: How long does it take to fix a JS fault?

Bug Report Study: Causes of JS Faults

- **Error Locations**
  - Most errors manually committed by programmer in JS code

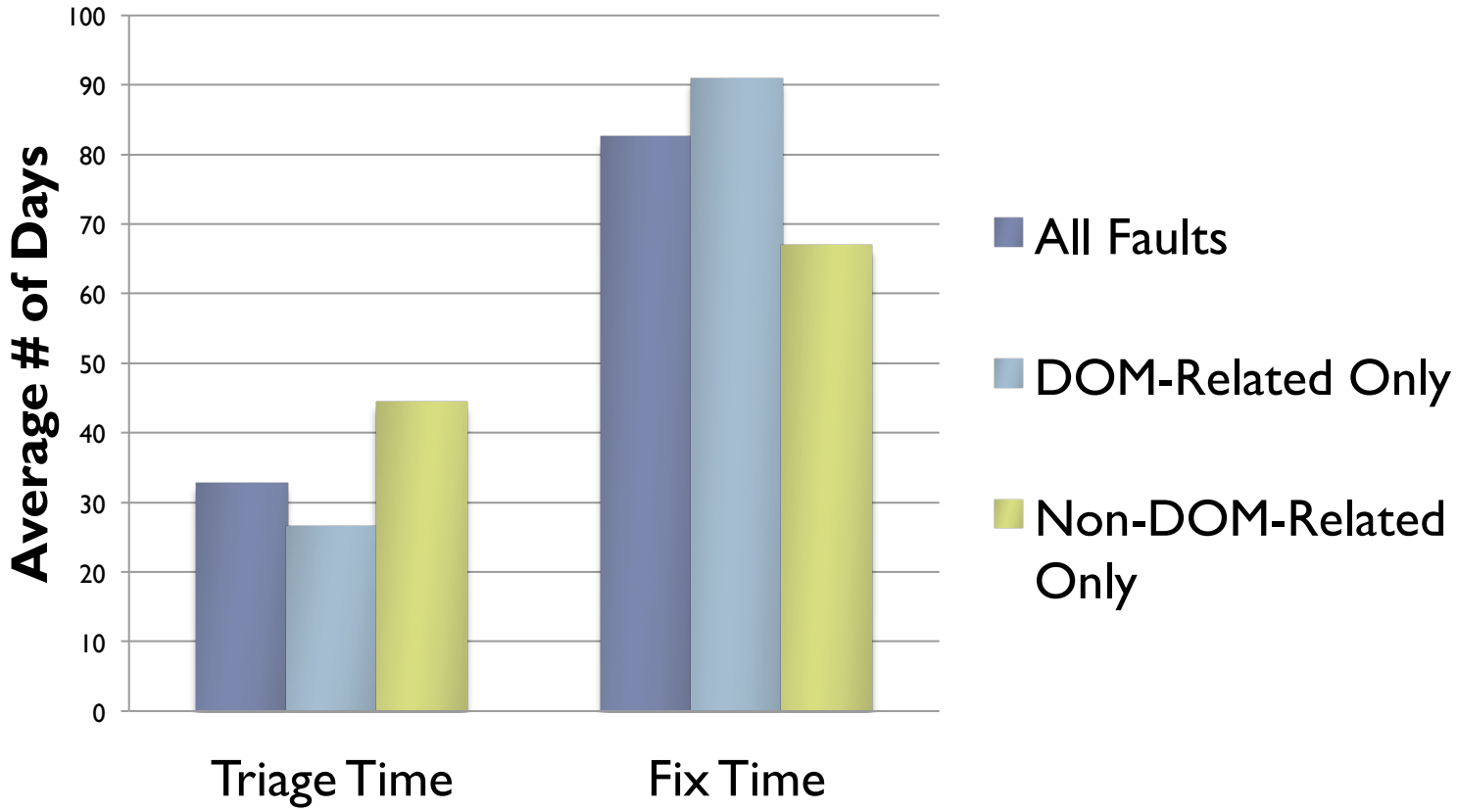


Bug Report Study: Research Questions

- **RQ1**: What types of JavaScript *faults* occur in web apps?
- **RQ2**: What is the nature of *failures* stemming from JS faults?
- **RQ3:** What is the impact of JS faults ?
- **RQ4**: What is the root cause of JS faults?
- **RQ5**: How long does it take to fix a JS fault?

Bug Report Study: Triage and Fix Times

- **Triage Time**: Time it took to assign/comment on bug
- **Fix Time**: Time it took to fix the bug since it was triaged



Bug Report Study: Summary

- **Bug report study of 12 applications: JS faults**
  - Over 300 bug reports analyzed; only fixed bugs considered

- **DOM-related faults dominate JavaScript faults**
  - Responsible for nearly two-thirds of all faults
  - Mostly lead to output errors (not exceptions)
  - Responsible for 80% of highest impact faults
  - Arise in the JavaScript code (not server/HTML)
  - Take 50% longer time to fix for developers

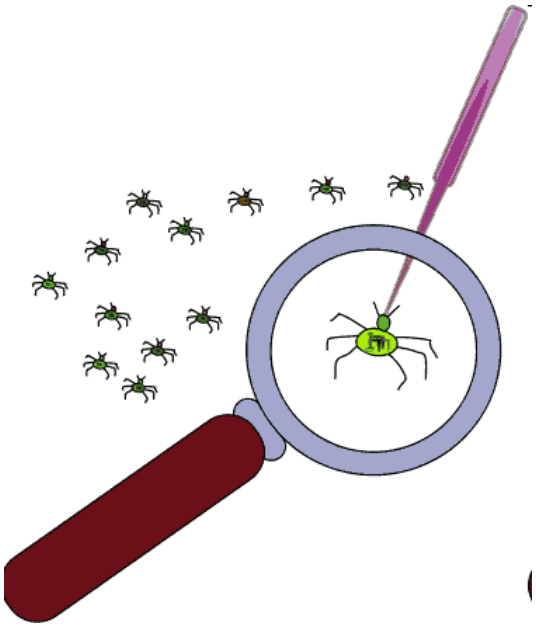
- **Need low-cost solutions for DOM-related faults**

Talk Outline

- Bug Report Study of twelve open source JS applications
  - To understand bug characteristics [ESEM'13]
- AutoFlox: Localizing DOM-related faults in JS applications
  - Based on dynamic backward slice [ICST'12 best paper nominee]
- Vejovis: Automatically fixing JavaScript Faults [in preparation]
- Future Directions & Other Work

AutoFlox: Fault Localization

- 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: Scope of Technique

- Types of DOM-related JS errors

  Code-terminating DOM-related JS errors

```
element = $("elem");
b = element.getAttribute("badAttr")
element.innerHTML = "text";
b.value = "newValue";
```

exception

  Output DOM-related JS errors

```
function changeToBlue(elem) {
    elem.style.color = "red";
}
```

Wrong colour change

AutoFlox: Running Example

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

```
function changeBanner(bannerID) {
  clearTimeout(changeTimer);
  changeTimer = setTimeout(changeBanner, 500
  
  prefix = "banner_";
  currBannerElem = document.getElementById(prefix+currentBannerID);
  bannerToChange = document.getElementById(prefix + bannerID);
  currBannerElem.removeClassName("active");
  bannerToChange.addClassName("active");
  currentBannerID = bannerID;
}
currentBannerID = 1;
changeTimer = setTimeout(changeBanner, 5000);
```

bannerID will be set to undefined

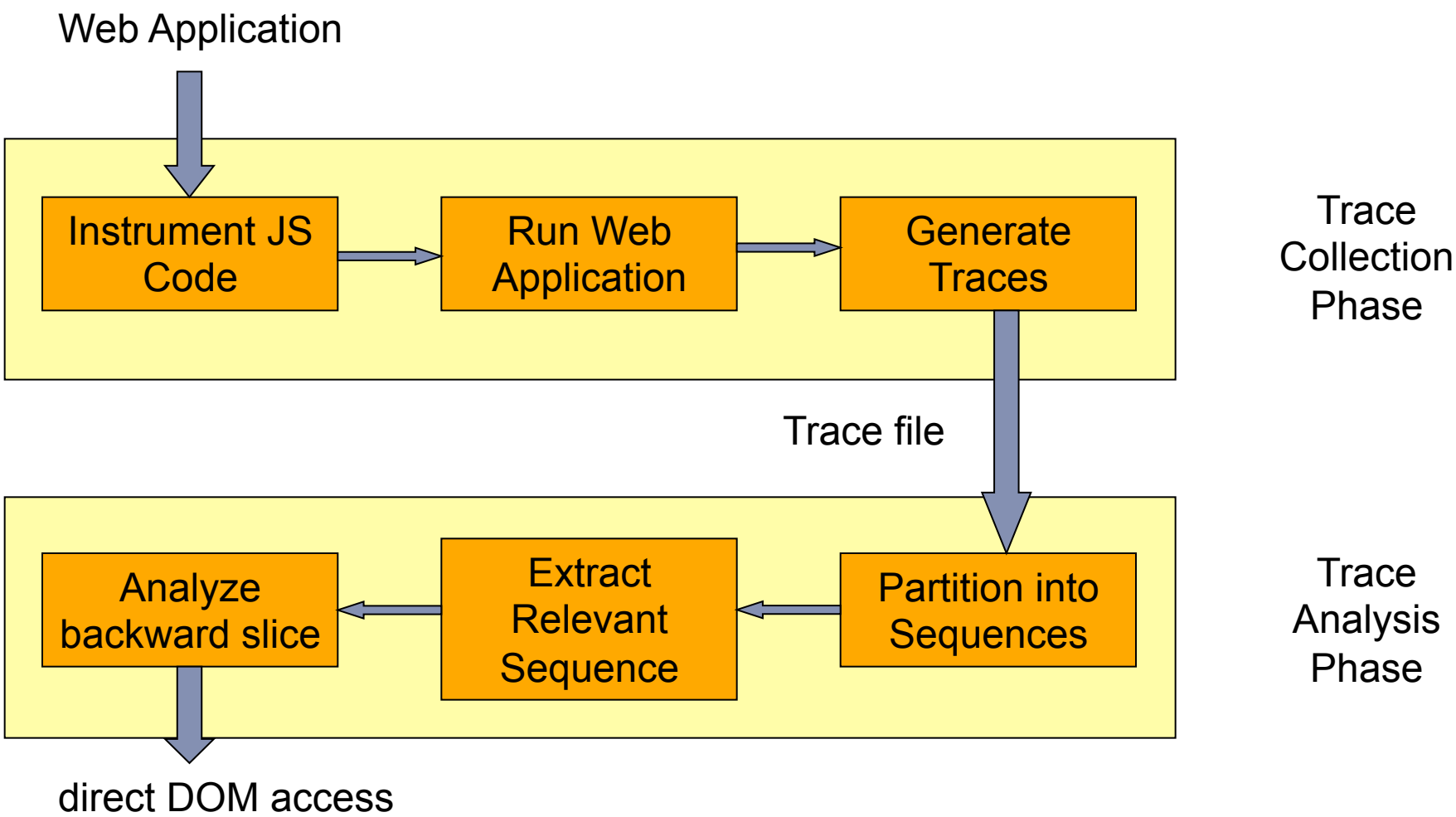
Would return null

Passed with no argument (even though changeBanner needs one argument)

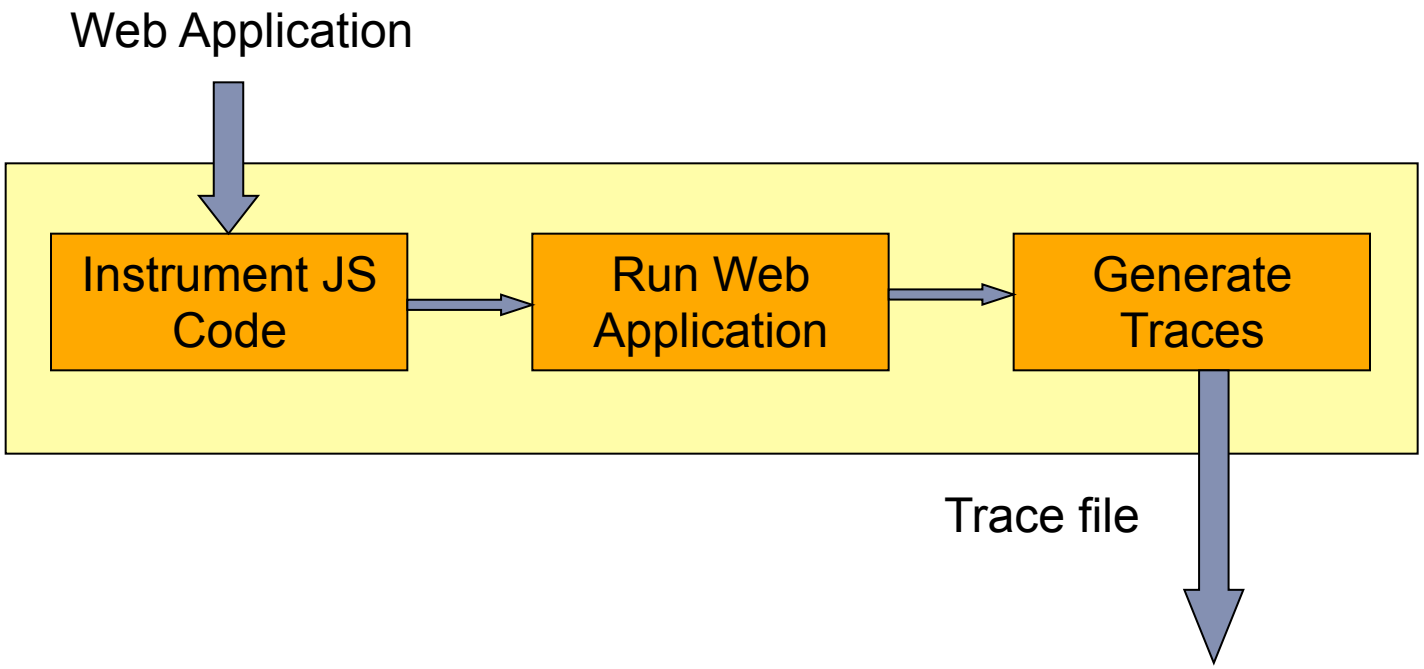
NULL EXCEPTION!

asynchronous call

AutoFlox: Block Diagram



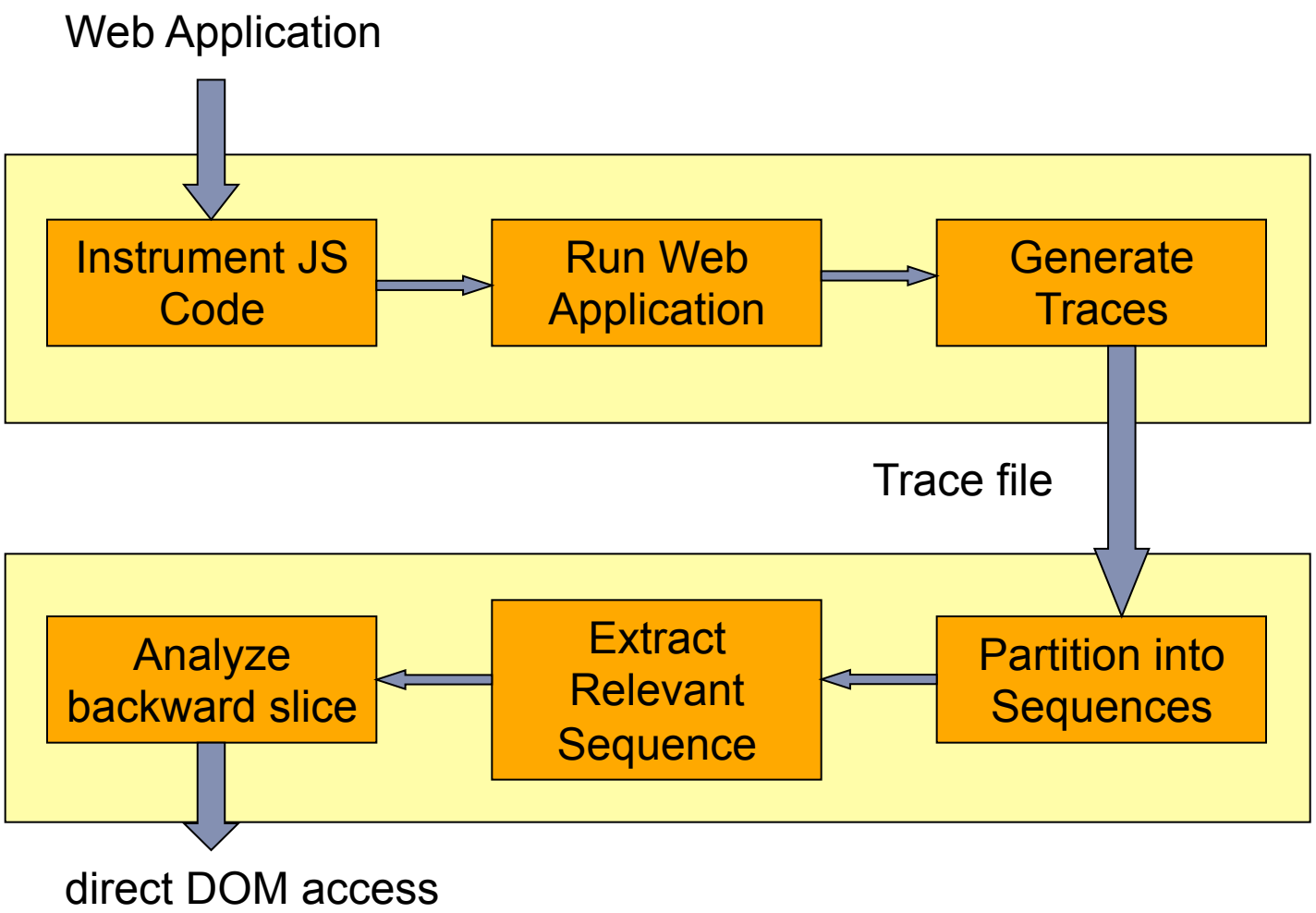
AutoFlox: Trace Collection



AutoFlox: Trace Collection

```
Trace Record Prefix:
    changeBanner:::4
Variables:
    currentBannerID (global): 1
    changeTimer (global): 2
    bannerID (local): -11
    prefix (local): none
    currBannerElem (local): none
    bannerToChange (local): none
13 changeTimer = setTimeout(changeBanner, 5000); trace();
```

AutoFlox: Trace Analysis



AutoFlox: Trace Analysis

```
1  function changeBanner(bannerID) {
2   clearTimeout(changeTimer);
3   changeTimer = setTimeout(changeBanner, 5000);
4
5   prefix = "banner_";
6   currBannerElem = document.getElementById(prefix+currentBannerID);
7   bannerToChange = document.getElementById(prefix + bannerID);
8   currBannerElem.removeClassName("active");
9   bannerToChange.addClassName("active");
10  currentBannerID = bannerID;
11 }
12 currentBannerID = 1;
13 changeTimer = setTimeout(changeBanner, 5000);
```

AutoFlox outputs this line as direct DOM access

bannerToChange being assigned return value of DOM access function

Last variable to take on null value

Error Marker

Sequences.: (1) line2 -> line3 -> line5 -> line6 -> line7 -> line8 -> line9

Relevant Seq.: line12 -> line13 -> line2 -> line3 -> line5 -> line6 -> line7 -> line8 -> line9

(2) line12 -> line13

AutoFLox: Implementation

- Trace Collection: Modified versions of existing tools
  - InvarScope [Groeneveld et al.]
  - Crawljax [Mesbah et al.]

- Trace Analysis: Written from scratch

- Evaluated on three applications
  - Tudu
  - TaskFreak
  - Wordpress

AutoFlox: Accuracy

- **Approach**: Fault injection into TUDU, TaskFreak, and WordPress to emulate DOM-related JS faults

Web App	Total Number of Mutations	Number of direct DOM accesses identified	Percentage identified
TaskFreak	29	29	100%
TUDU	24	24	100%
WordPress	13	7	53.8%
Overall	66	60	90.9%

AutoFlox: Performance

- **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 DOM-related JS errors**
  - Errors due to interaction of DOM and JS
  - Assumes code-terminating faults that result in exceptions

- **AutoFlox uses dynamic backward slicing to successfully isolate > 90% of injected faults**

Talk Outline

- Bug Report Study of twelve open source JS applications
  - To understand bug characteristics [ESEM'13]
- AutoFlox: Localizing DOM-related faults in JS applications
  - Based on dynamic backward slice [ICST'12 best paper nominee]
- Vejovis: Automatically fixing JavaScript Faults [in preparation]
- Future Directions & Other work

Vejovis: Motivation

- Automatically "fix" DOM-related faults
- Starts from DOM interaction point (i.e., AutoFLox's output)
- Finds symptoms to determine "possible sicknesses"
  - Suggests workarounds to get rid of these symptoms and, hopefully, the actual error.
  - Workaround patterns based on "common fixes" applied to DOM-related errors

Vejovis: Example

JavaScript Code:

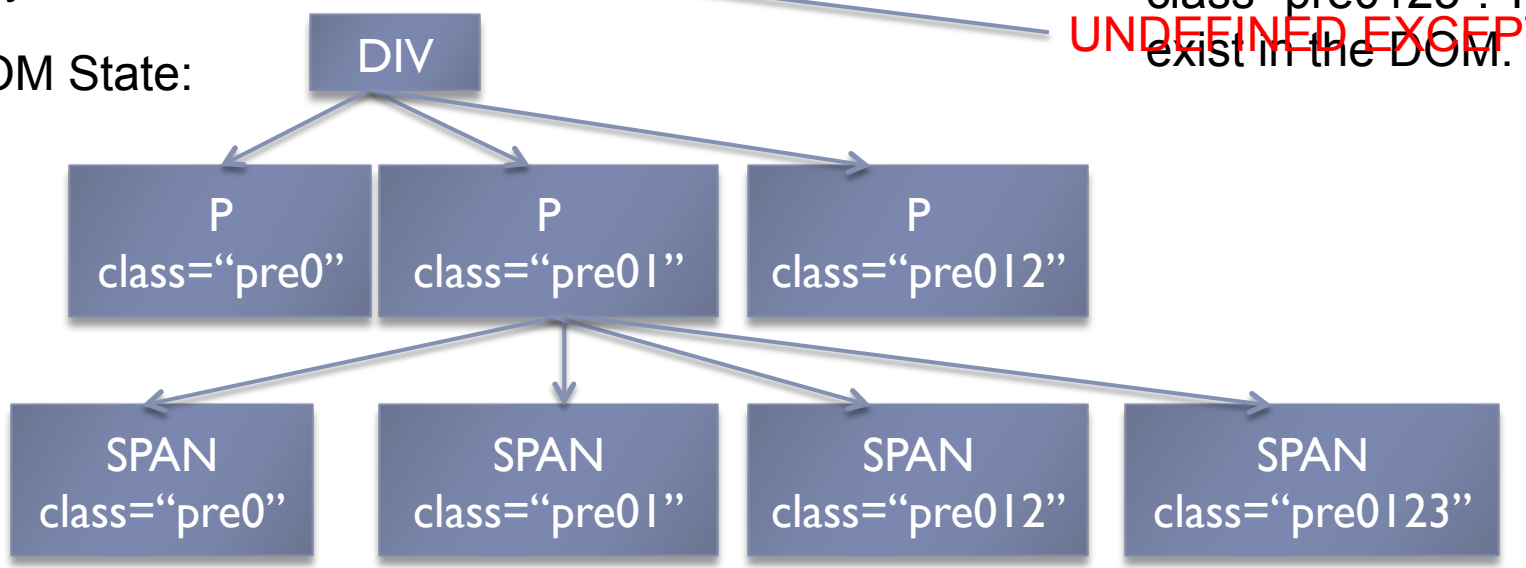
```
1 var cls = "pre";
2 for (var i = 0; i <= 3; i++) {
3   cls = cls + i;
4   var elem = $("p." + cls);
5   elem[0].style.display = "block";
6 }
```

Set up the class name of element being retrieved

Retrieve element using $(). In last iteration of loop, code tries to retrieve p element with class "pre0123". This does not exist in the DOM.

UNDEFINED EXCEPTION!

DOM State:

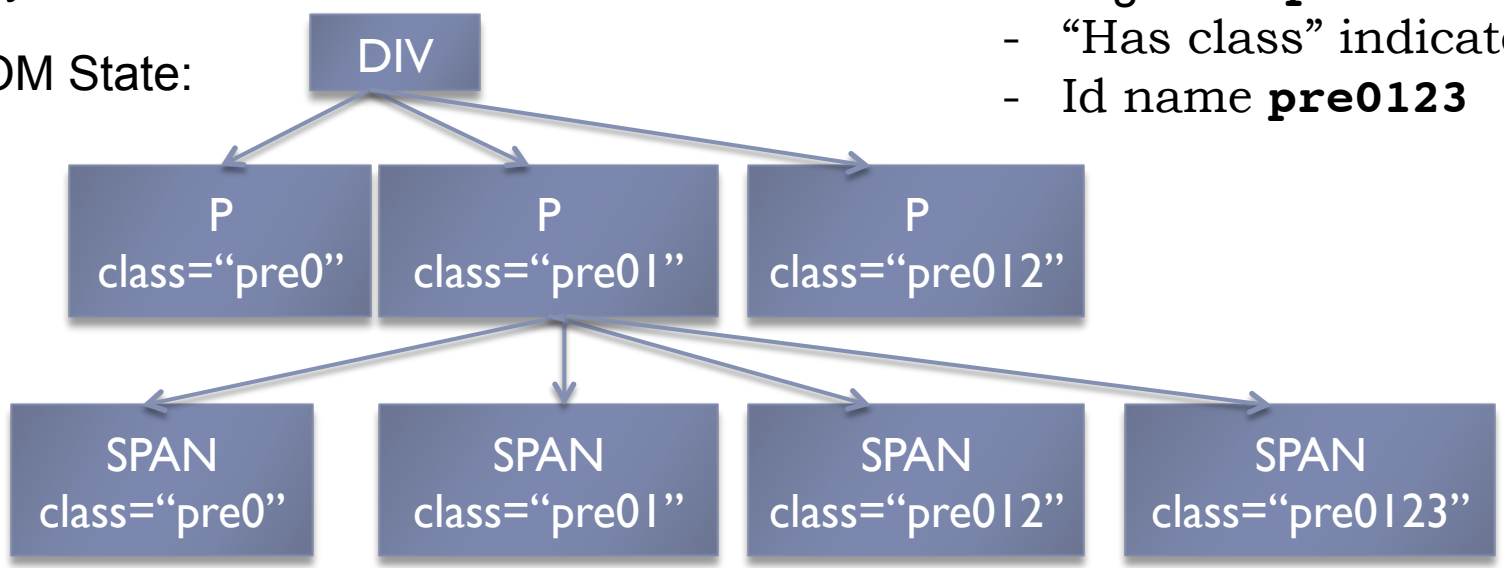


Vejovis: Example

JavaScript Code:

```
var cls = "pre";
for (var i = 0; i <= 3; i++) {
  cls = cls + i;
  var elem = $("p." + cls);
  elem[0].style.display = "block";
}
```

DOM State:



Vejovis

Step 1: Assume CSS selector is wrong. Divide the selector into components.

e.g., Selector is "`p.pre0123`".
Components:

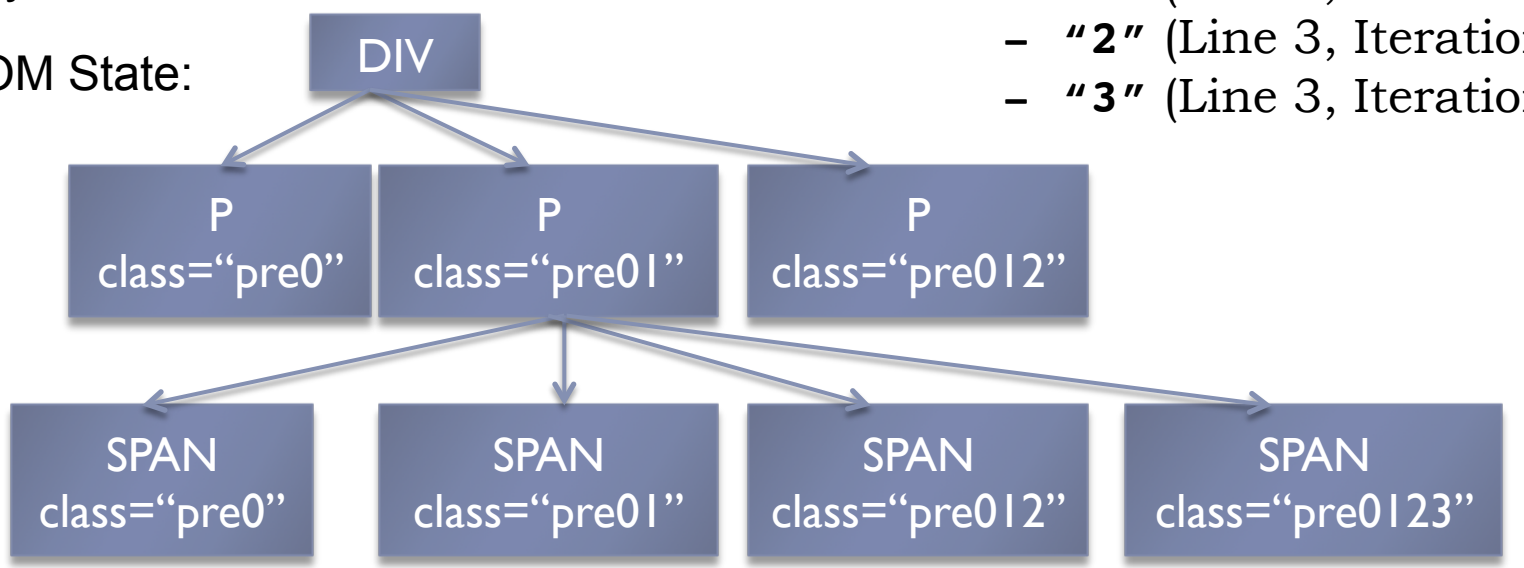
- Tag name **`p`**
- "Has class" indicator **`.`**
- Id name **`pre0123`**

Vejovis: Example

JavaScript Code:

```
1 var cls = "pre";
2 for (var i = 0; i <= 3; i++) {
3   cls = cls + i;
4   var elem = $("p." + cls);
5   elem[0].style.display = "block";
6 }
```

DOM State:



Vejovis

Step 2: Divide each component into *string set*.

e.g., String set of "**pre0123**"

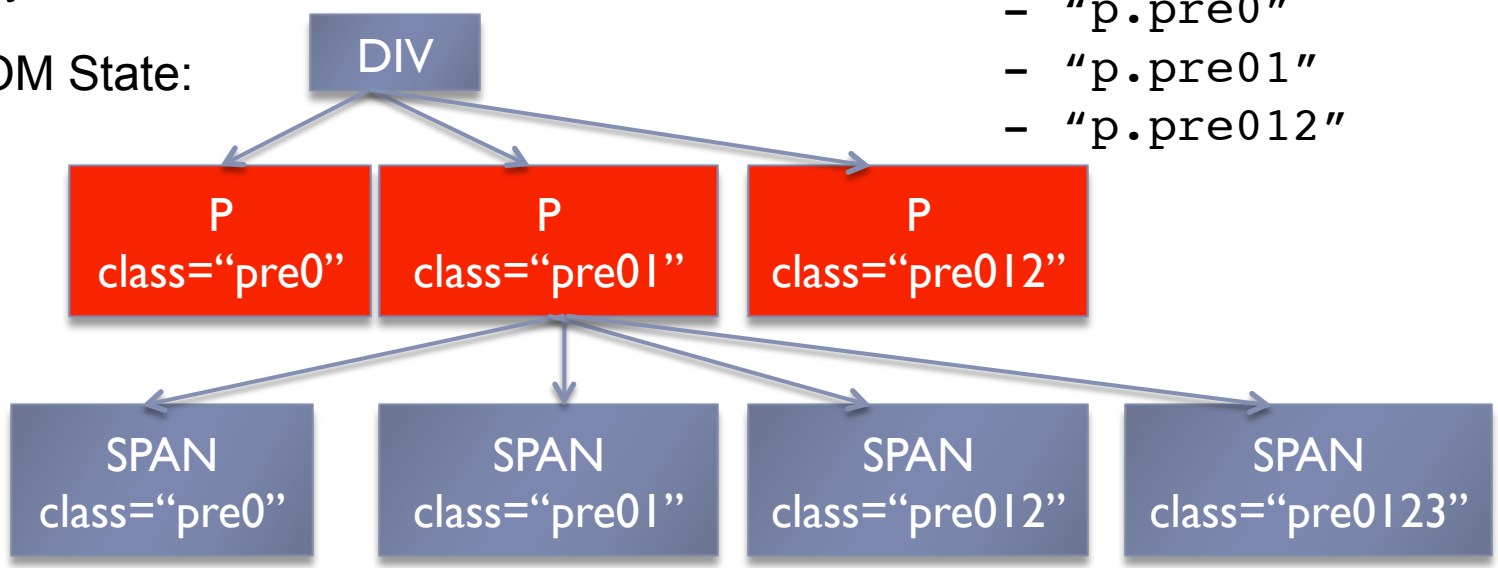
- "**pre**" (Line 1)
- "**0**" (Line 3, Iteration 1)
- "**1**" (Line 3, Iteration 2)
- "**2**" (Line 3, Iteration 3)
- "**3**" (Line 3, Iteration 4)

Vejovis: Example

JavaScript Code:

```
var cls = "pre";
for (var i = 0; i <= 3; i++) {
  cls = cls + i;
  var elem = $("p." + cls);
  elem[0].style.display = "block";
}
```

DOM State:



Vejovis

Step 3: Find *valid replacements* for each component of the selector, based on the current DOM state.

e.g., Valid replacements for the ID component **`pre0123`** in the original selector "`p.pre0123`" are

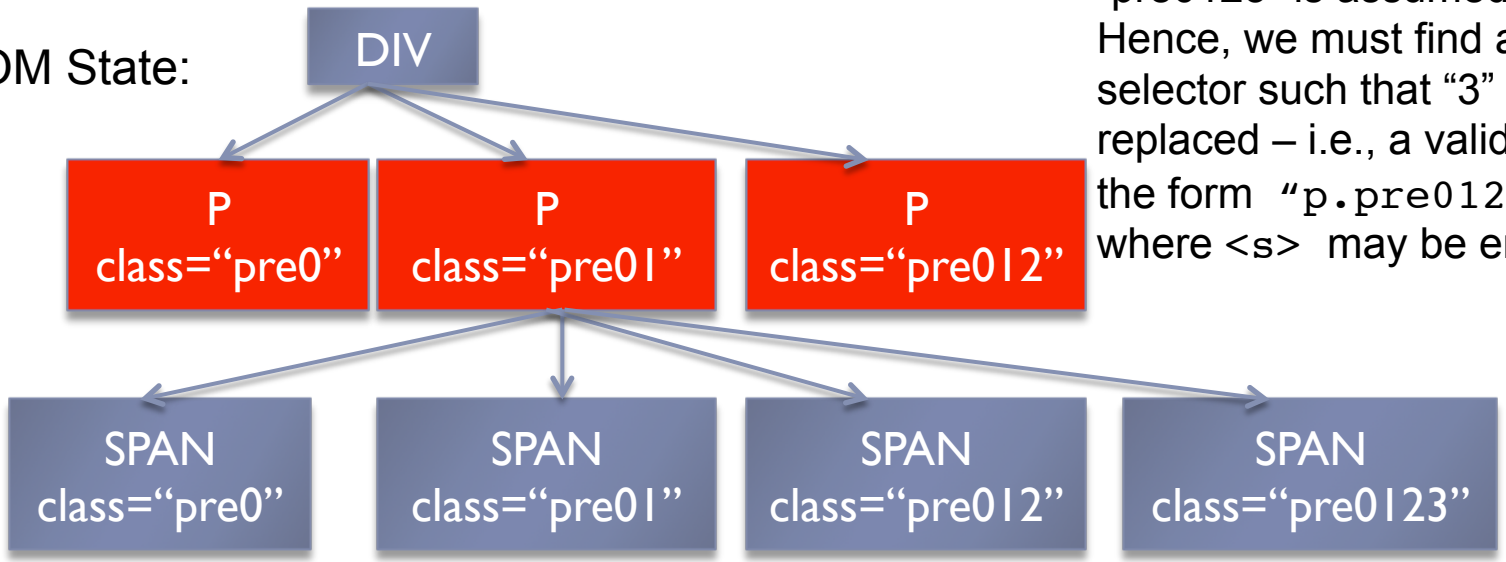
- "`p.pre0`"
- "`p.pre01`"
- "`p.pre012`"

Vejovis: Example

JavaScript Code:

```
var cls = "pre";
for (var i = 0; i <= 3; i++) {
  cls = cls + i;
  var elem = $("p." + cls);
  elem[0].style.display = "block";
}
```

DOM State:



Vejovis

Step 4: For each string set part, assume that part is wrong.
Use string constraint solver to find suitable replacements based on valid selectors found earlier.

Example: Let's say the "3" part of "pre0123" is assumed wrong. Hence, we must find a valid selector such that "3" has been replaced – i.e., a valid selector of the form `"p.pre012<s>"`, where `<s>` may be empty string.

Vejovis: Example

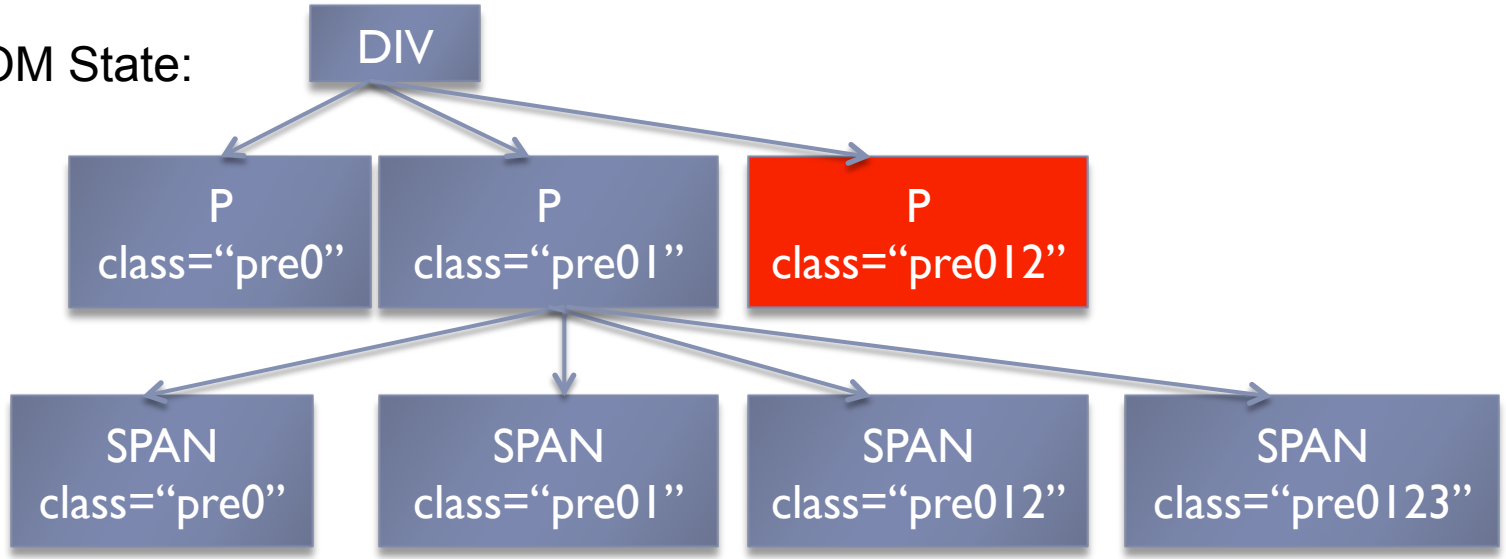
JavaScript Code:

```
1 var cls = "pre";
2 for (var i = 0; i <= 3; i++) {
3   cls = cls + i;
4   var elem = $("p." + cls);
5   elem[0].style.display = "block";
6 }
```

Example: To replace "`p.pre0123`" with "`p.pre012`", the following suggestion is displayed:

"Off by one. Modify the upper bound of the for loop that contains Line 3"

DOM State:



Vejovis: Implementation

- **Crawljax** used to crawl web application [Mesbah'09]
  - User specifies "clickables" to reproduce bug
- **Rhino** used to instrument JS code (for retrieving traces and supplementary information)
  - Instrumentation done at AST level
- **Hampi** used to perform string constraint solving when finding potential replacement selectors [Kiezun'09]
  - Valid selectors used when defining "Context Free Grammar"

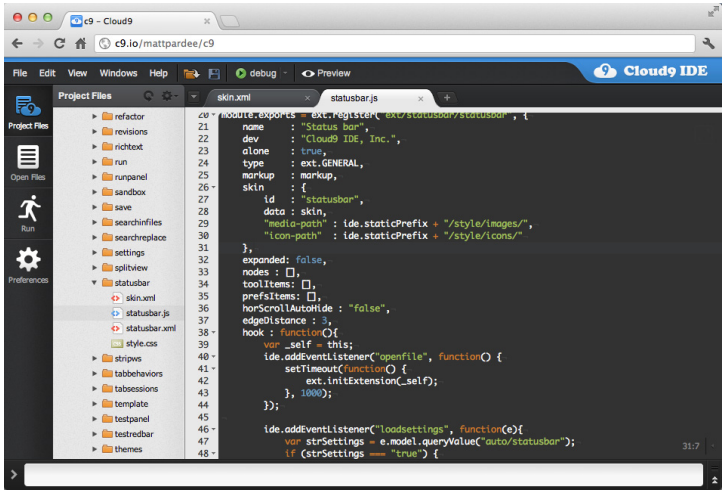
Talk Outline

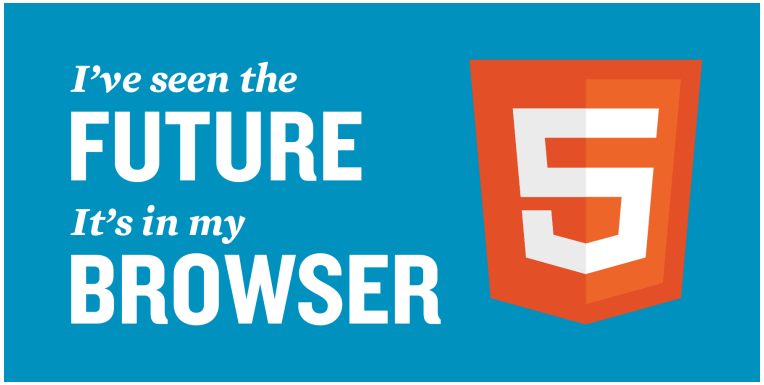
- Bug Report Study of twelve open source JS applications
  - **To understand bug characteristics [ESEM'13]**
- AutoFlox: Localizing DOM-related faults in JS applications
  - Based on dynamic backward slice [ICST'12 best paper nominee]
- Vejovis: Automatically fixing JavaScript Faults [in preparation]
- Future Directions & Other work

Future Directions

- Integrating multiple techniques into a single IDE
  - Allow programmers to reason about DOM interactions
  - Automated code synthesis for DOM-JS interactions

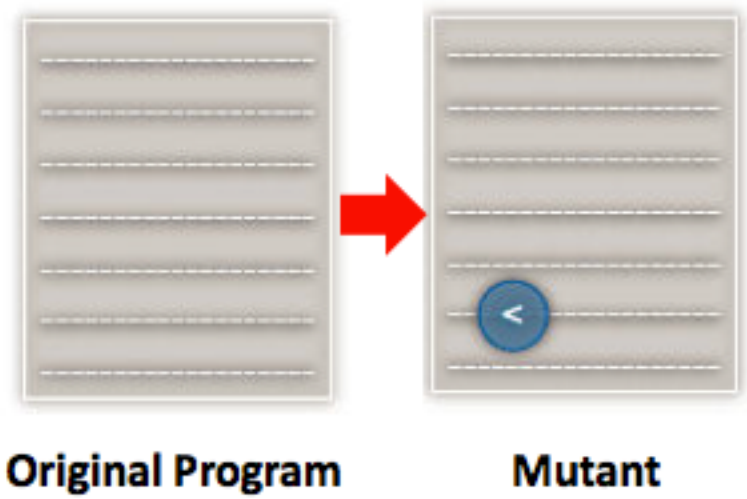
- Support for HTML5 primitives and features
  - Canvas interactions, local storage etc.





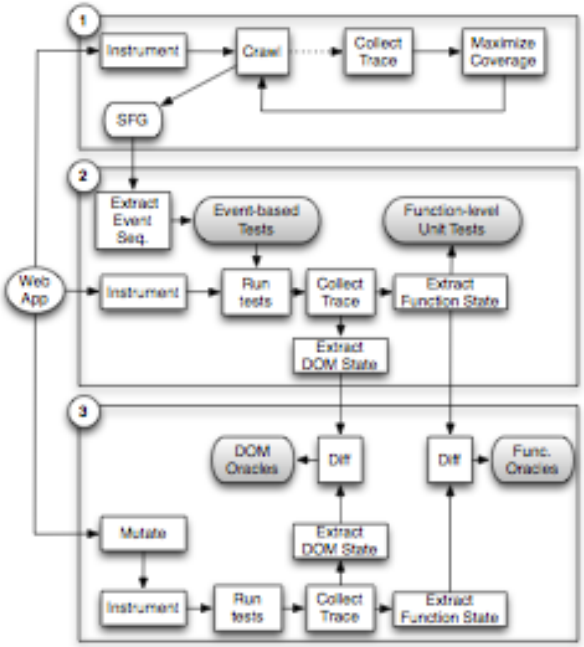
Mutandis [Mirshokraie - ICST 2013]

- Mutates original program to test quality of test suites
- Problem: Equivalent mutants obscure the value
- Generate only a few equivalent mutants – FunctionRank
- Introduced DOM-specific and JS-specific mutations



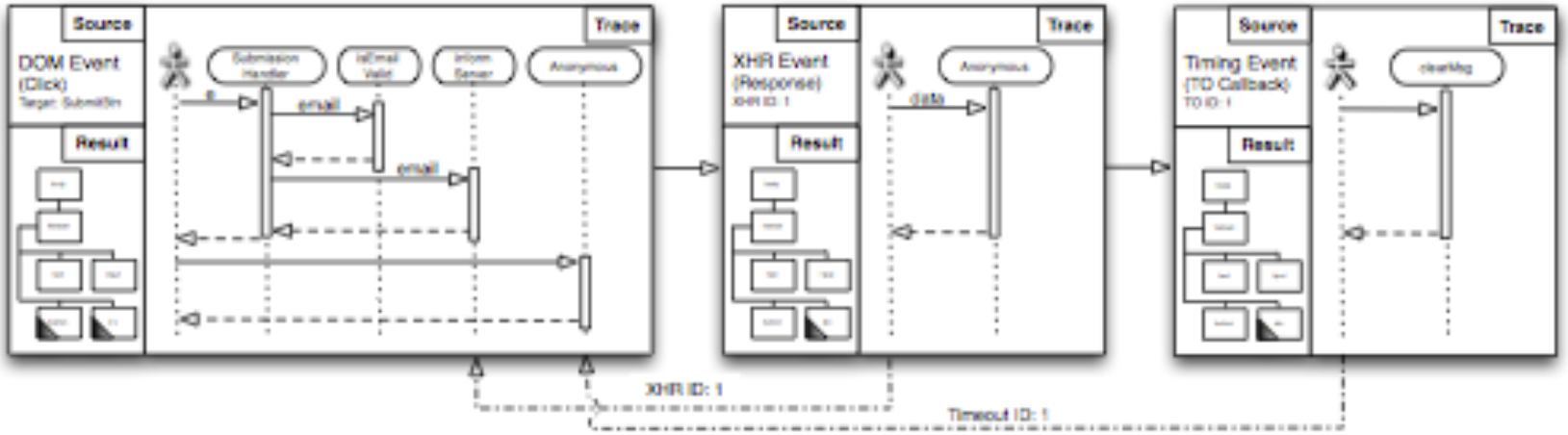
Pythia [Mirshokraie – ASE 2013]

- Automated unit test and oracle generation for web apps.
- First, crawls application to generate event sequences
- Extracts unit tests from sequences with high coverage
- Creates Oracles for unit tests using mutation testing



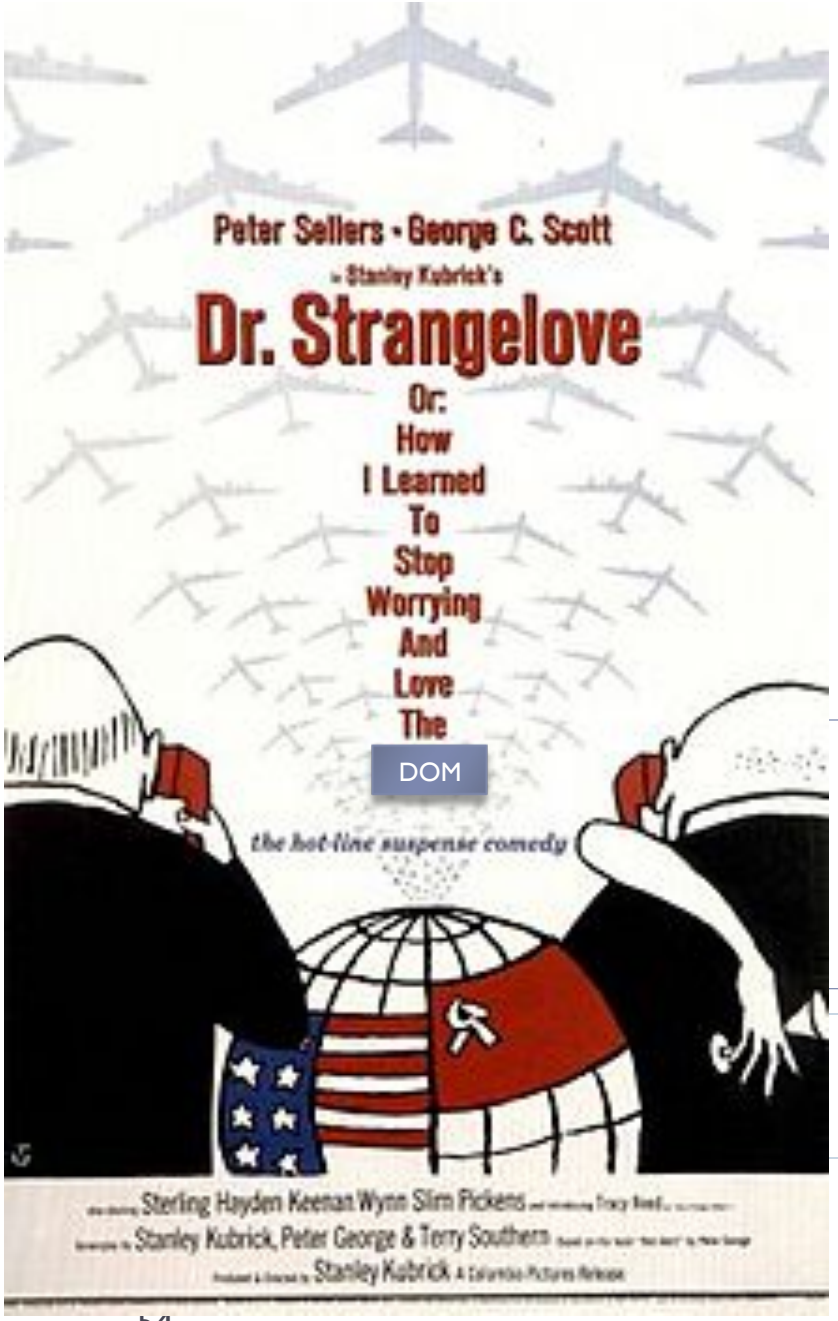
Clematis [Alimadi – under preparation]

- Challenge: Web applications are complex, and consist of DOM interactions, AJAX messages and timeouts
- Difficult to trace the links between events and JS code
- Clematis allows users to visualize causal dependencies between events and code, and between asynchrous events



Conclusions

- **Modern web applications growing in importance**
  - Reliability is a significant challenge for these applications
- **Characterized the reliability of modern web applications [ESEM'13]**
  - Majority of errors are DOM-related (66%)
  - Majority of highest impact errors are DOM-related (80%)
- **Techniques to address DOM-related faults**
  - **AutoFlox:** To localize DOM-related faults [ICST'12]
  - **Vejovis:** To automatically fix DOM-related faults [in prep.]



Questions ?

karthikp@ece.ubc.ca

http://blogs.ubc.ca/karthik/Software