

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 ?

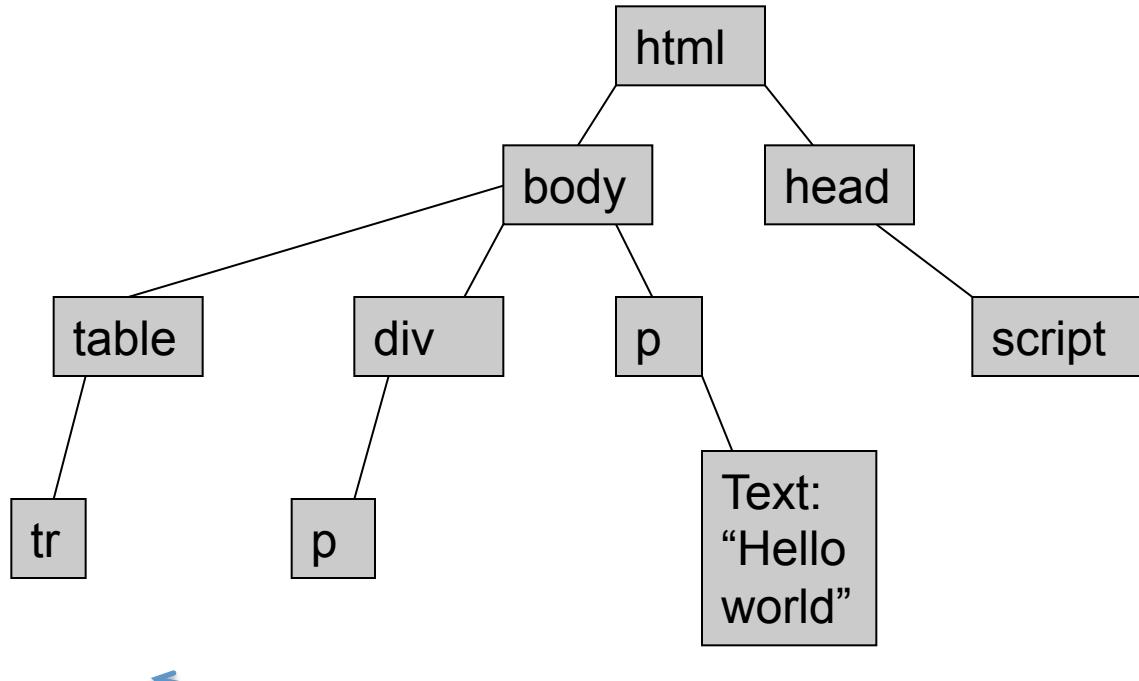
Talk Outline

- 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

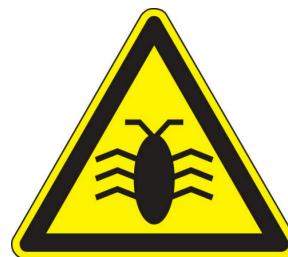
Our Prior Work [ESEM'13]

- **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)

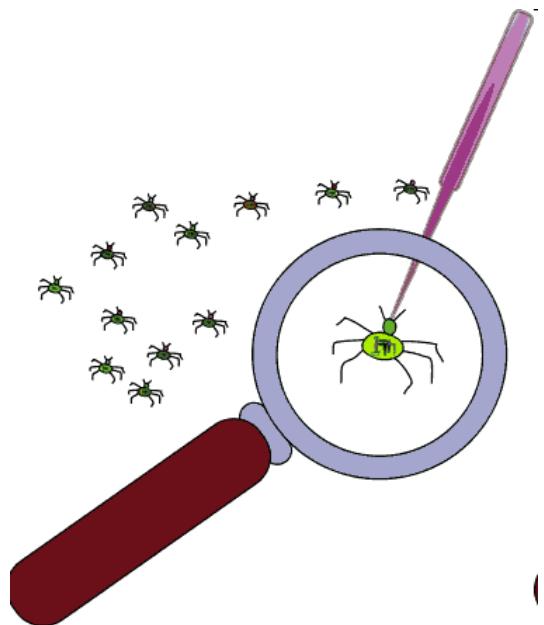


Talk Outline

- 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

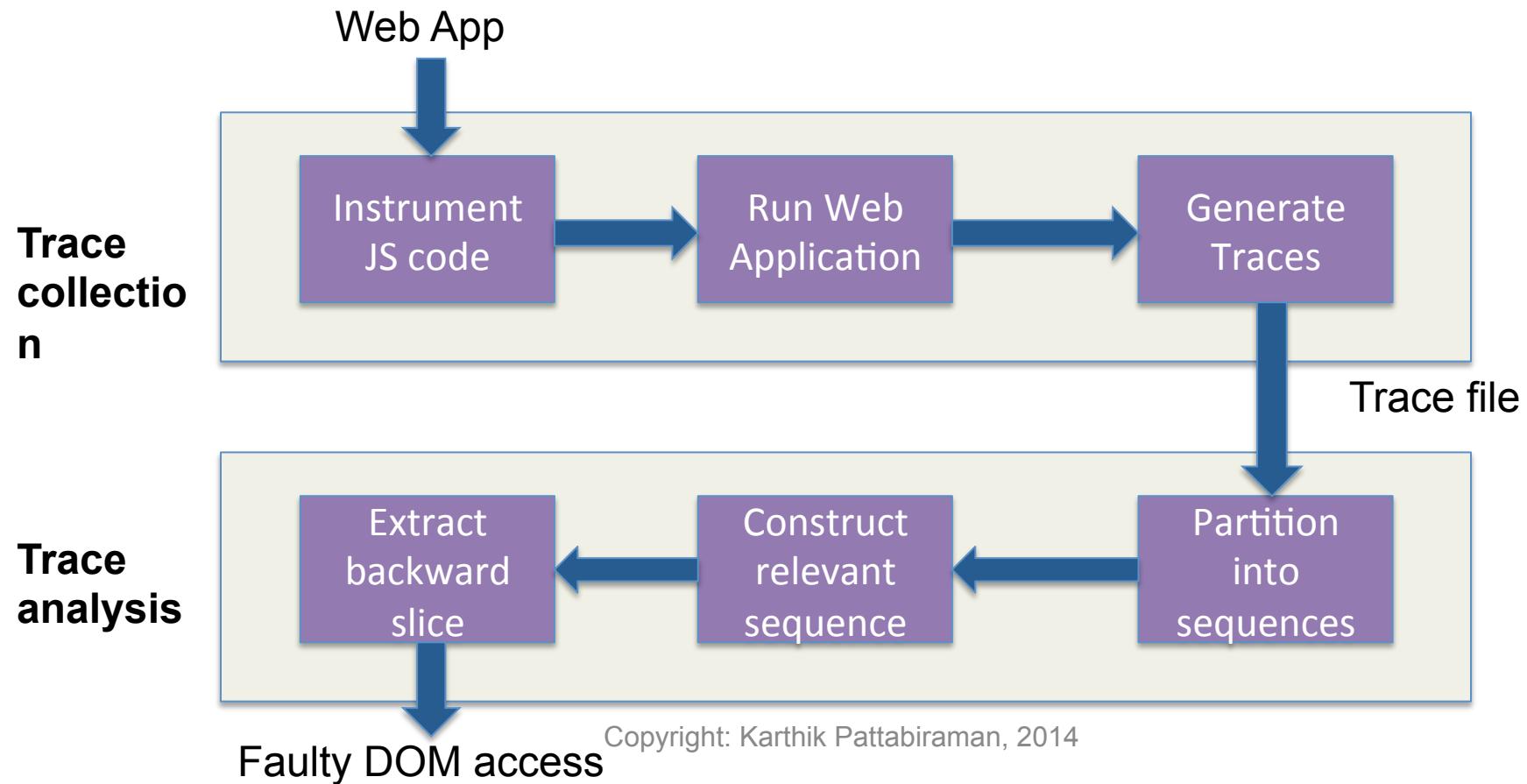
```
1 function changeBanner(bannerID) {  
2     clearTimeout(changeTimer);  
3     changeTimer = setTimeout(changeBanner, 5000);  
4  
5     prefix = "banner_";  
6     currBannerElem = document.getElementById(prefix+currentBannerID);  
7     bannerToChange = document.getElementById(prefix + bannerID);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

bannerID will be set to undefined

Would return null
Passed with no argument
(even though
NULL changeBanner needs one
asynchronous EXCEPTON)

AutoFlox: Approach

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



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

Web Application	Successful Detections	Failed Detections	Percent Successful
TaskFreak	39	0	100%
WordPress	14	3	82.4%
ChatJavaScript	10	0	100%
TUDU	9	0	100%
JSScramble	6	0	100%
JavaScript-Todo	2	0	100%
Total	80	3	96.4%

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

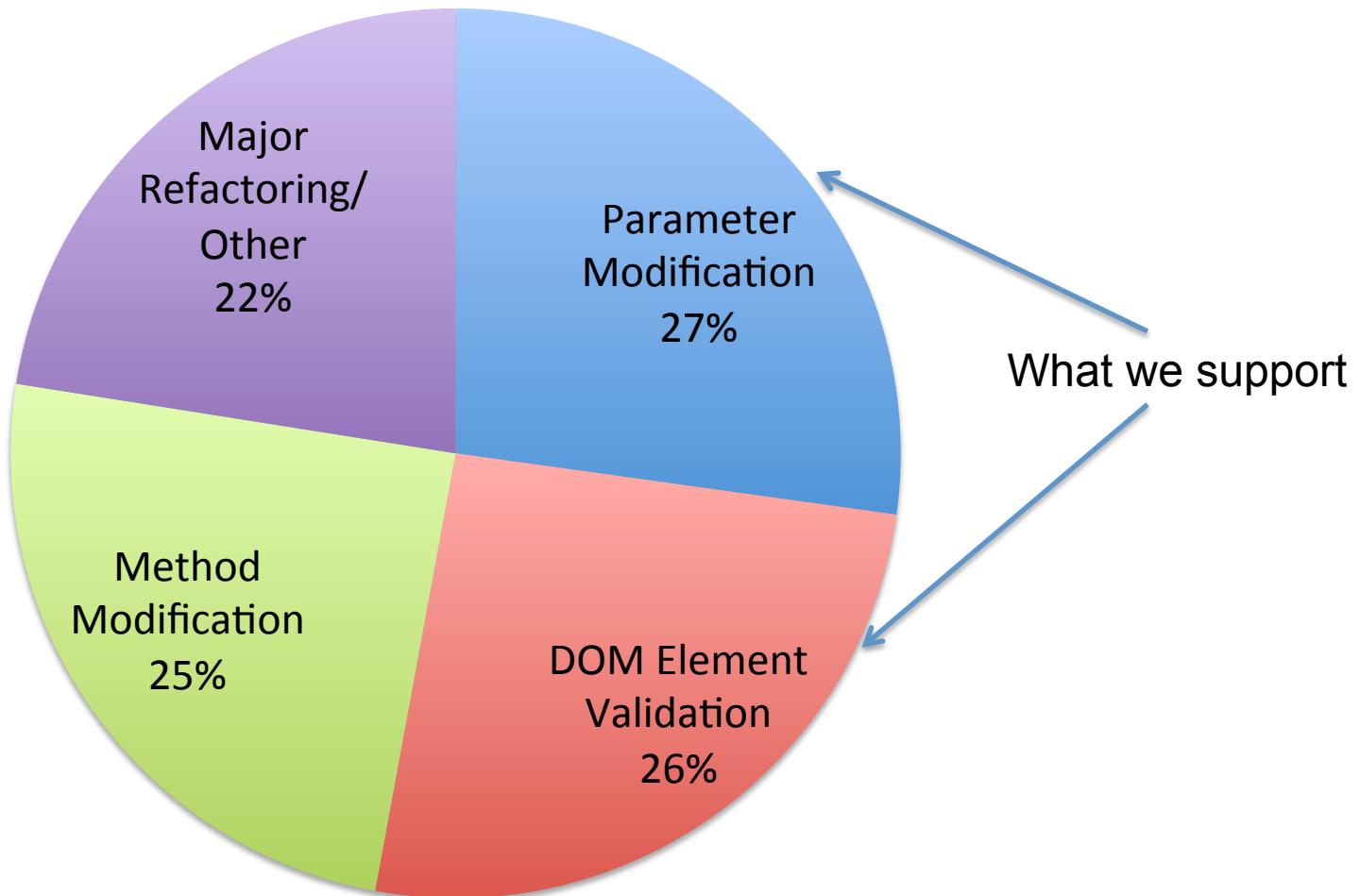
- Our Prior Work: DOM-Related Faults [ESEM'13]
- AutoFlex: Automatically localizing JavaScript Faults [ICST'12]
- Vejovis: Automatically fixing JavaScript Faults [ICSE'14]
- Clematis: Understanding JavaScript Events [ICSE'14] – Distinguished paper award
- Dompletion: Code completion for JavaScript [ASE'14]
- 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



Vejovis: Main Idea

WRONG

RIGHT

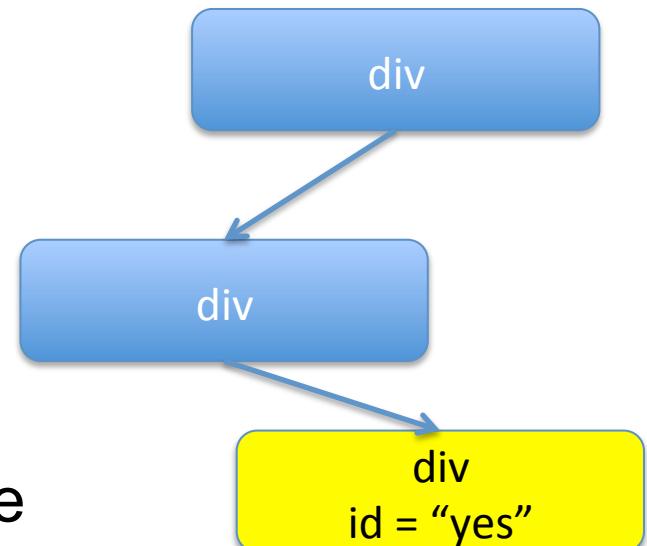
`getElementById("no")` `getElementById("yes")`

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...

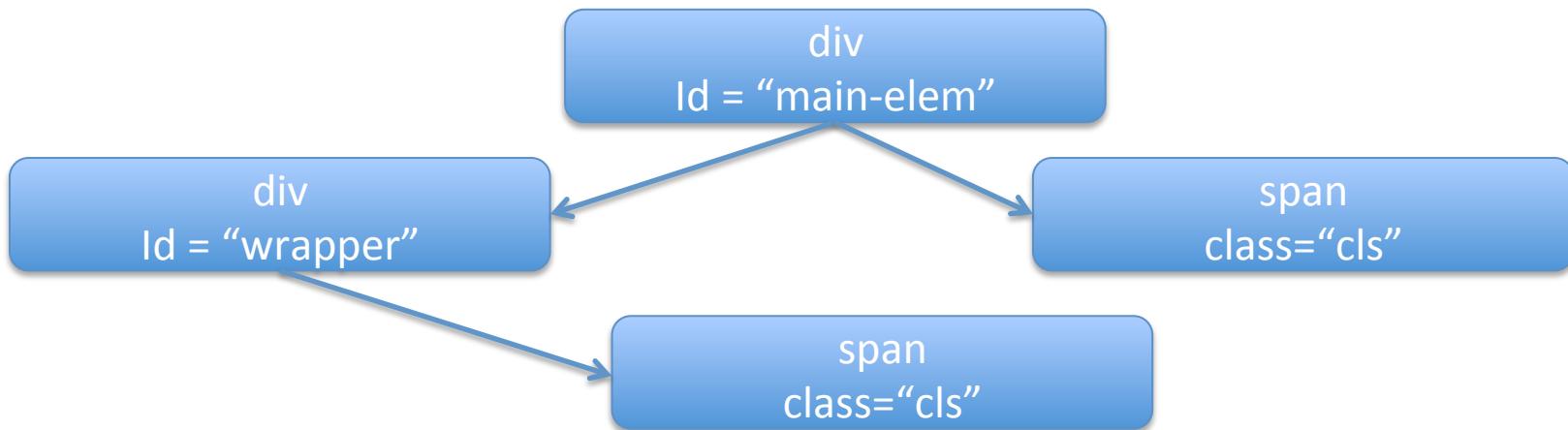
- We can use the DOM’s structure



Vejovis: Example

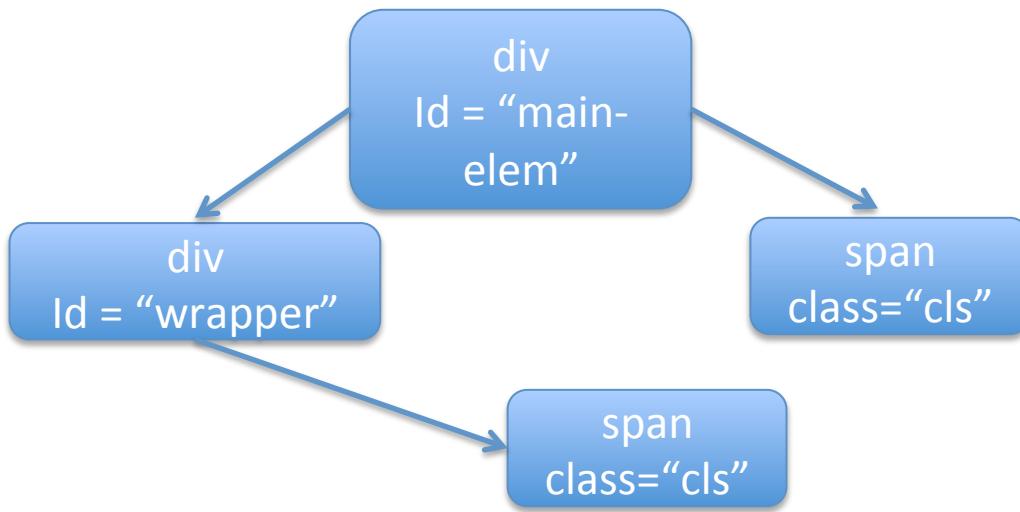
```
1 firstTag = "div";
2 prefix = "pain-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2); ← Would return
7 e[0].innerHTML = "new content"; empty set!
```

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:

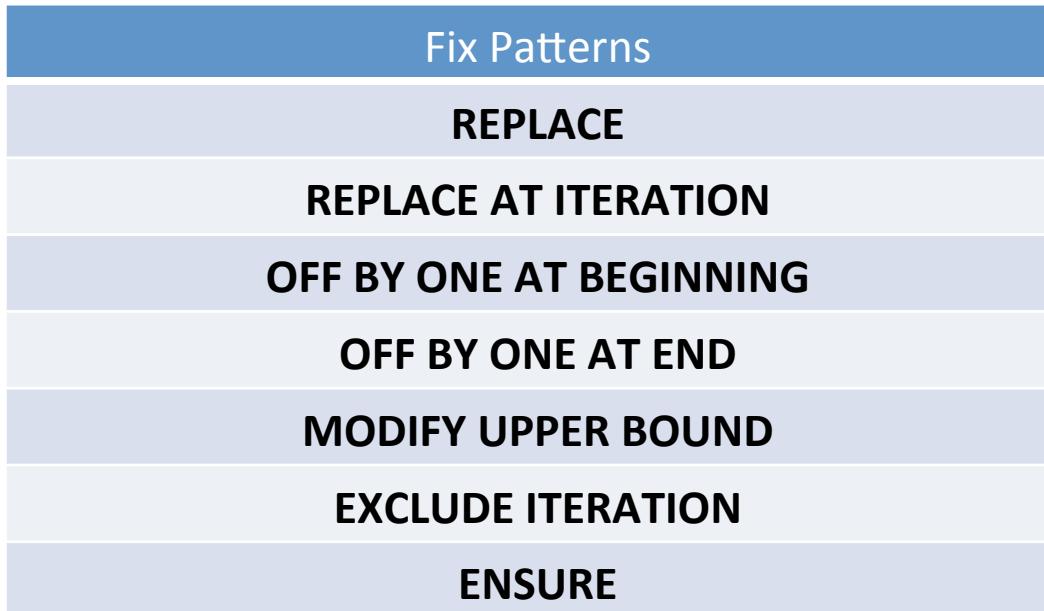
div#main-elem span.cls

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



Example: REPLACE string literal “pain-” in line 2 with string “main-”

Vejovis: Evaluation

Web Applications
Drupal
Ember.js
Joomla
jQuery
Moodle
MooTools
Prototype
Roundcube
TYPO3
WikiMedia
WordPress

- 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

Subject	Bug Report #1	Bug Report #2
Drupal	✓	✓
Ember.js	✓	✓
Joomla	✓	✓
jQuery	✓	✗
Moodle	✓	✓
MooTools	✓	✓
Prototype	✓	✓
Roundcube	✓	✗
TYPO3	✓	✓
WikiMedia	✓	✓
WordPress	✓	✓

NOTE: We consider a fix to be correct if and only if it matches the programmers' fix for the bug.

Overall Recall:
 $(20/22) = 91\%$

Vejovis: Precision and Ranking

Subject	Bug Report #1	Bug Report #2
Drupal	31 / 40	1 / 4
Ember.js	1 / 2	1 / 3
Joomla	1 / 88	1 / 88
jQuery	2 / 108	-
Moodle	2 / 37	1 / 37
MooTools	2 / 2	1 / 2
Prototype	1 / 6	1 / 2
Roundcube	4 / 79	-
TYPO3	1 / 187	1 / 1
WikiMedia	6 / 24	1 / 71
WordPress	13 / 30	1 / 170

#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
 - <http://ece.ubc.ca/~frolino/projects/vejovis>
- 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

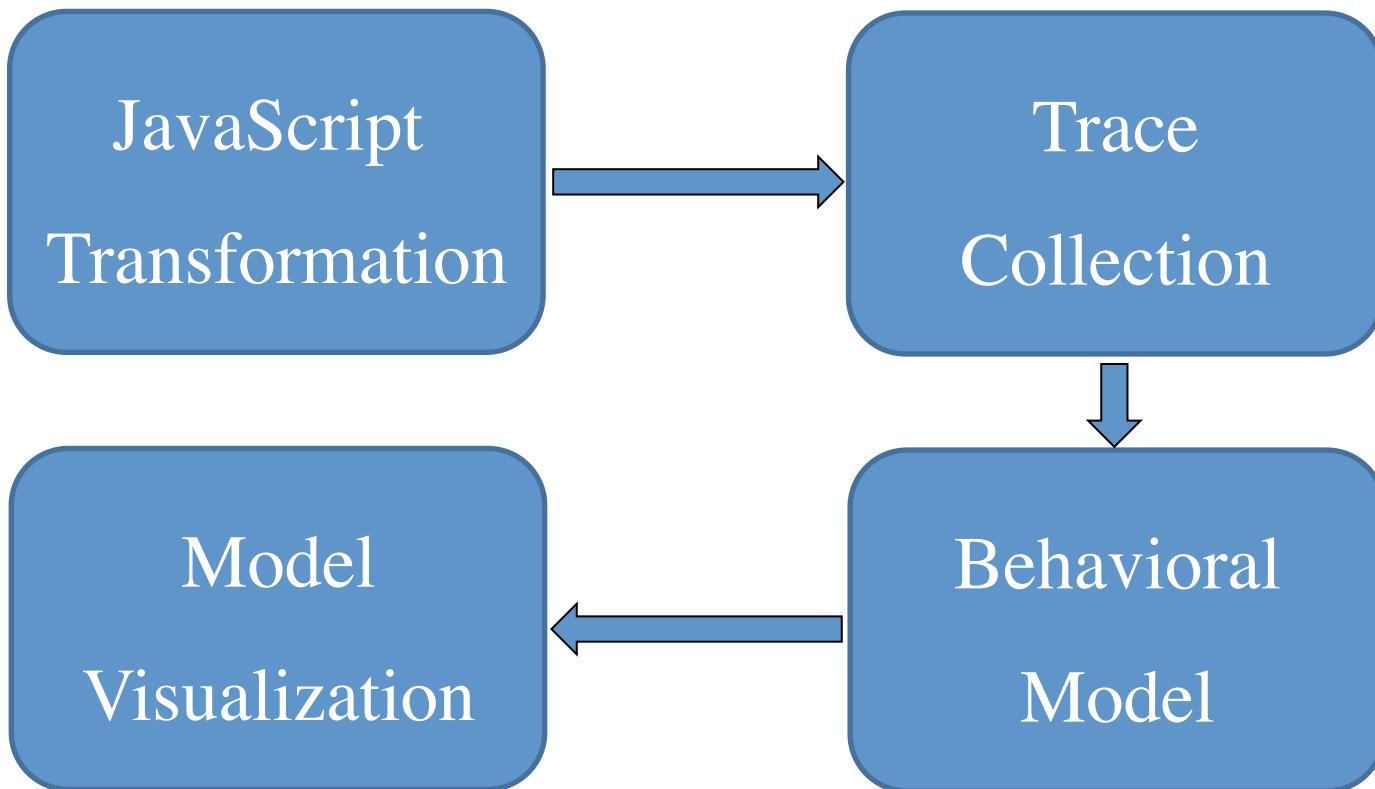
Talk Outline

- Our Prior Work: DOM-Related Faults [ESEM'13]
- AutoFlex: 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

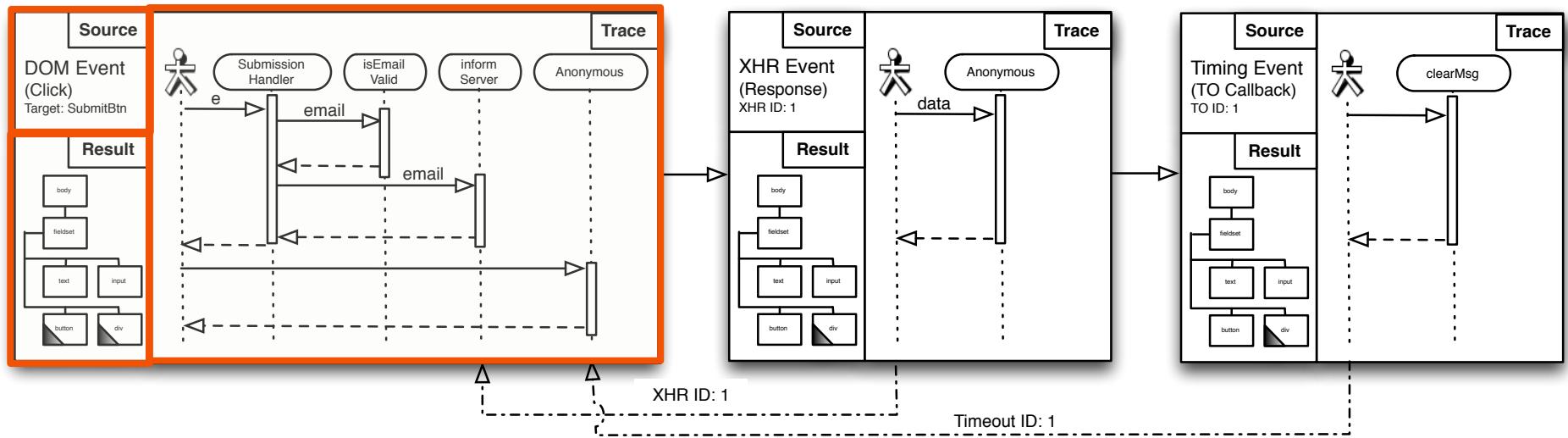
Clematis: Motivation

- **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

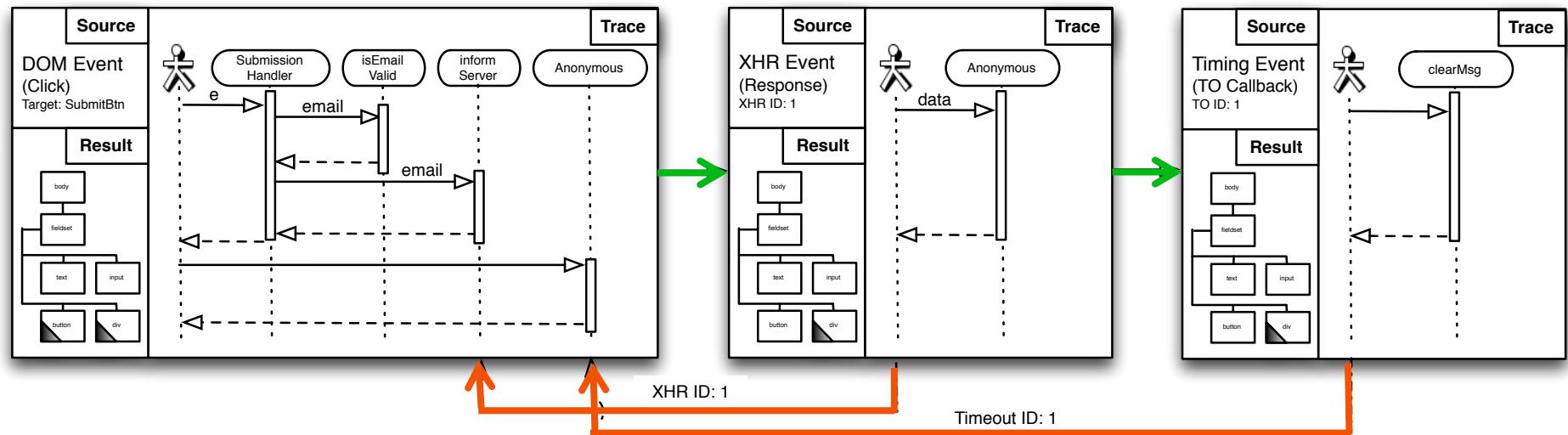
Clematis: Approach



Clematis: Model Episodes

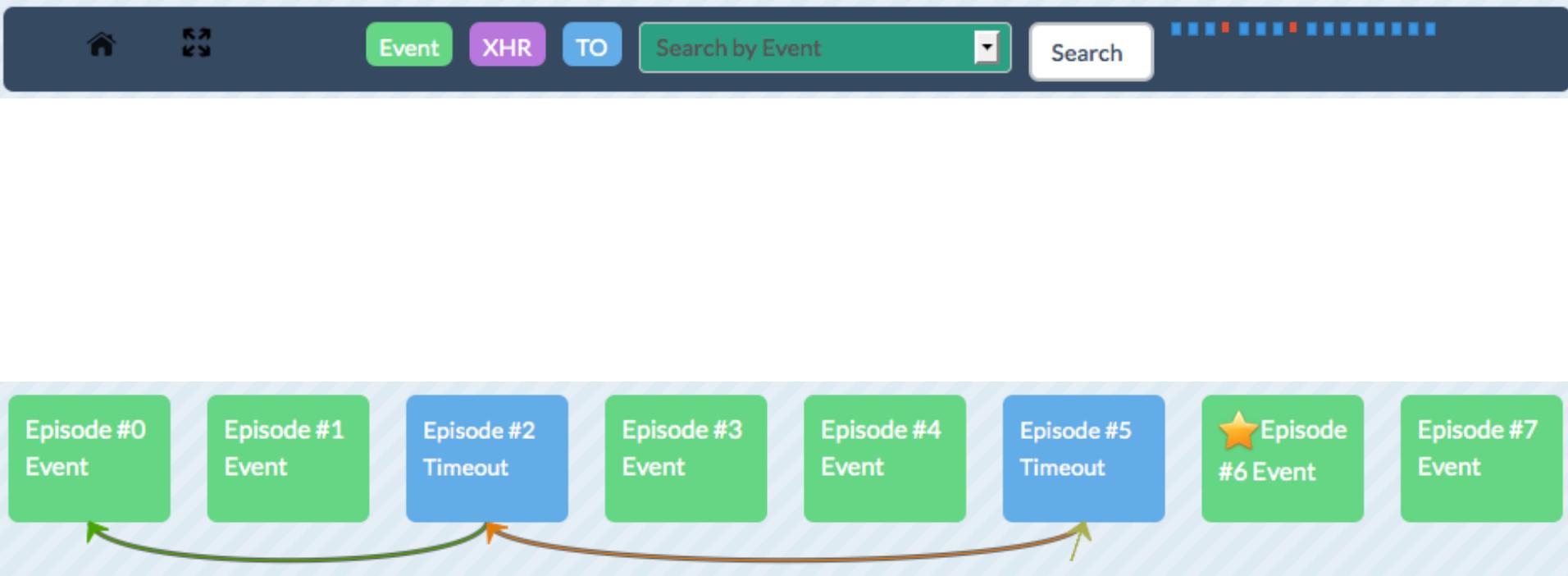


Clematis: Model Links



Temporal
Causal

Clematis: Visualization



CLEMATIS: VISUALIZATION

The screenshot shows the CLEMATIS visualization interface with two main panels. The top navigation bar includes icons for home, refresh, and search, followed by tabs for Event (selected), XHR, TO, and a search bar. Below the navigation is a timeline represented by a series of colored dots.

Episode #3 Trace:

- Source:** "click"
- Trace:**

Event type:click	onclick()	ss_next()
ss_update()	hideElem(x)	dg(x)
inlineElem(x)	Event type:load	updateNumOfLoads()
storeUserInformation()	sendStatsToServer()	ss_loaddone()
onload()		
- Dom Mutations:** "text" "removed", "text" "removed", "text" "added", "text" "added"

Episode #7 Trace:

- Source:** TO:0
- Trace:**

TID: 0	ss_slideshow()	ss_update()
hideElem(x)	dg(x)	inlineElem(x)
ss_run()	TID: 0	TID: 0
Event type:load	sts_data_collection()	updateNumOfLoads()
storeUserInformation()	sendStsToServer()	ss_loaddone()
onload()		
- Episode #7 Event:** (represented by a green box)

Zoom Level 1



Event

XHR

TO

Search by Event



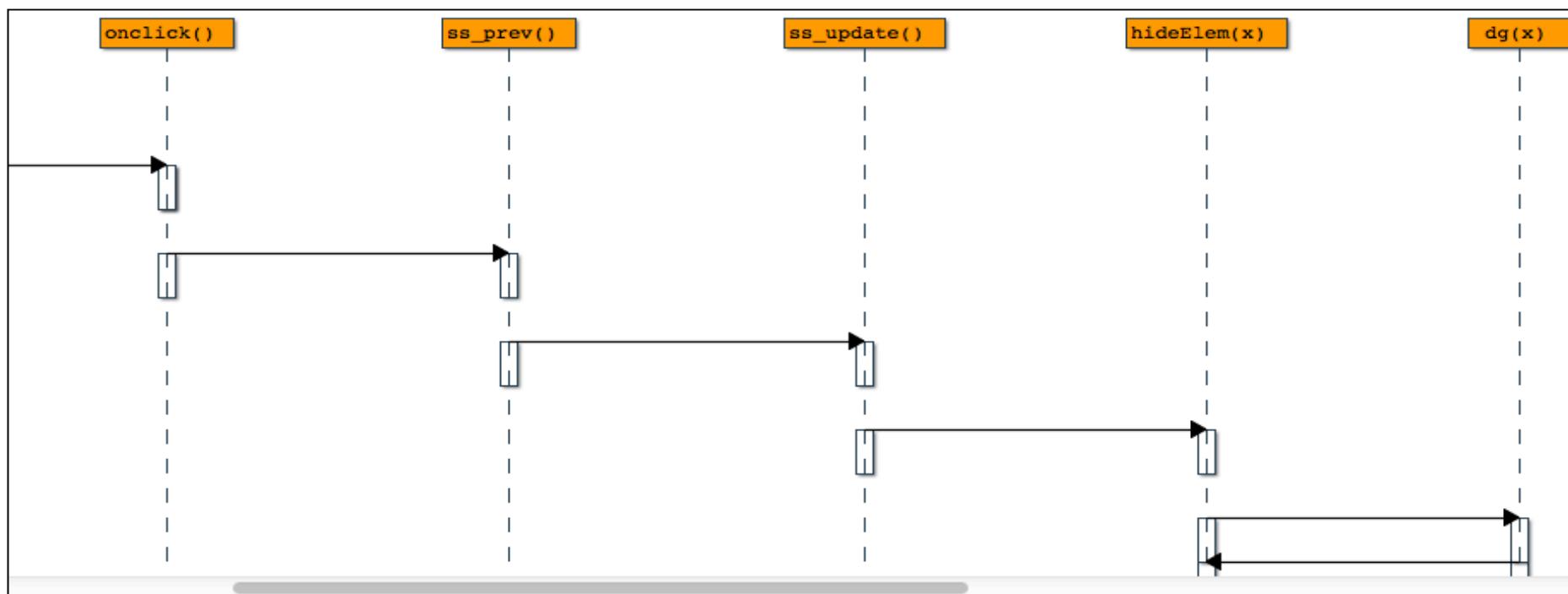
Search

Episode 5

Event Type

DOM mutations

Trace of Episode 5



phorm.js

```
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();
    }
}
```

Zoom Level 2

Copyright: Karthik Pattabiraman, 2014

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



Accuracy

Task	Improvement	Task	Improvement
T1	(39%↑)	T1	(67%↑)
T2	(48%↑)	T2	(41%↑)
T3	(68%↑)	T3	(20%↑)
T4	(32%↑)	T4	(68%↑)

Task	Description
T1	Following control flow in presence of asynchronous events
T2	Finding DOM mutations caused by a DOM event
T3	Locating the implementation of a malfunctioning feature
T4	Detecting control flow in presence of event propagation

Clematis: Summary

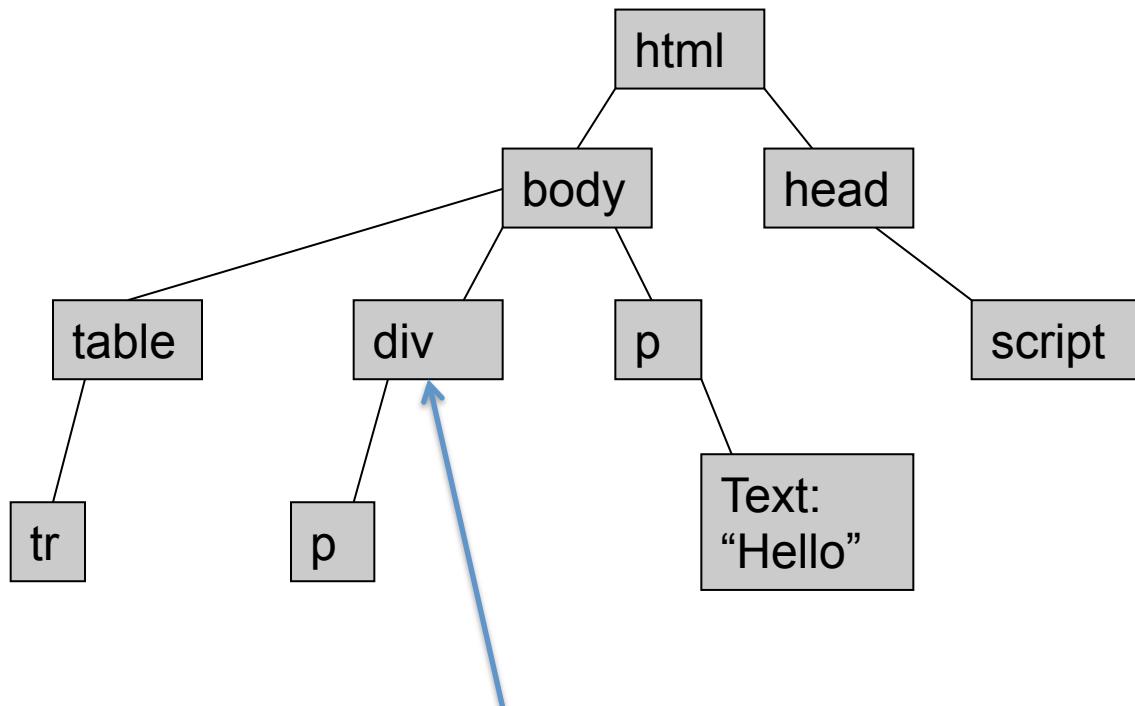
- Freely available:
 - <https://github.com/saltlab/clematis>
- Ability to visualize JavaScript events and DOM states
 - No changes to server side or client side code
 - Causal dependencies between events incl. AJAX requests
 - DOM state changes and event propagation in the DOM
- Significantly improved task duration and accuracy compared to other state-of-the-art tools

Talk Outline

- Our Prior Work: DOM-related Faults [ESEM'13]
- AutoFlex: 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

Completion: Motivation

- Provide code-completion for DOM-JavaScript interactions

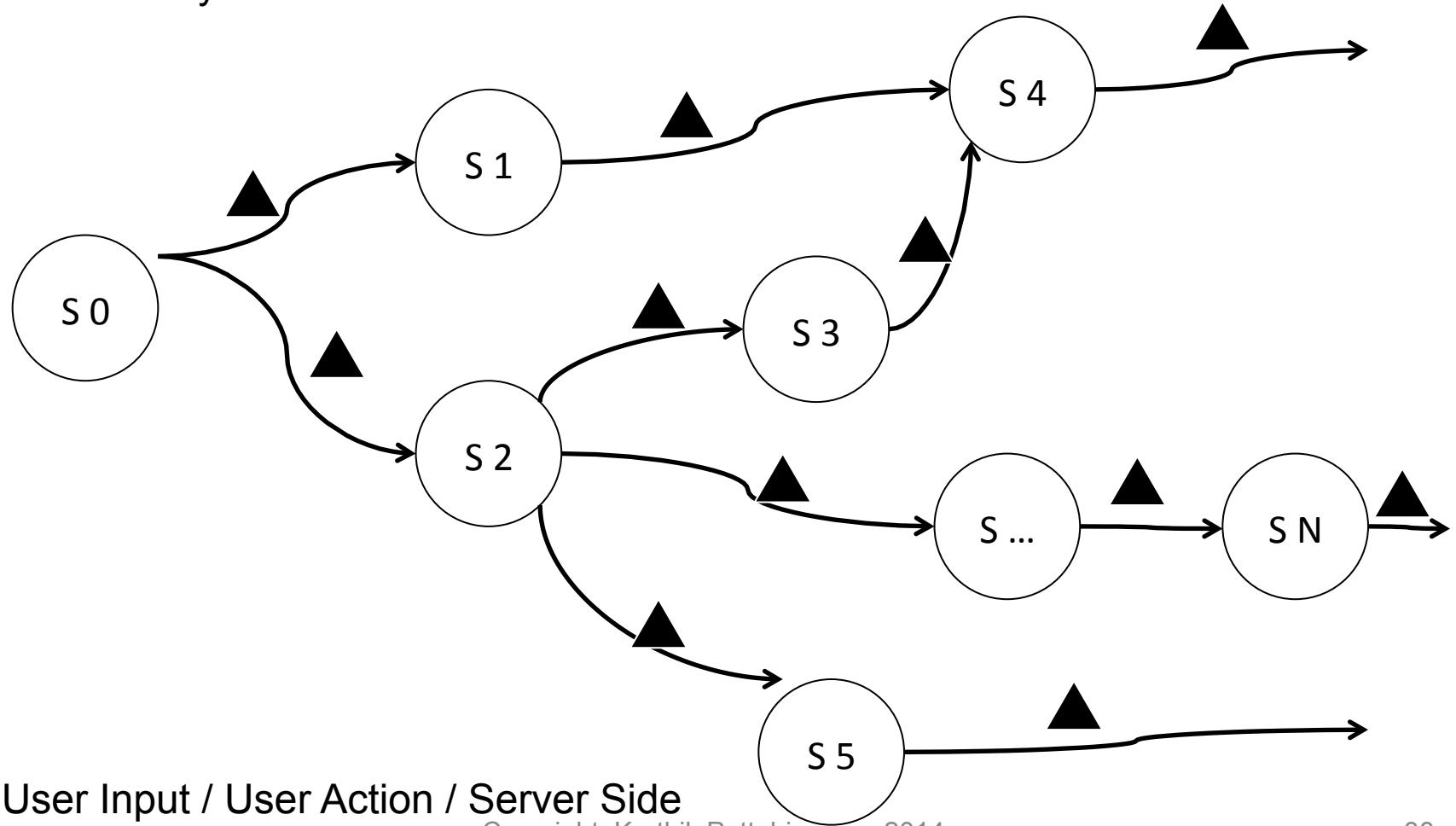


Want to retrieve element with id “elem”

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

Completion: Challenge

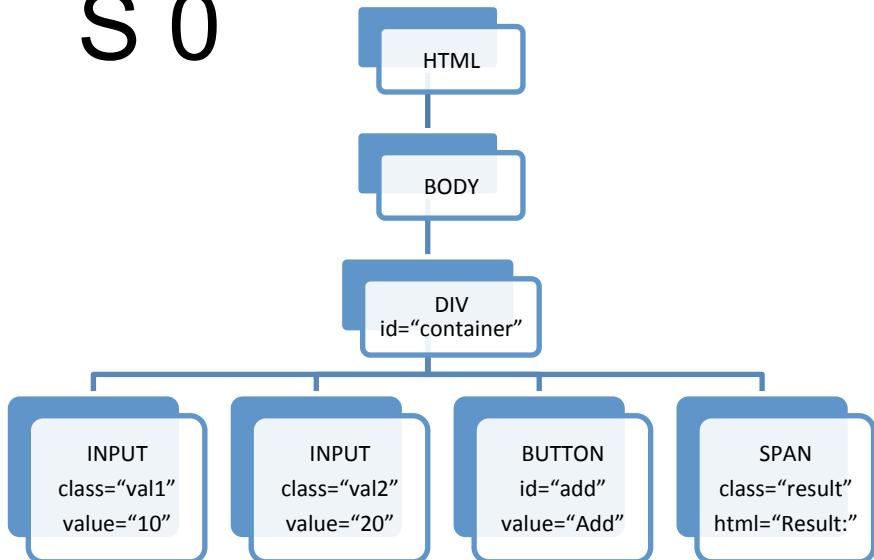
Potentially infinite number of DOM states !



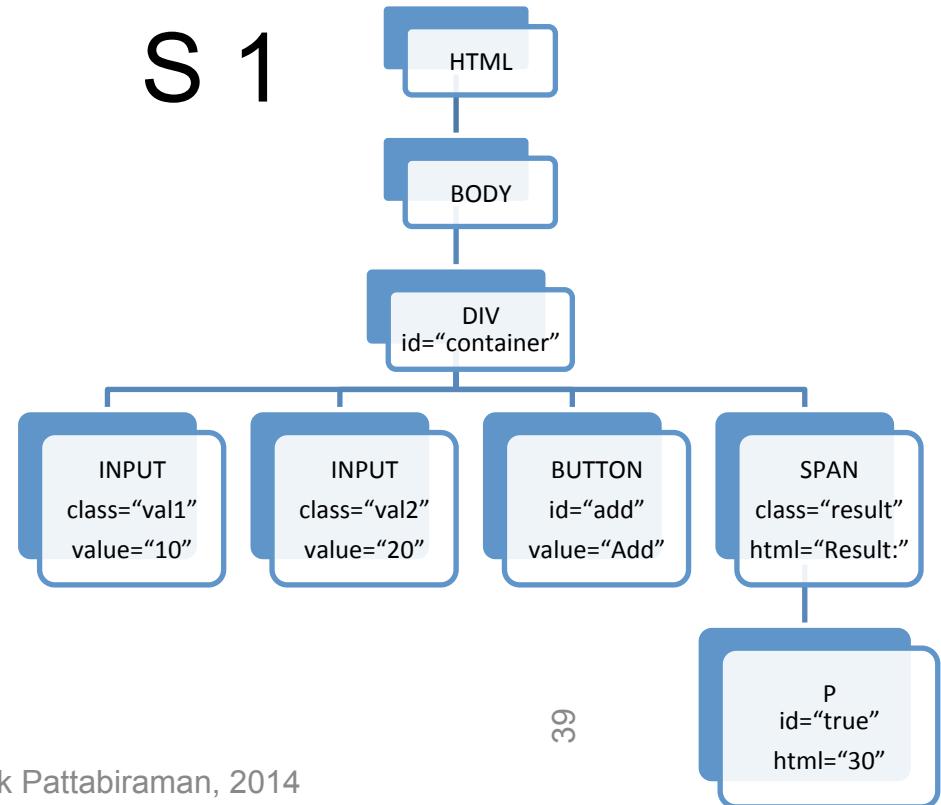
Completion: Intuition

DOM states exhibit patterns

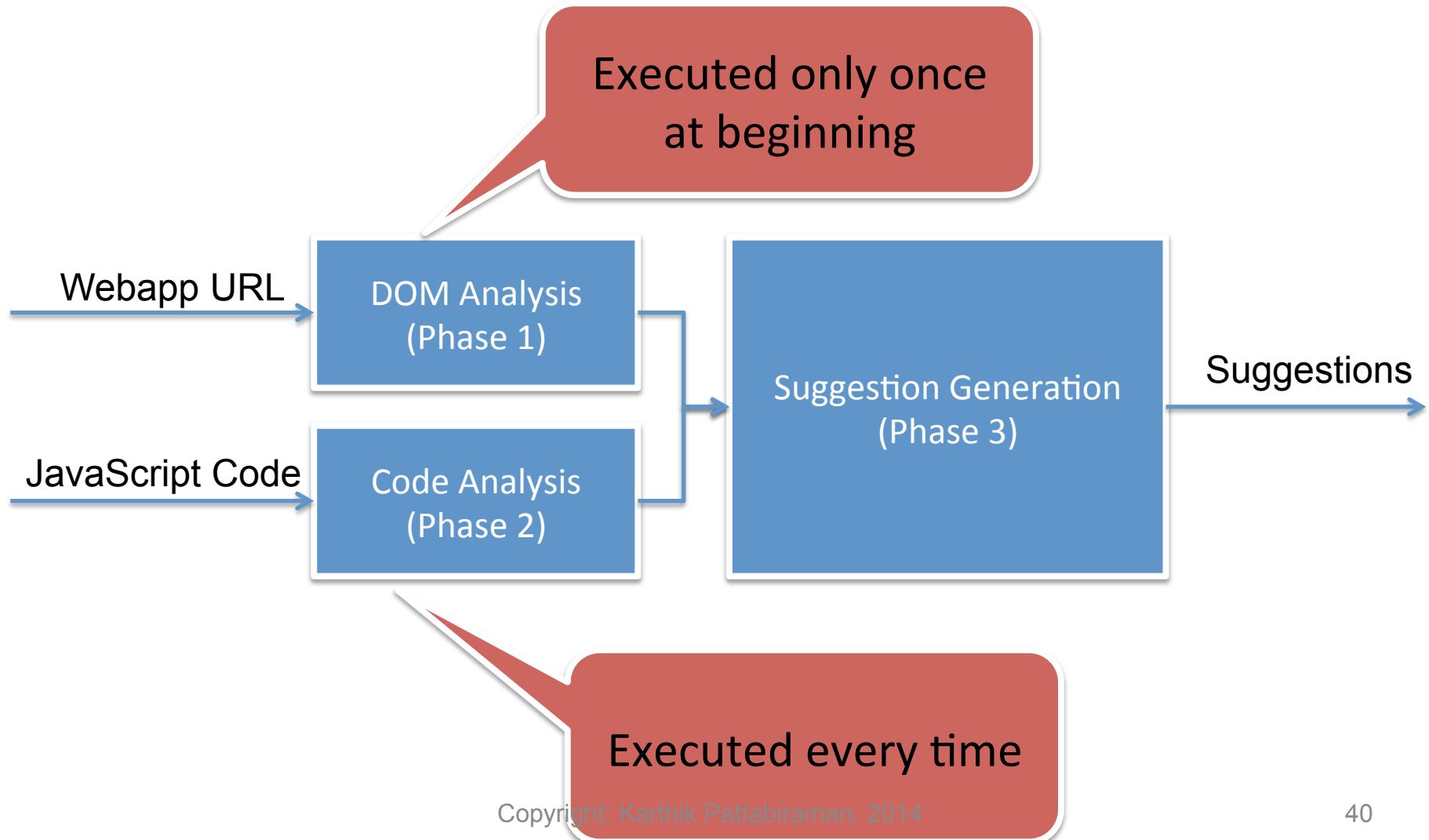
S 0



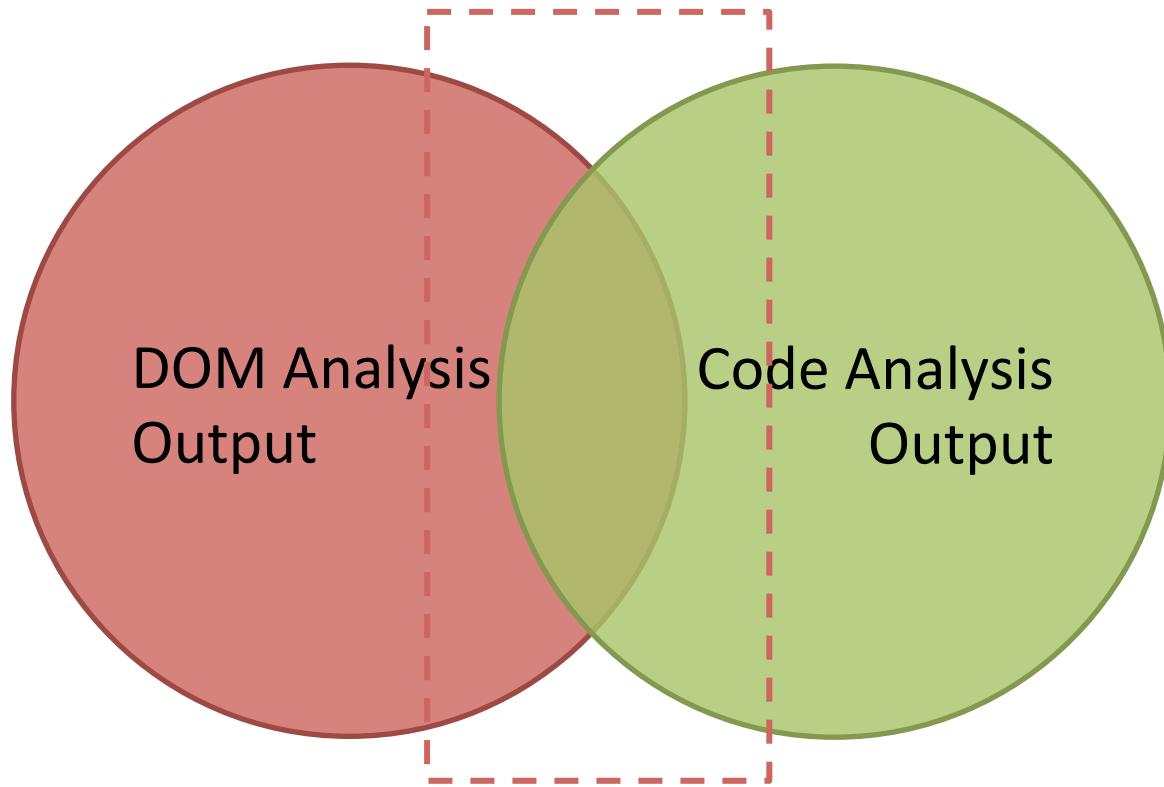
S 1



Dompletion: Approach



Dompletion: Suggestion Generation



Suggestions

Dompletion: Screenshot

```
1 a = document.getElementById('maincol').innerHTML;-
2 -
3 if(a == "header"){-
4     elem = document.getElementById('headerBar');-
5 } else {-
6     elem = document.getElementById('photoBoxes');-
7 }-
8 elem.getElementsByClassName('')  
Path: 0 VeryTitle          (span) DOM Level: 1
Path: 1 photoBox            (div) DOM Level: 1
Path: 0 topHeadAround       (a) DOM Level: 2
Path: 1 titlePhotoBox       (span) DOM Level: 2
Path: 1 darkdot              (span) DOM Level: 2
Path: 1 spc                  (span) DOM Level: 2
Path: 1 rate                 (select) DOM Level: 2
Path: 1 dot                  (span) DOM Level: 3
```

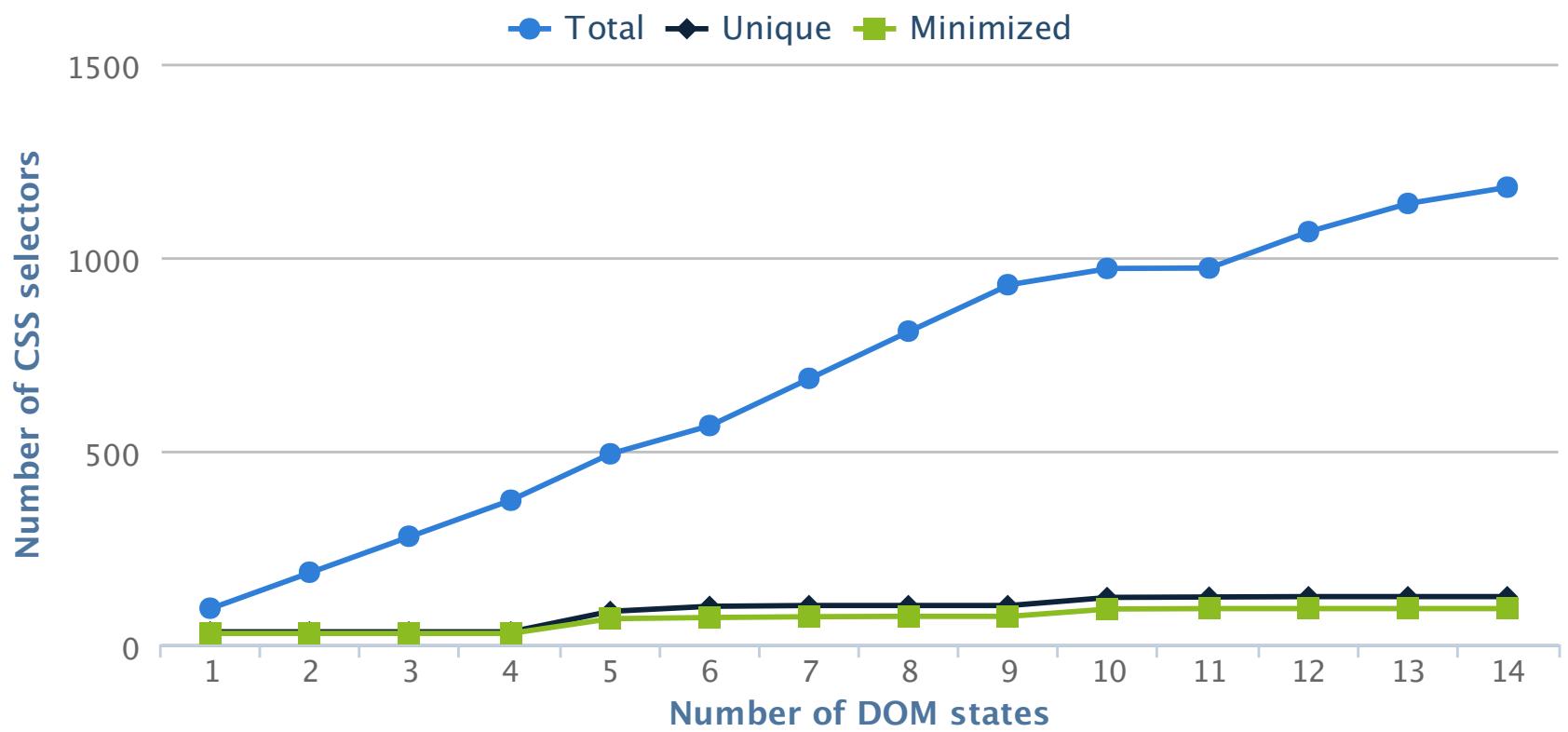
Implemented in the Brackets IDE



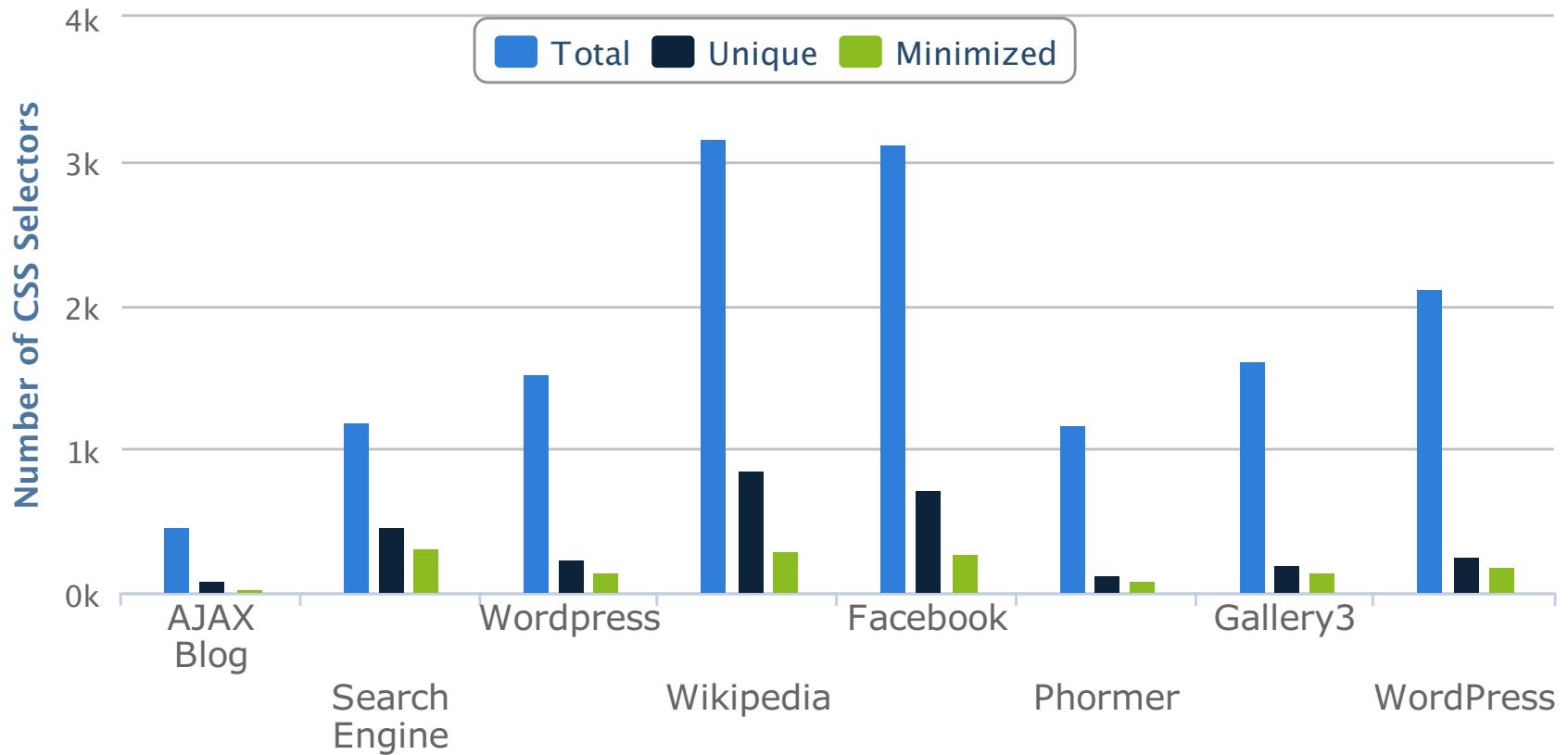
Dompletion: Evaluation

- 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?

Completion: Convergence (RQ1)



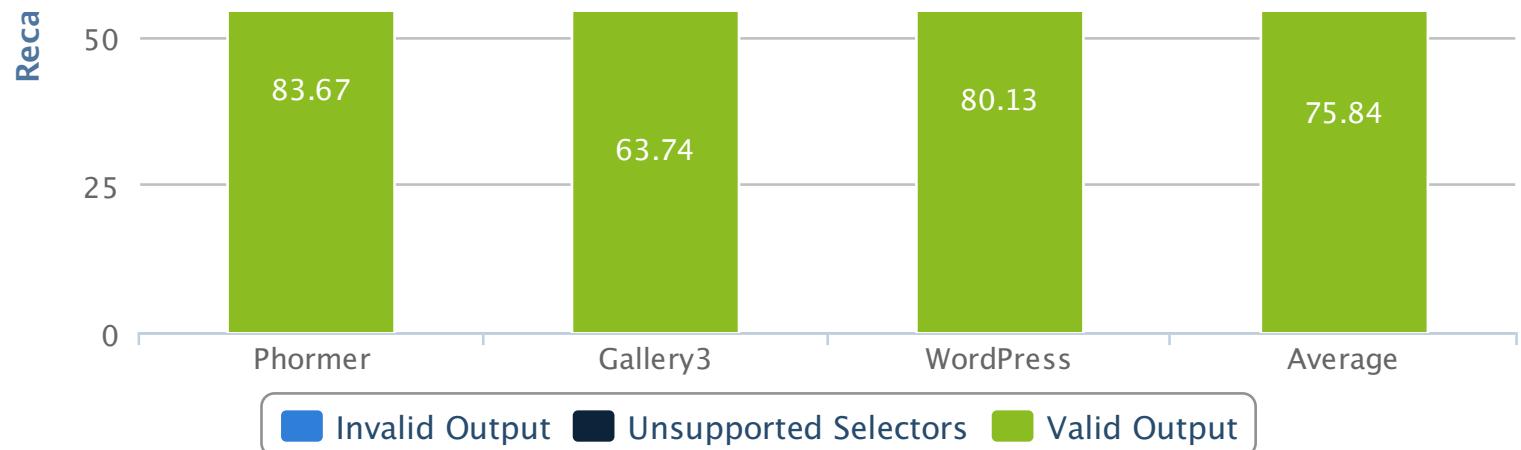
Completion: Convergence (RQ1)



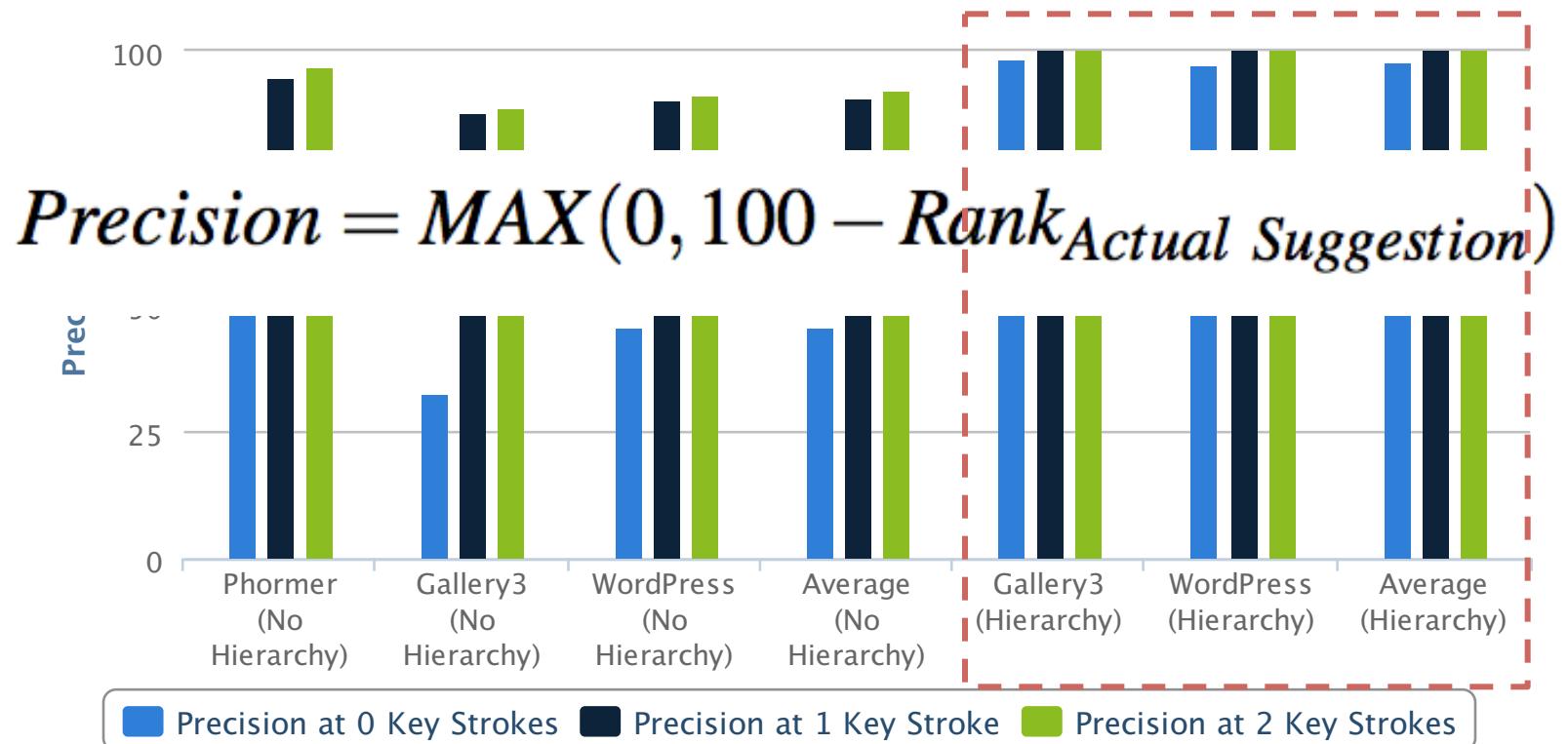
Completion: Accuracy (RQ2)



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



Dompletion: Accuracy (RQ2)



Completion: User Study (RQ3)

- 9 Participants
- 4 Tasks

Group	Description	No. of Participants	Average Time	Precision	Recall
Group A	Using Dompleiton	5	1m 43s	90.83%	97.5%
Group B	Without Completion	4	4m 28s	76.25%	47.5%

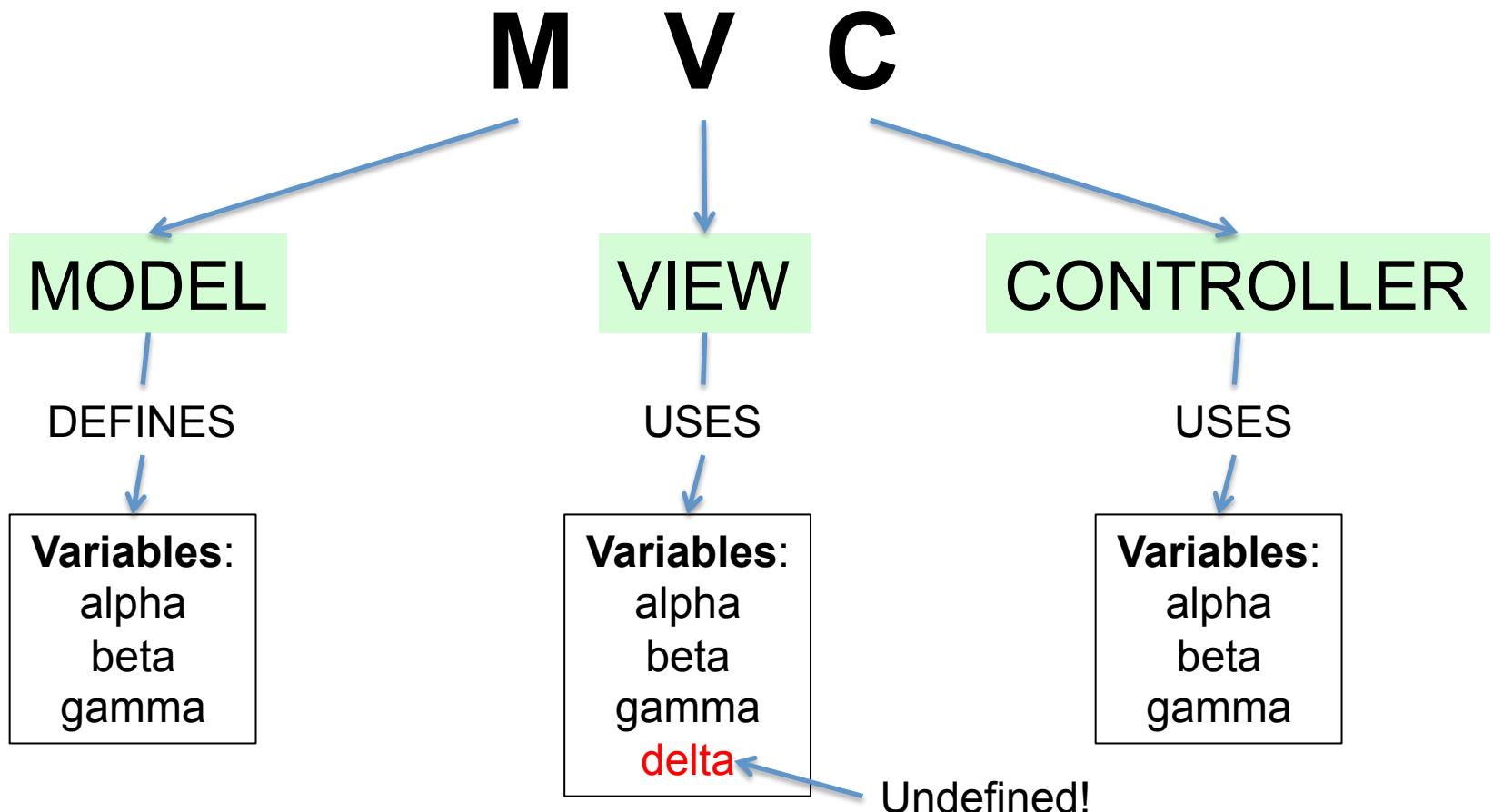
Dompletion: Summary

- 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.

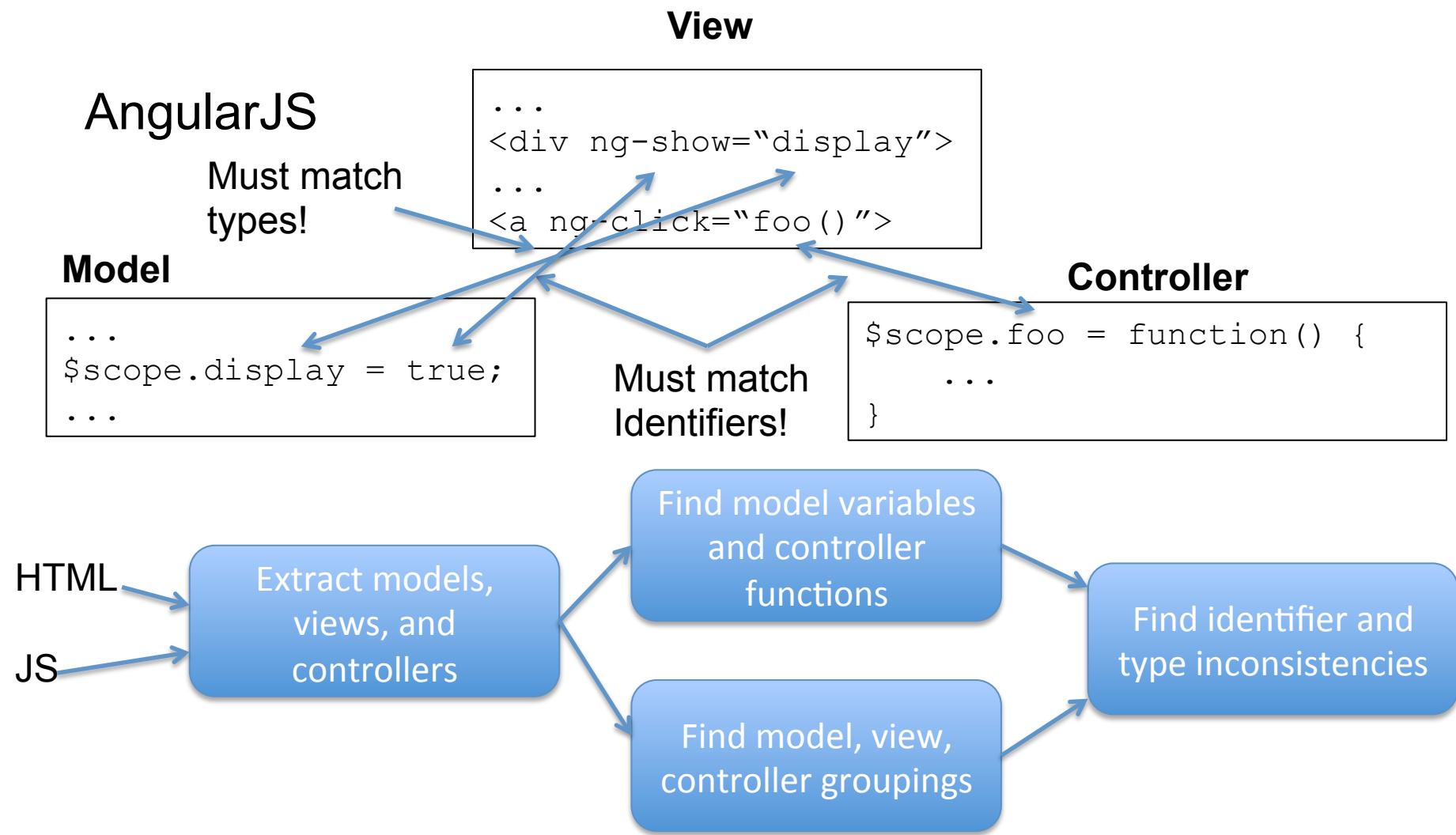
Talk Outline

- Our Prior Work: DOM-related Faults [ESEM'13]
- AutoFlex: 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



Ongoing Work: Aurebesh - 2



Ongoing Work: Aurebesh - 3

Fault Injection Experiment

95% Recall and
100% Precision

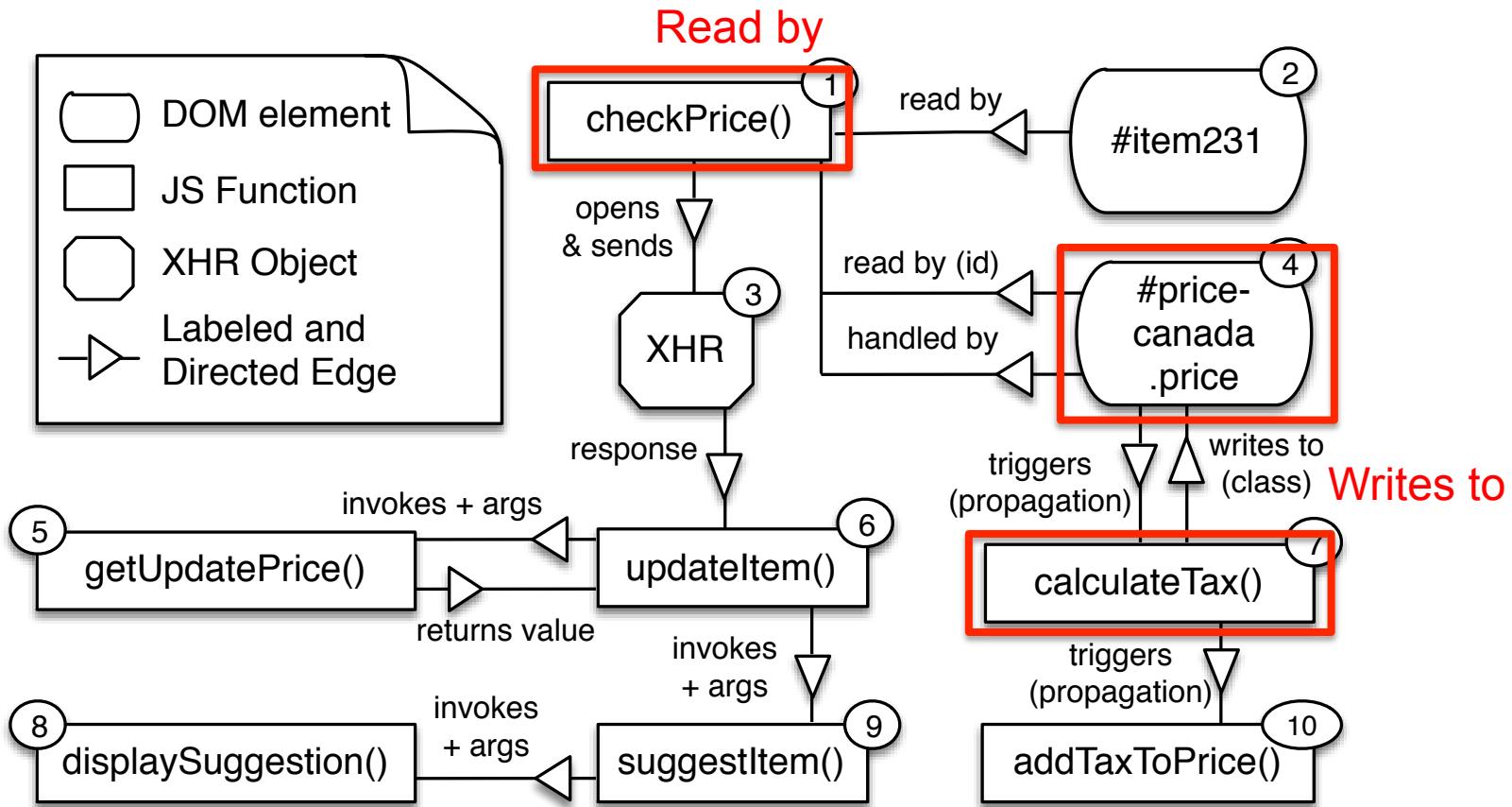
Test on Real Applications

Detected 13 real-world bugs in 9 applications

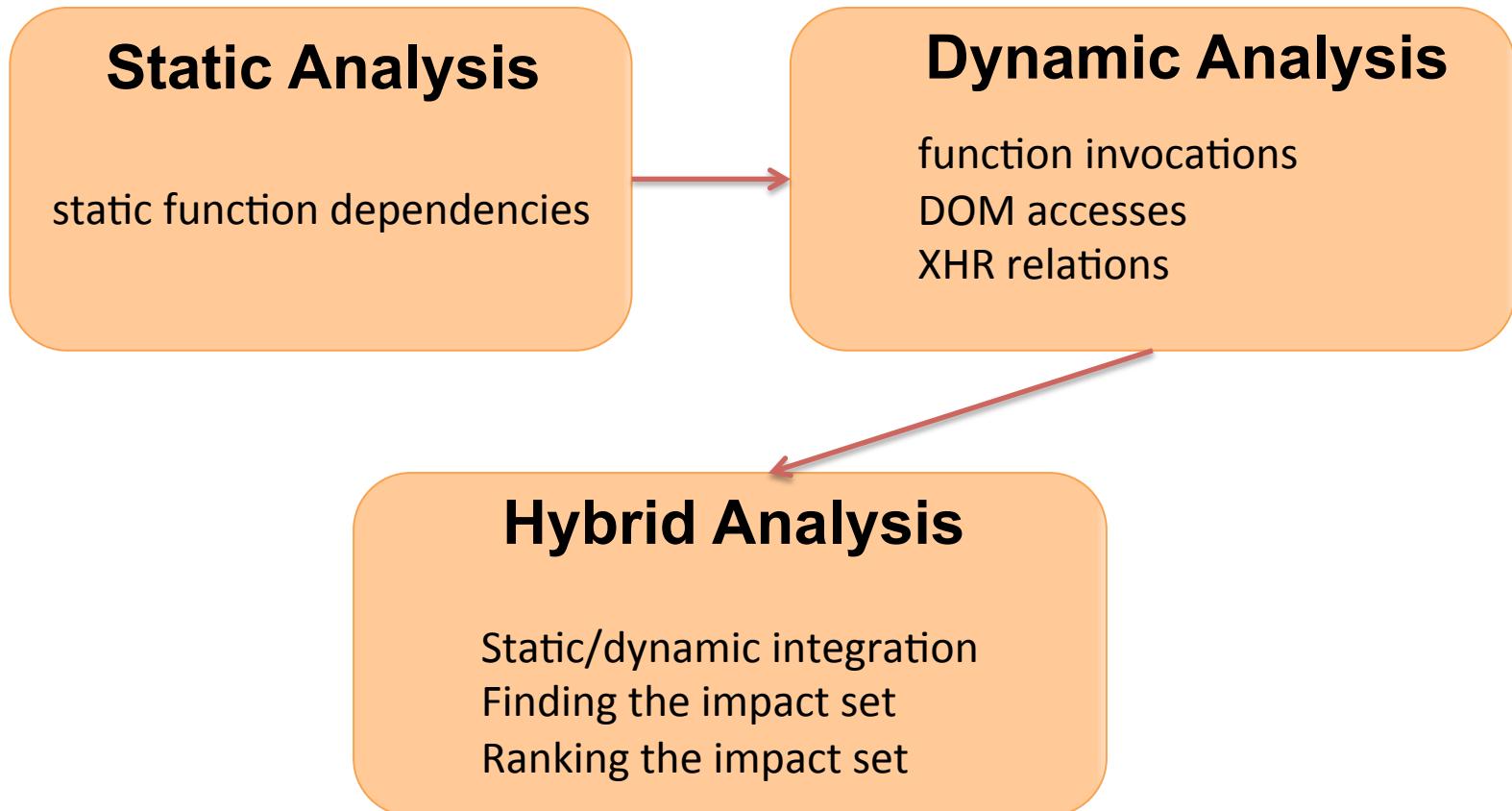
Performance

Ran for ~0.8 s in application with 5K LOC

Ongoing Work: ToChal - 1

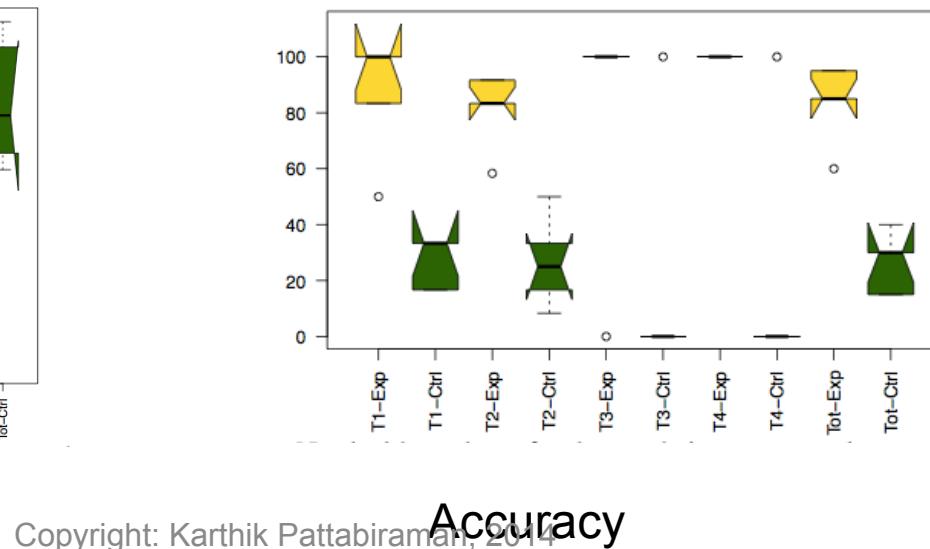
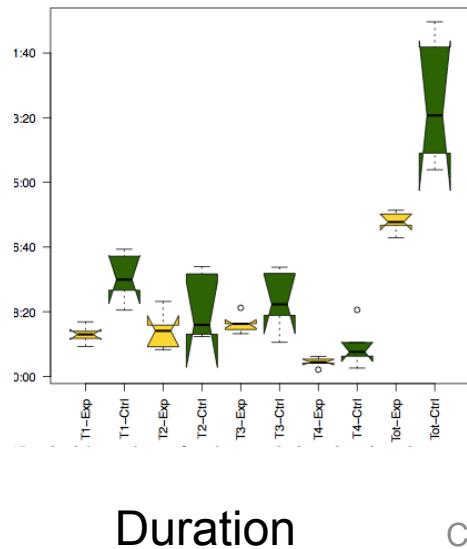


Ongoing work: ToChal - 2



Ongoing Work: Tochal - 3

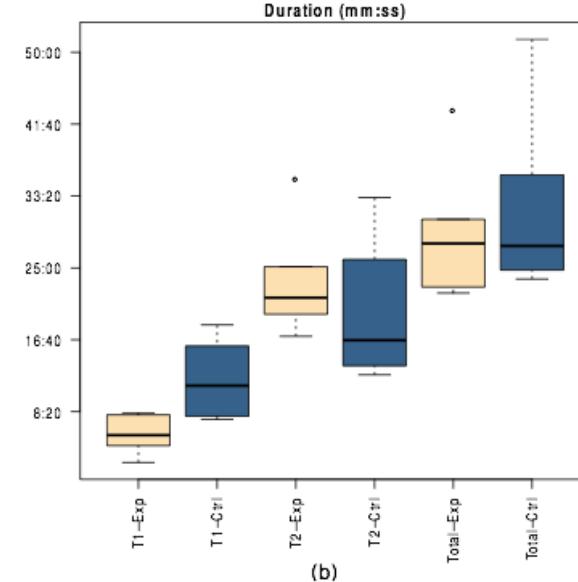
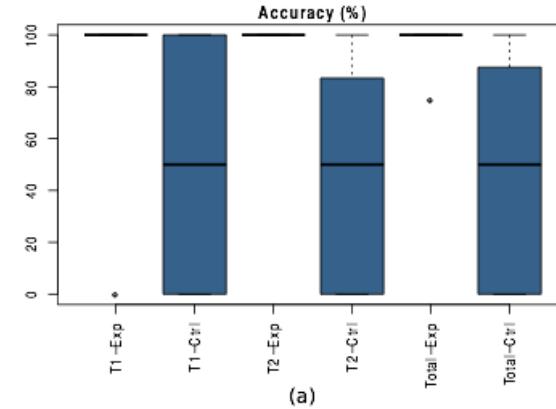
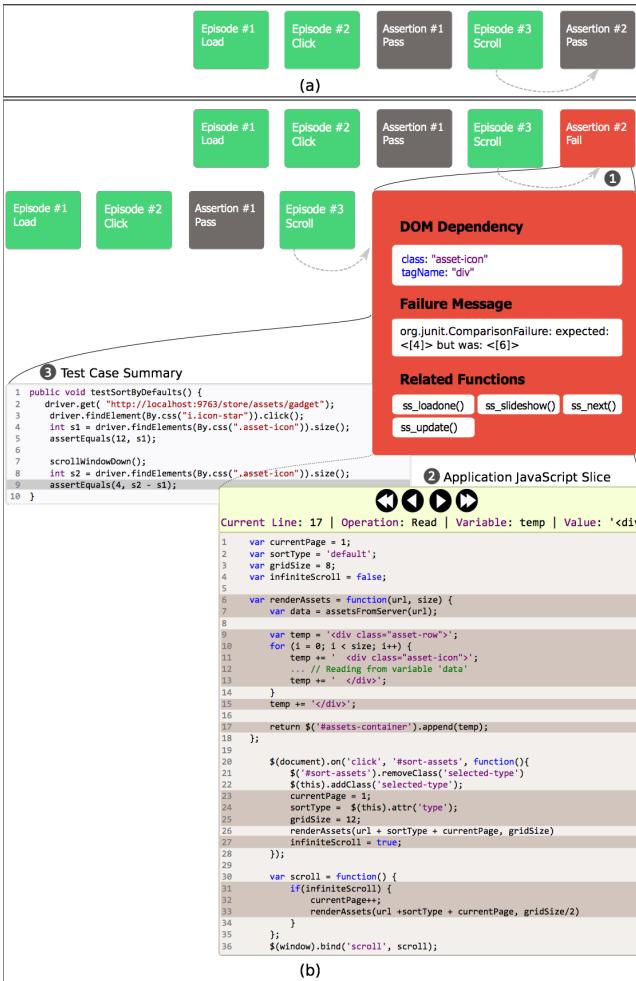
- Empirical study: validation of impact through DOM
- User experiment: 10 professional web developers
 - Experimental group: Tochal / control group: Chrome
 - Results: improved developers' speed and accuracy
 - 44% faster, 70% more accurate answers



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>

Talk Outline

- Our Prior Work: DOM-related Faults [ESEM'13]
- AutoFlex: 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

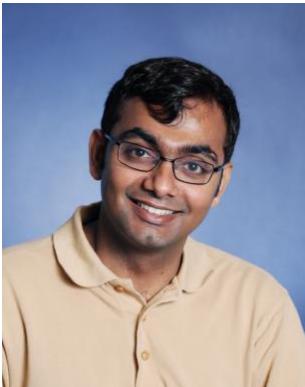
- What about faults that do not involve DOM-JS interactions ?
 - Currently account for about 35% of faults, but may increase as we mitigate DOM-related faults
 - Three related efforts
 - Non-DOM related API faults [FSE'14]
 - Type-related faults (Typedevil) [Berkeley'14]
 - Race conditions: Webracer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



Karthik Pattabiraman



Ali Mesbah



Kartik Bajaj



Frolin Ocariza



Saba Alimadadi



Sheldon Sequira



Microsoft®
Research