Tutorial 5: Modern Web Applications' Reliability Engineering

Karthik Pattabiraman
Electrical and Computer Engineering
University of British Columbia (UBC)

Introductions

- Who am I ?
  - Associate professor at Univ of British Columbia, Vancouver, Canada
  - Research interests in software reliability and security
  - Been working on web applications' reliability for past six years (from 2010)

- Who are you ?
  - Name, affiliation, and web application experience (if any)

- IEEE Reliability Certificate (please add your name to signup sheet)

Modern Web Applications: Examples

facebook

YouTube

Google

amazon

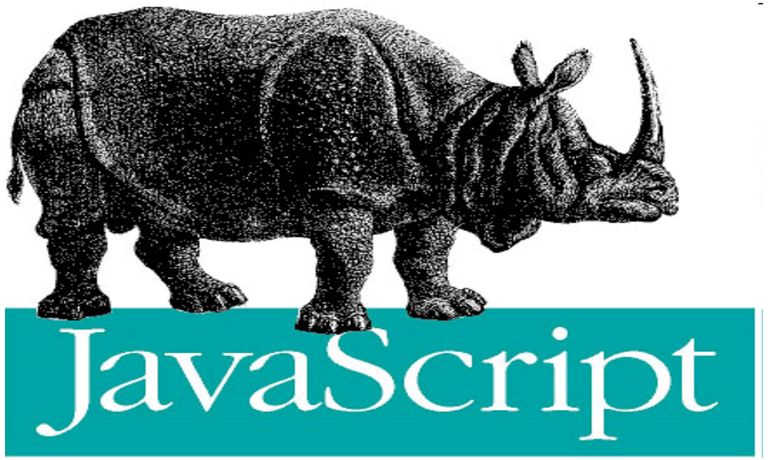
tumblr.

Modern Web Application: Amazon.com

Web 2.0 applications allow rich UI functionality within a single web page

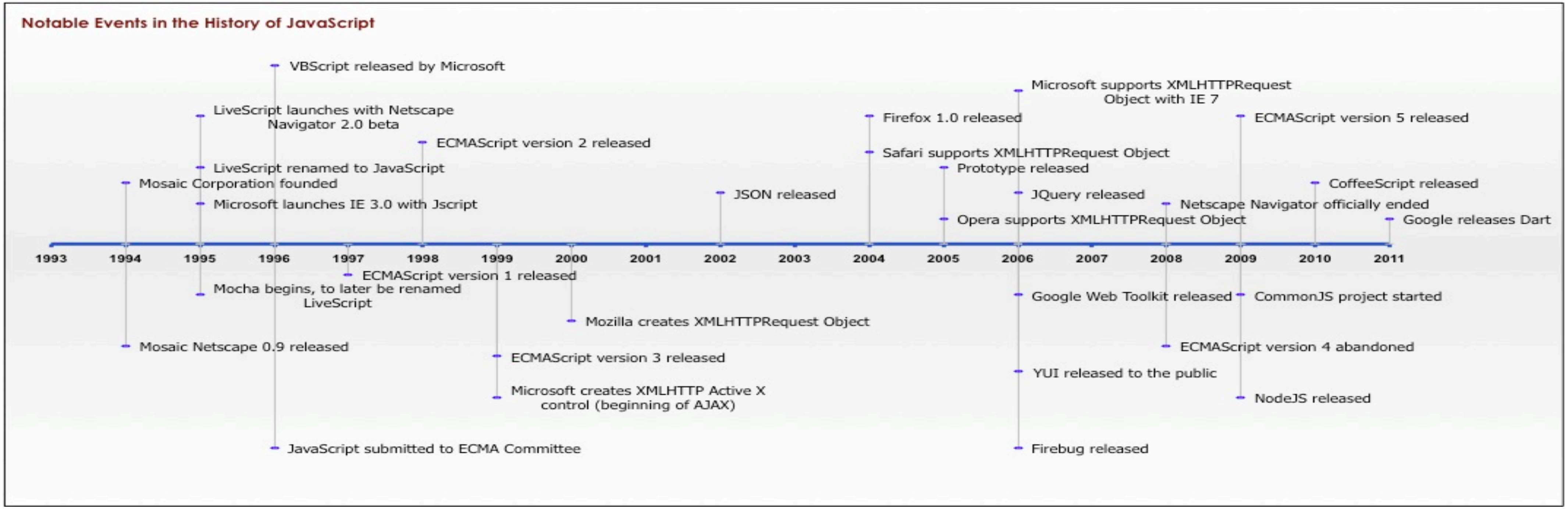
Modern Web Applications: JavaScript

- JavaScript: Implementation of ECMAScript standard
  - Client-Side JavaScript: used to develop web apps
- 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



JavaScript: History

Brief History of JavaScript (Source: TomBarker.com)

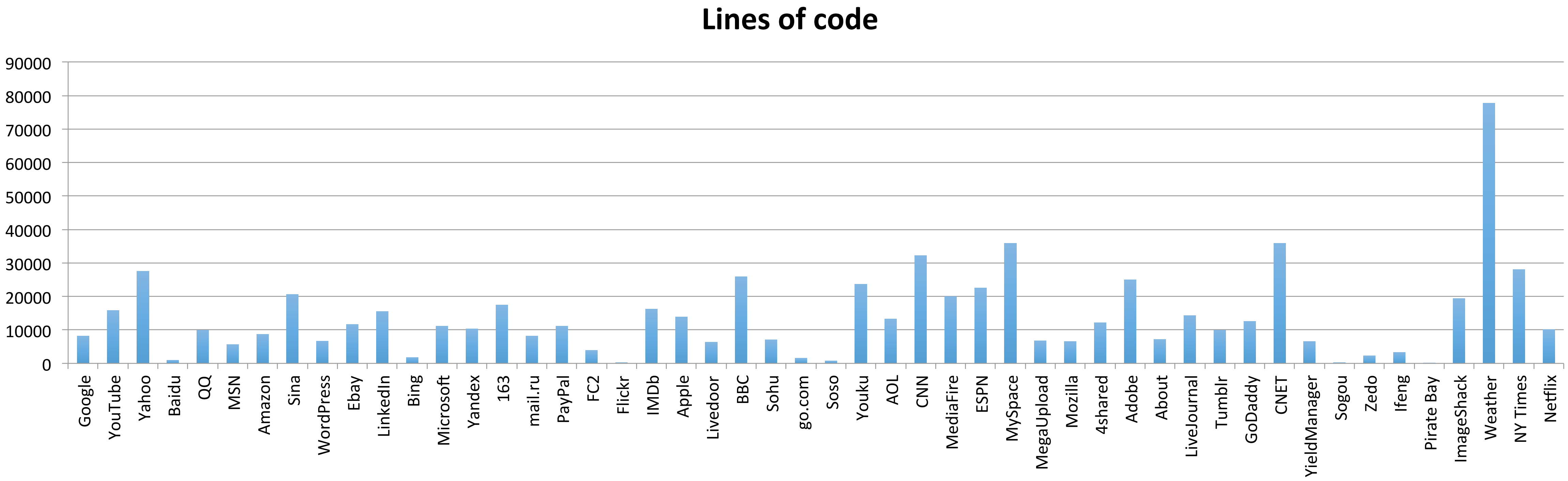


JavaScript (JS) had to “look like Java” only less so, be Java’s dumb kid brother or boy-hostage sidekick. Plus, I had to be done **in ten days** or something worse than JS would have happened

– Brendan Eich (Inventor of JavaScript)

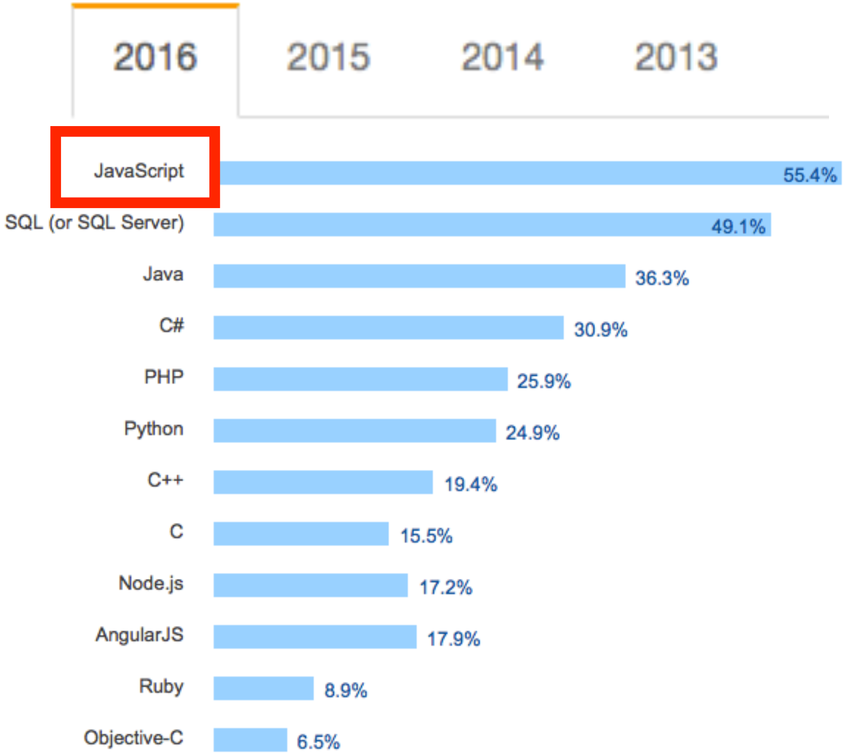
JavaScript: Prevalence

- 97 of the Alexa top 100 websites use JavaScript
- Thousands of lines of code, often > 10,000



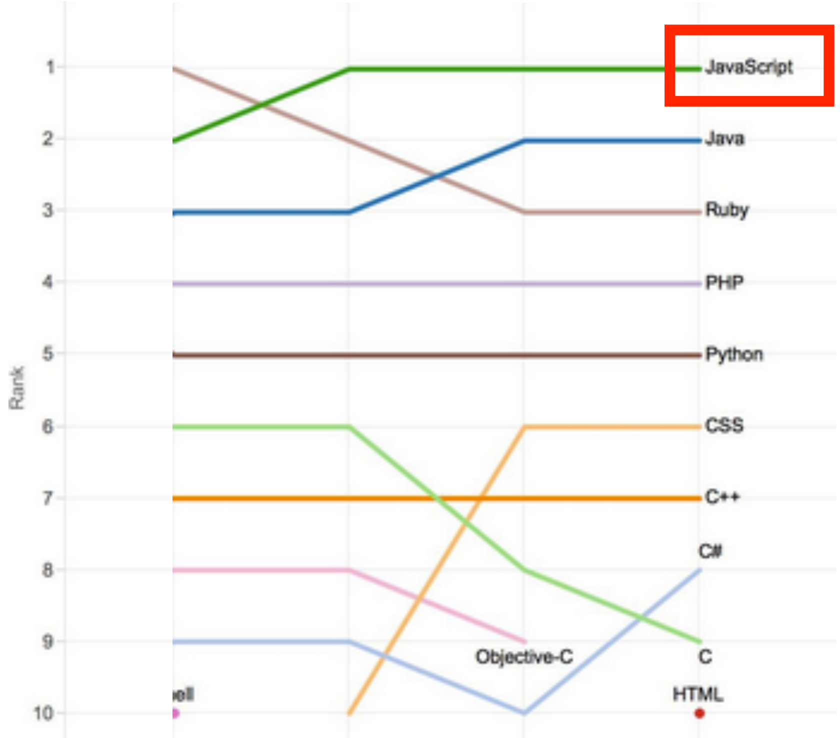
stackoverflow

JavaScript:
Most popular language

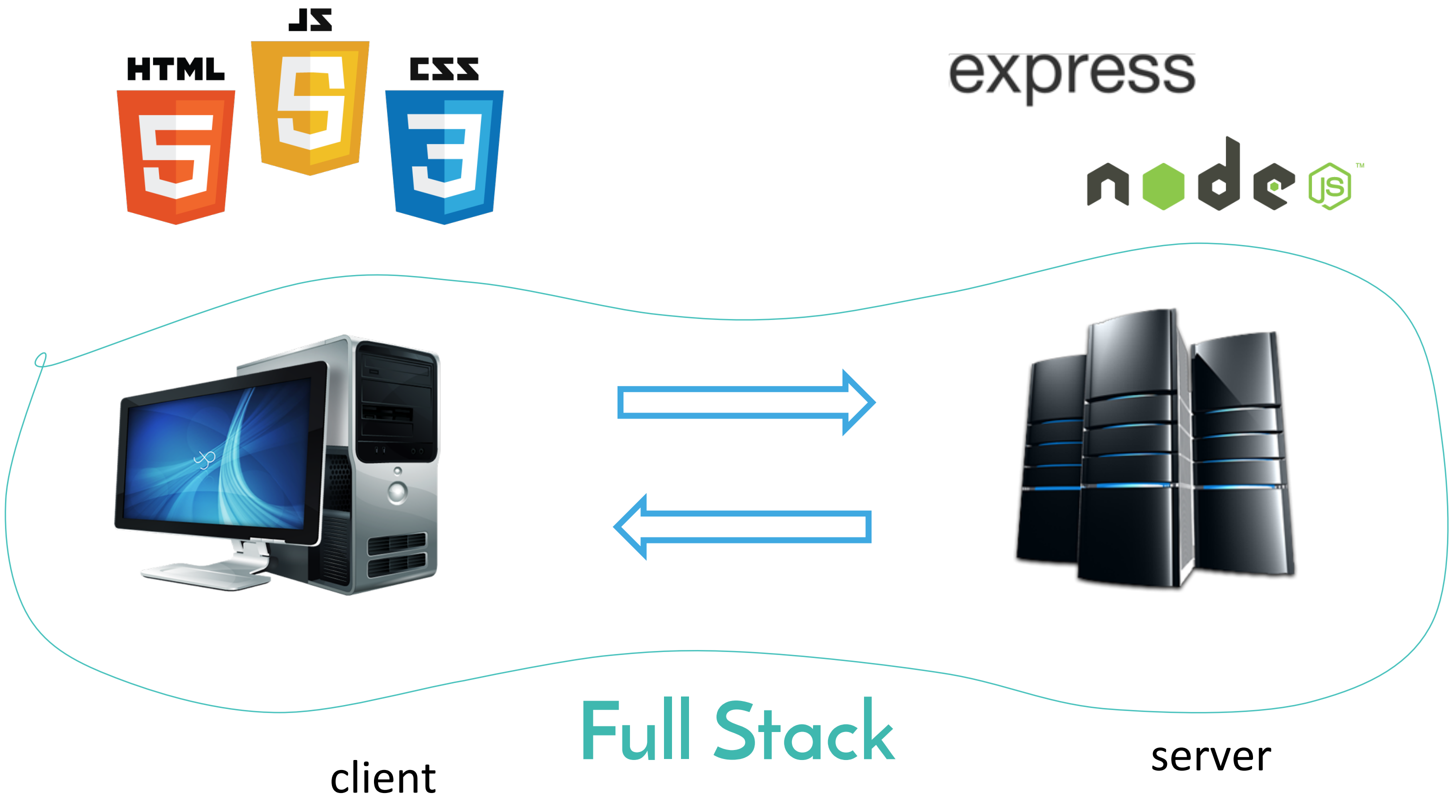


GitHub

JavaScript:
Top languages on GitHub



JavaScript and the Web

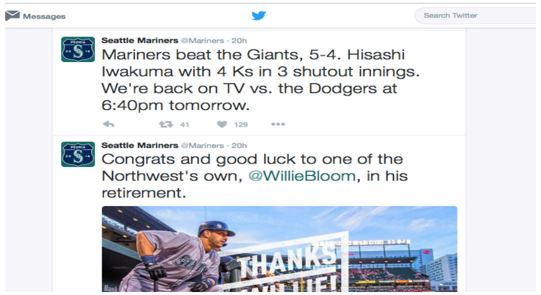


Client-Side JavaScript

EASY TO DEPLOY

Write code

Open browser

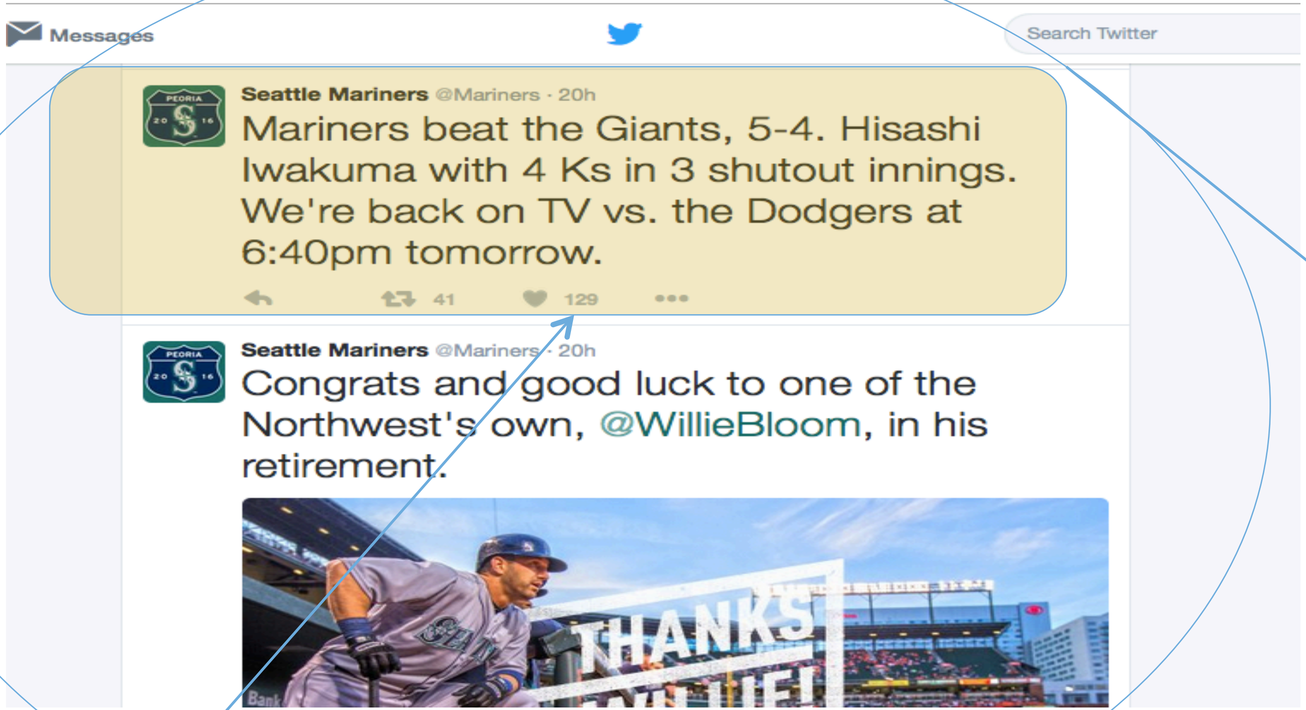


Web app in action!

Client-Side JavaScript

Flexible programming features
→ Rich user interactions

Client-Side JavaScript



Client-side execution → No need to contact the server all the time!

Click to expand tweet

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]

Goal: Study and improve the reliability of JavaScript web applications

Does Reliability Matter ?

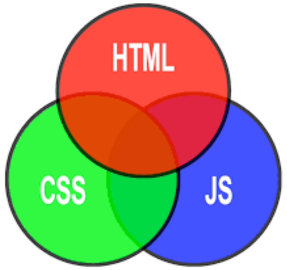
- Snapshot of iFeng.com: Leading media website in China

an error occurred when processing this directive

Analyzing JS Code: Challenges

JS has loose semantics

Lack of standard programming style & JS frameworks



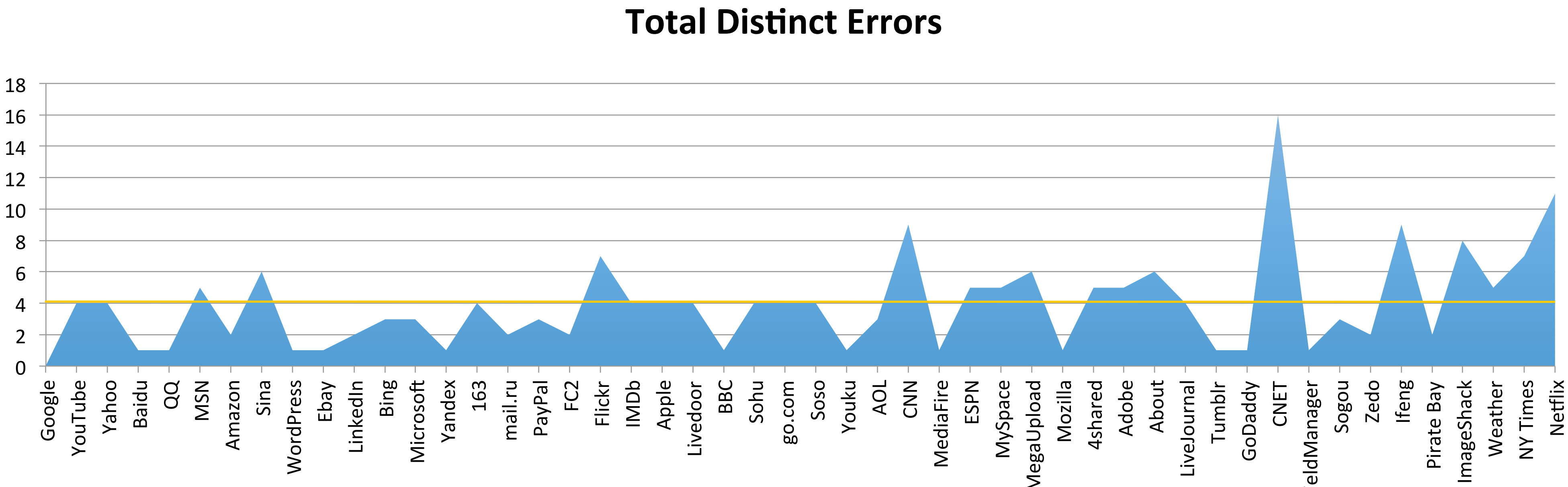
Frequent cross-language interactions

Talk Outline

- Motivation and Goals
- Empirical Study of reliability
- Reliability Improvements
- Program Understanding
- IDE Support
- Other Work and Future Directions

Our Prior Work

- Empirical study based on Console Error Messages: Alexa top 100
- **Popular web applications experience four distinct JavaScript error messages on average across their web-pages [Ocariza - ISSRE'11]**
- Many errors were non-deterministic or dependent on event order - hard to determine the root cause and impact of these errors



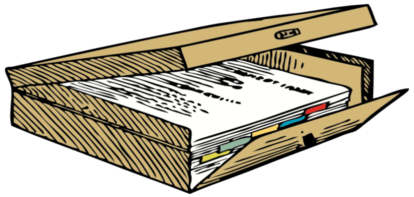
Empirical Study: Research Questions

- What errors/mistakes ***cause*** JavaScript faults?
- What ***impact*** do JavaScript faults have?
- How long does it take to fix these errors?

Bug Report Study of 19 popular and open source JavaScript applications & libraries

- Over a span of 10 years
- Over 500 bug reports

Bug Report Study: Methodology



Collected 502 bug reports from 19 web applications

Qualitatively analyzed and classified bug reports manually

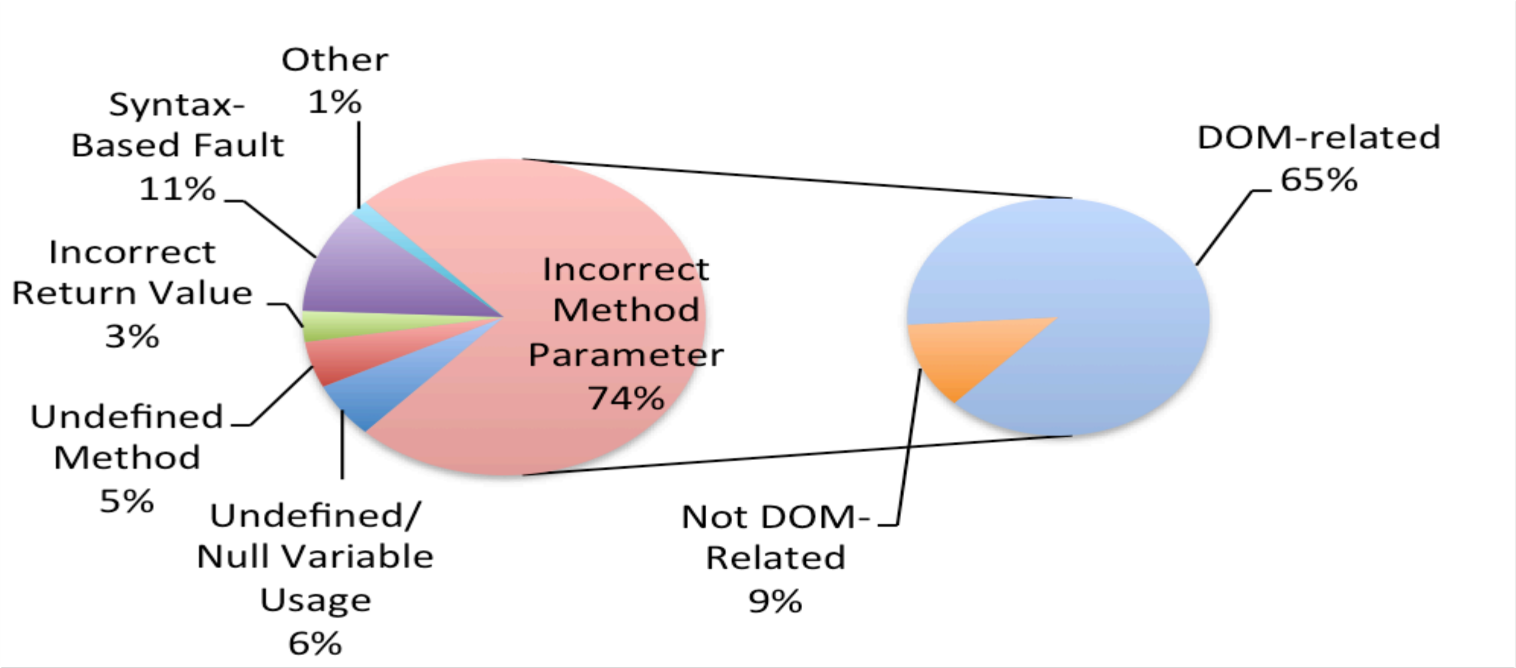
Aggregated data for further analysis

Bug Report Study: Objects

Eight JavaScript Web Applications

Four JavaScript Libraries

Bug Report Study: Categories

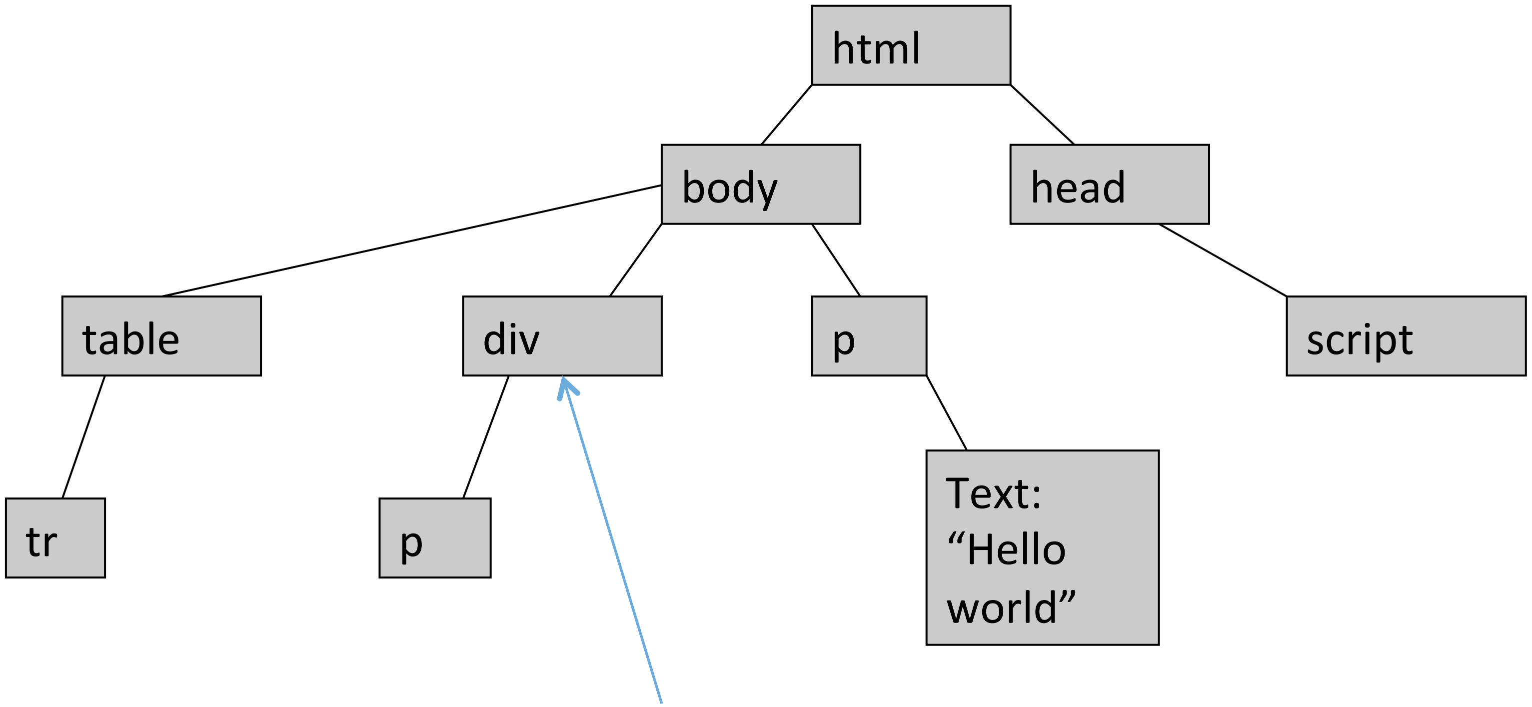


Incorrect Method Parameter Fault: Unexpected or invalid value passed to JS method or assigned to JS property

DOM-Related Fault: The method is a DOM API method

- Account for around two-thirds of JavaScript Faults

Bug Report Study: DOM



Want to retrieve this element

22

Bug Report Study: DOM-Related Faults

JavaScript code:

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

Will return null

Inexistent ID

div

DOM:

id: elem

23

DOM-Related Fault: Example

```
var elem, retrievedStr = [Retrieved via XHR];
var dotsInStr = retrievedStr.split(".").length;
if (dotsInStr == 0) {
      var prefix = "id_";
      elem = $("#" + prefix + retrievedStr);
}
else {
      elem = $(retrievedStr);
}
elem[0].focus()
```

DOM-Related Fault: Example

```
var elem, retrievedStr = [Retrieved via XHR];
var dotsInStr = retrievedStr.split(".").length;
if (dotsInStr == 0) {
      var prefix = "id_";
      elem = $("#" + prefix + retrievedStr);
}
else {
      elem = $(retrievedStr);
}
elem[0].focus()
```

Retrieved string via XHR

DOM-Related Fault: Example

```
var elem, retrievedStr = [Retrieved via XHR];
var dotsInStr = retrievedStr.split(".").length;
if (dotsInStr == 0) {
      var prefix = "id_";
      elem = $("#" + prefix + retrievedStr);
}
else {
      elem = $(retrievedStr);
}
elem[0].focus()
```

Find the number of dots in the string

DOM-Related Fault: Example

```
var elem, retrievedStr = [Retrieved via XHR];
var dotsInStr = retrievedStr.split(".").length;
if (dotsInStr == 0) {
      var prefix = "id_";
      elem = $("#" + prefix + retrievedStr);
}
else {
      elem = $(retrievedStr);
}
elem[0].focus()
```

If there are no dots, prepend “id_” to the string and access it via $(). Otherwise, leave it as is, and access it via $().

DOM-Related Fault: Example

```
var elem, retrievedStr = [Retrieved via XHR];
var dotsInStr = retrievedStr.split(".").length;
if (dotsInStr == 0) {
      var prefix = "id_";
      elem = $("#" + prefix + retrievedStr);
}
else {
      elem = $(retrievedStr);
}
elem[0].focus()
```

UNDEFINED EXCEPTION!

Retrieved string of “editor” would go here even though it has no dots, which would erroneously cause $() to use selector “editor”, which doesn’t match any elements.

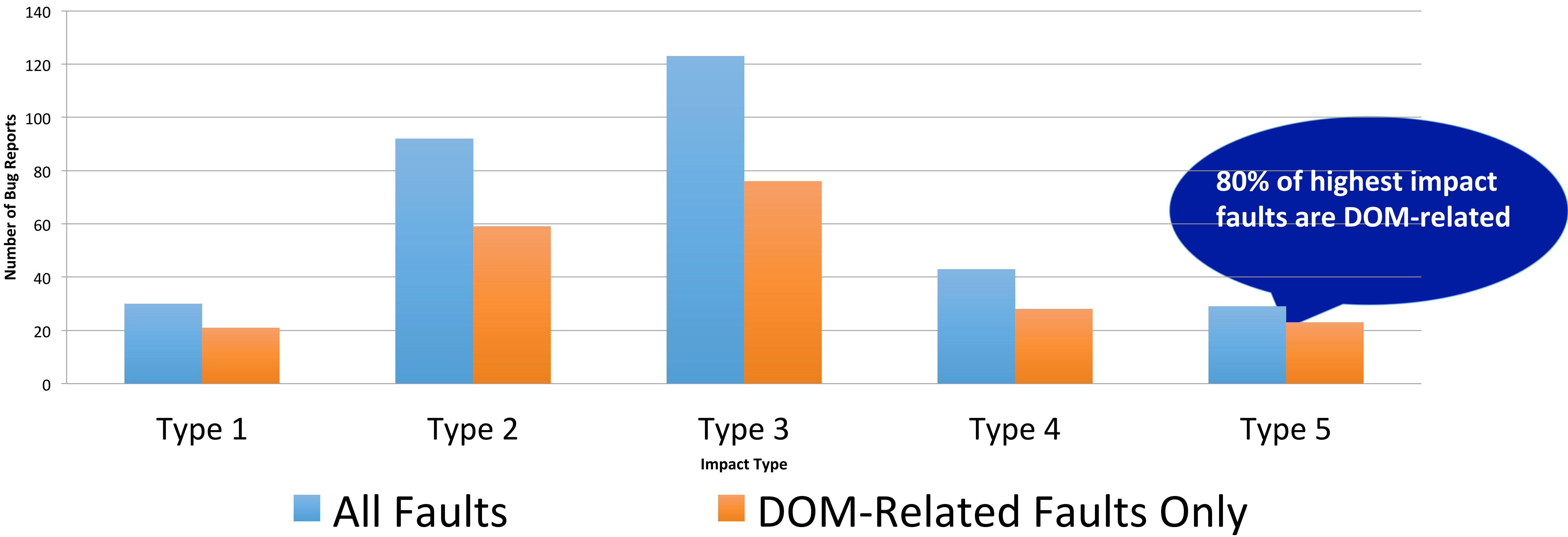
DOM-Related Fault: Example

```
var elem, retrievedStr = [Retrieved via XHR];
var dotsInStr = retrievedStr.split(".").length;
if (dotsInStr == 0) {
      var prefix = "id_";
      elem = $("#" + prefix + retrievedStr);
}
else {
      elem = $(retrievedStr);
}
elem[0].focus()
```

BUG: The assigned value should be retrievedStr.split(“.”).length – 1, as length() always returns at least 1.

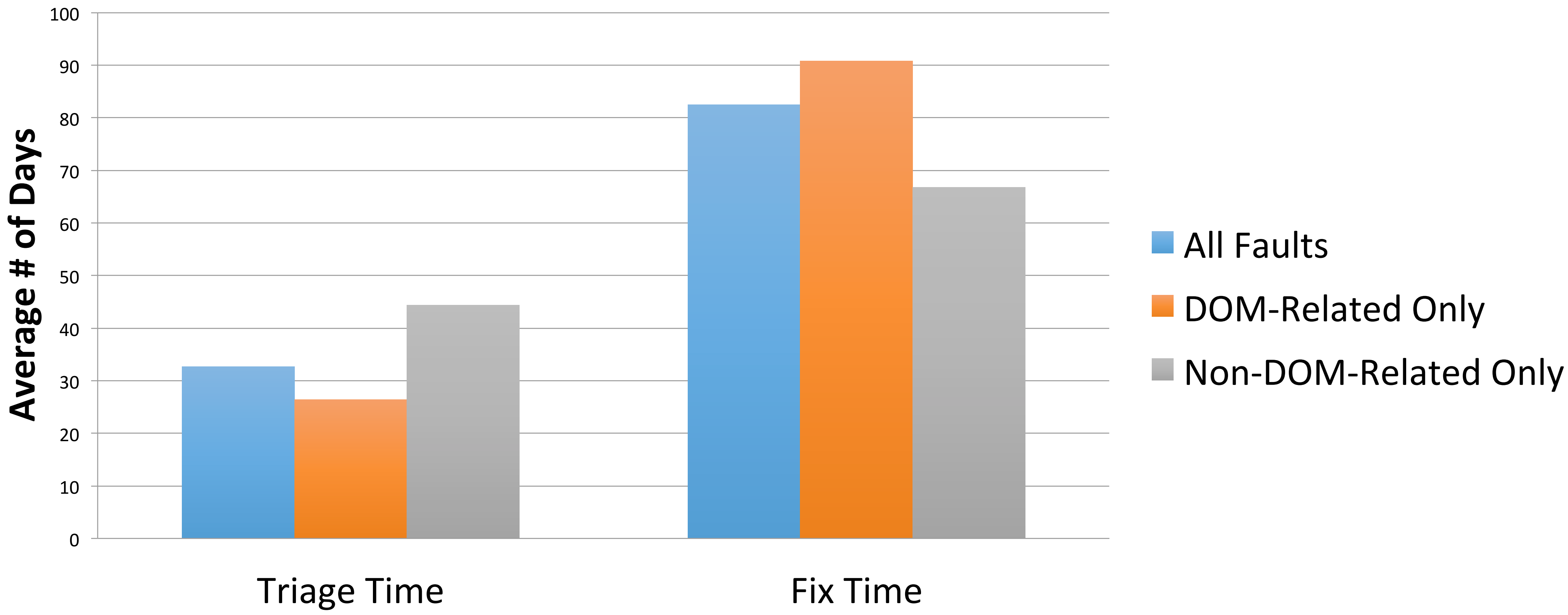
Bug Report Study: Impact

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



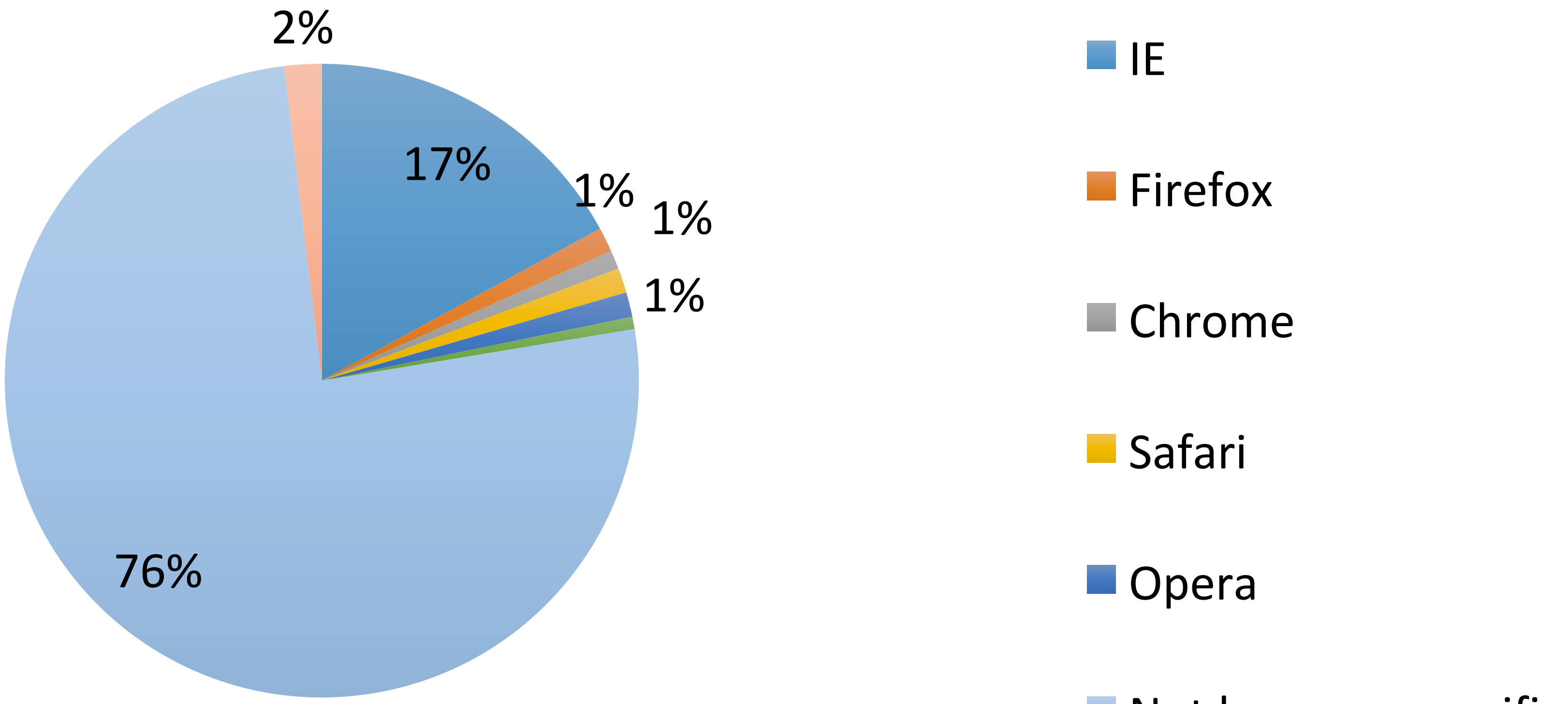
Bug Report Study: Fix Times

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



Bug Report Study: Browser Specificity

Most JavaScript faults are not browser-specific



Bug Report Study: Summary

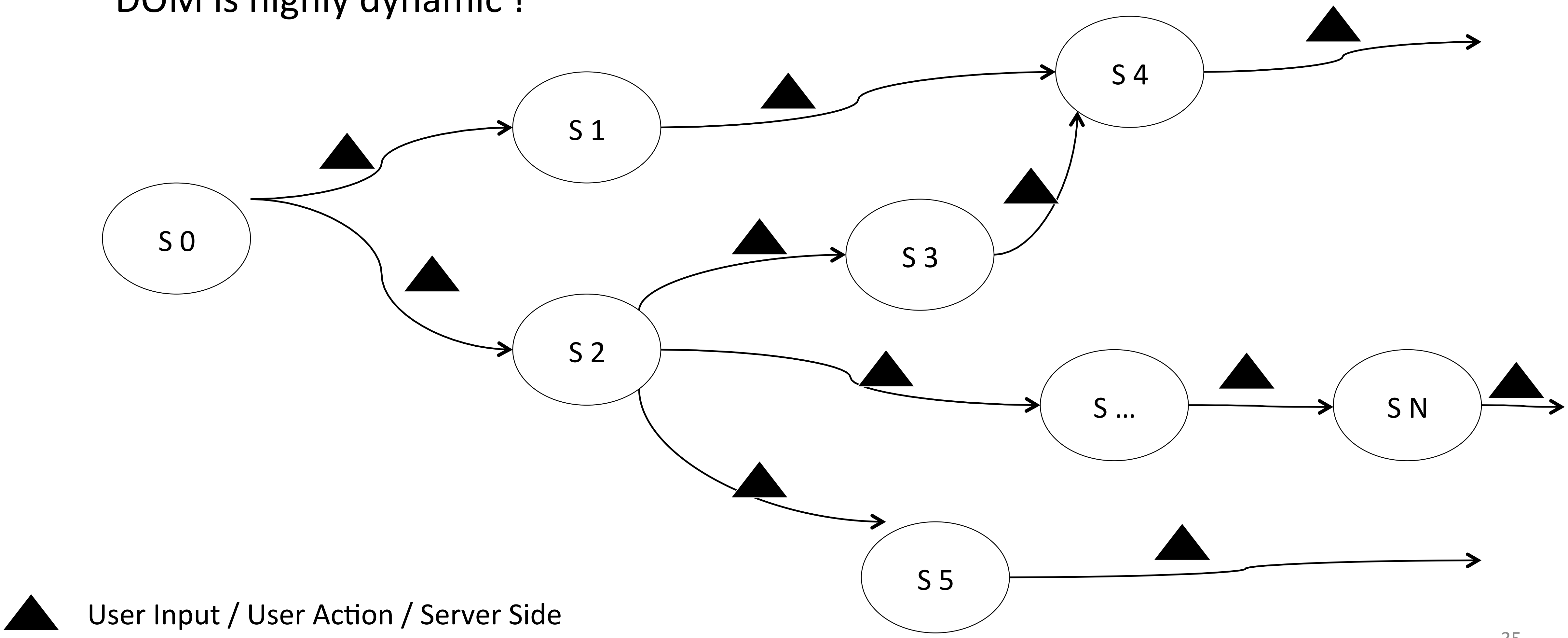
- **Bug report study of 12 applications: JS faults**
  - Over 300 bug reports analyzed; only fixed bugs considered
- **DOM-related faults dominate JavaScript faults**
  - Responsible for nearly two-thirds of all JavaScript faults
  - Responsible for 80% of highest impact faults
  - Take 50% longer time to fix for developers
  - Majority are not specific to web browser platform
- **Need robust solutions for DOM-related faults**
  - Fixing, Understanding and writing correct code

Web Applications: Existing Techniques

- **Add gradual typing to JavaScript (e.g., TypeScript from MS, DART from Google, Flow from Facebook)**
  - Typically ignore the DOM or provide only limited support
- **Use higher-level programming idioms in JavaScript**
  - MVC Frameworks (e.g., AngularJS)
  - Functional Reactive Programming (e.g., RxJS)
- **Detecting errors in web applications: Ignore DOM**
  - Race conditions [Vechev-OOPSLA'13][Livshits-FSE'15]
  - Type Coercion Errors [Pradel – ICSE'15]

Web Applications: Challenge

DOM is highly dynamic !



Web Applications: Our Approach

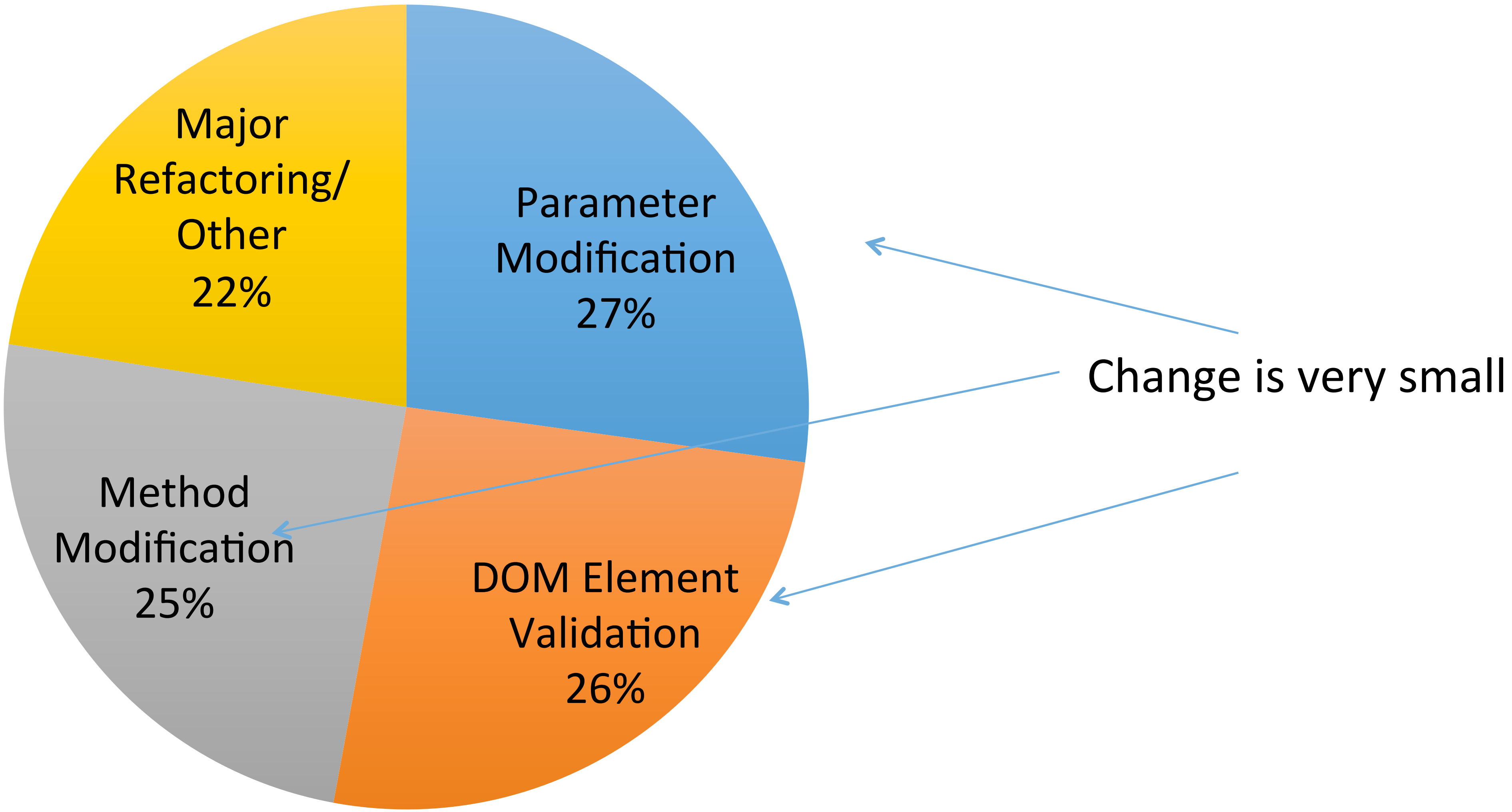
DOM-Related Faults [ESEM'13][TSE]

- **AutoFlox/Vejovis**: Localization and Repair [ICST'12] [ICSE'14A] [ICSE'15][STVR]
- **Clematis/ToChal/Shaand** : Program comprehension [ICSE'14B] [ECOOP'15] [ICSE'16][TOSEM]
- **DOMPletion/LED:** DOM Code completion and synthesis [ASE'14] [ASE'15]

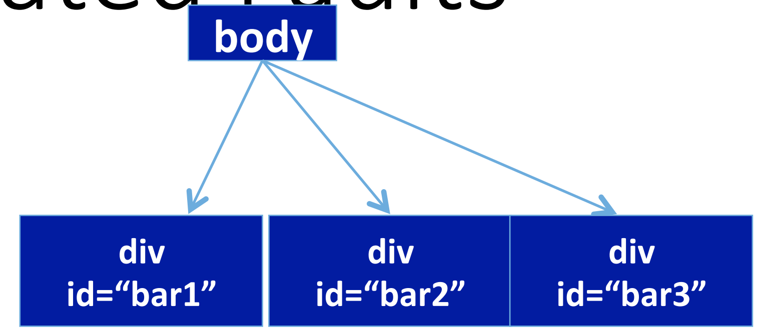
Talk Outline

- Motivation and Goals
- Empirical Study of reliability
- Reliability Improvements
- Program Understanding
- IDE Support
- Other Work and Future Directions

Web Applications: Bug Fix Patterns



Web Applications: DOM-Related Faults



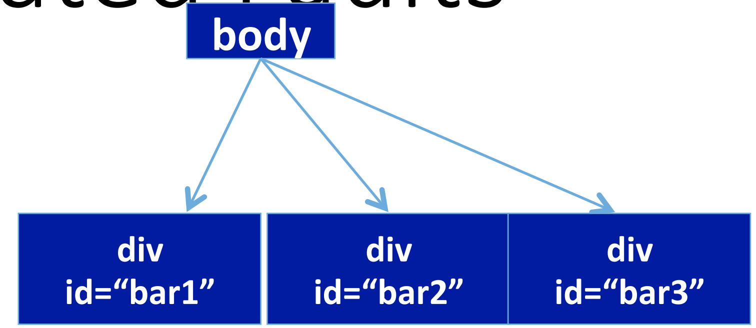

```
function generateId(index) {
        var prefix = "bar";
        var id = prefix + index;
        return id;
}

function retrieveElement(index) {
        var id = generateId(index);
        var e = document.getElementById(id);
        return e;
}

for (var i = 1; i <= 4; i++) {
        var elem = retrieveElement(i);
        elem.innerHTML = "Item #" + i;
}
```

Add the "bar" prefix to the ID

Retrieve the element with index i

Update retrieved element

Web Applications: DOM-Related Faults




```
function generateId(index) {
        var prefix = "bar";
        var id = prefix + index;
        return id;
}

function retrieveElement(index) {
        var id = generateId(index);
        var e = document.getElementById(id);
        return e;
}

for (var i = 1; i <= 4; i++) {
        var elem = retrieveElement(i);
        elem.innerHTML = "Item #" + i;
}
```

This should be “<“, not “<=“

Evaluates to “bar4” in 4th iteration

NULL EXCEPTION!

AutoFlox: Fault Localization

```
function generateId(index) {
        var prefix = "bar";
        var id = prefix + index;
        return id;
}

function retrieveElement(index) {
        var id = generateId(index);
        var e = document.getElementById(id)
        return e;
}

for (var i = 1; i <= 4; i++) {
        var elem = retrieveElement(i);
        elem.innerHTML = "Item #" + i;
}
```

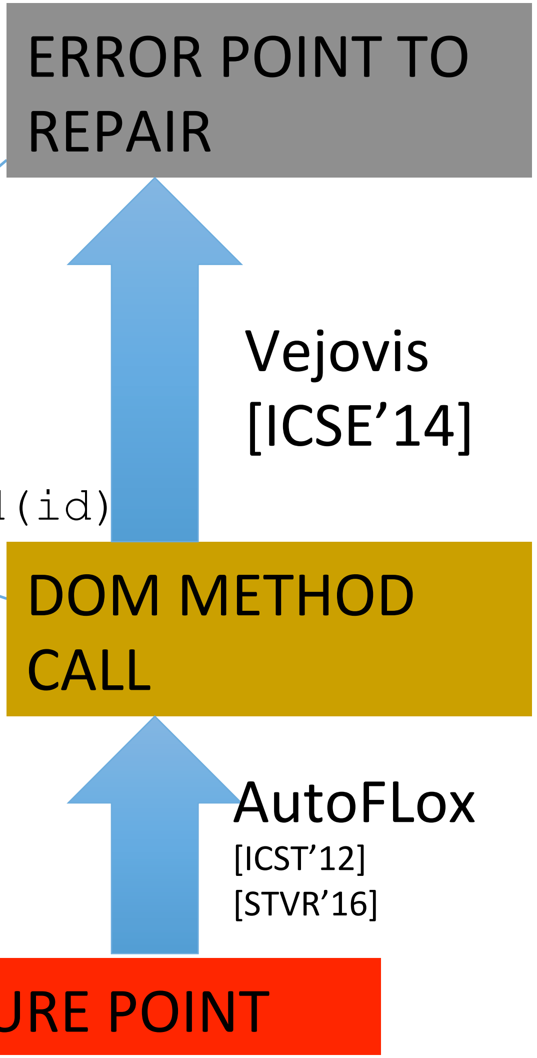
ERROR POINT TO REPAIR

Our Goal

FAILURE POINT

AutoFlox: Fault Localization

```
function generateId(index) {
        var prefix = "bar";
        var id = prefix + index;
        return id;
}

function retrieveElement(index) {
        var id = generateId(index);
        var e = document.getElementById(id)
        return e;
}

for (var i = 1; i <= 4; i++) {
        var elem = retrieveElement(i);
        elem.innerHTML = "Item #" + i;
}
```



Vejovis: Fault Repair

```
function generateId(index) {
        var prefix = "bar";


}

f



}

for (var i = 1; i <= 4; i++) {
        var elem = retrieveElement(i);
        elem.innerHTML = "Item #" + i;
}
```

“bar”

, and

ady

in

ions

“OFF-BY-ONE, SO REMOVE LAST ITERATION OF FOR LOOP”

Evaluates to “bar4”

“4”

“4” comes from iterator i

AutoFlox and Vejovis: Results-1

Real-world JS faults

20 bugs analyzed by AutoFLox, and 22 bugs by Vejovis (from applications with **100 LOC to 11000 LOC**)

RESULT 1: AutoFLox successfully localized *100%* of the real-world faults

RESULT 2: Vejovis successfully found repair for 91% of bugs

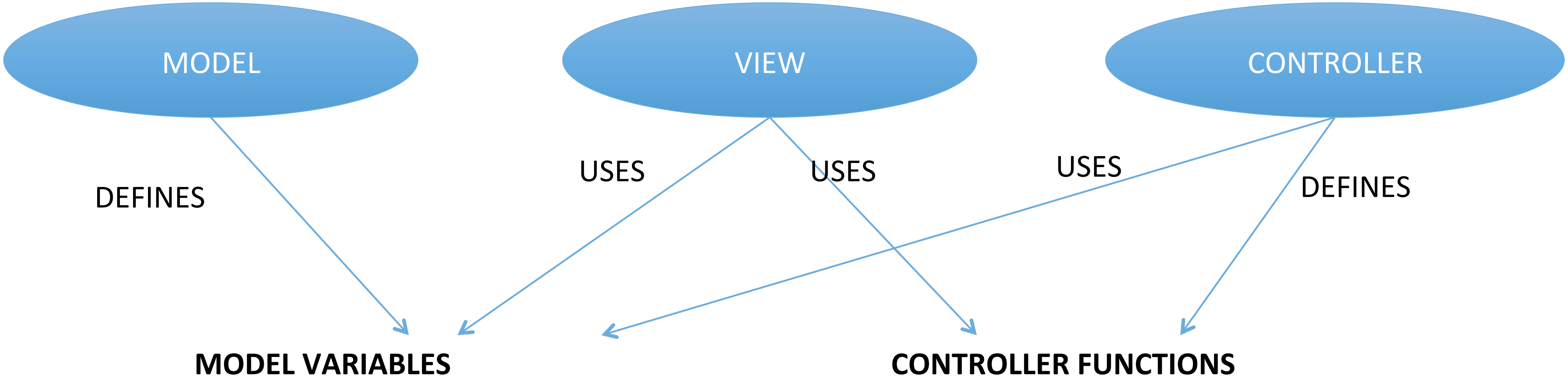
AutoFlox and Vejovis: Results-2

Performance

6 web applications for AutoFLox and 11 web applications for Vejovis, ranging from **100 LOC to 11000 LOC**

RESULT: Both tools execute 1 min. on average (worst case 90 seconds)

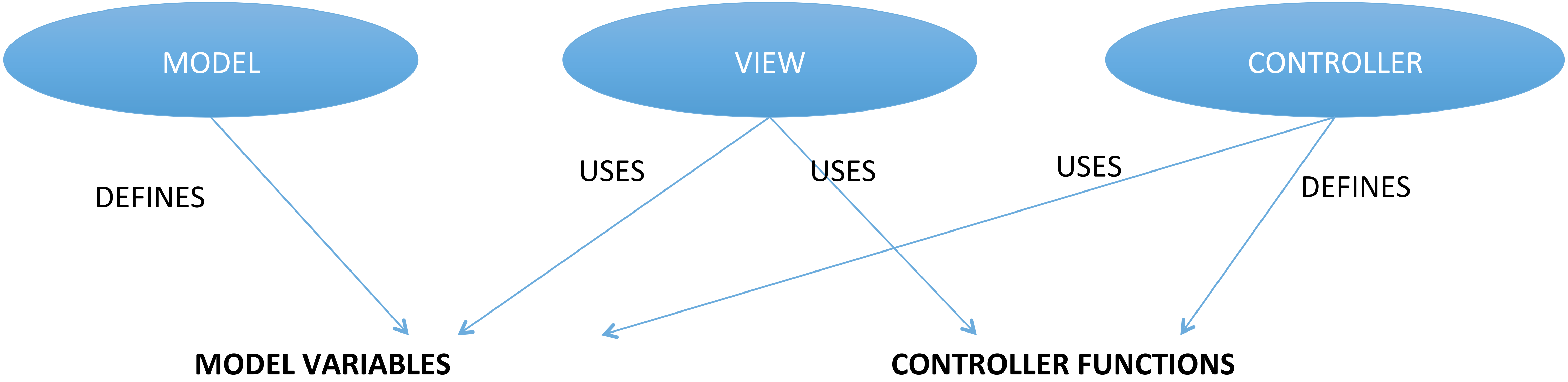
Automatic Fault Detection: Background



MVC Frameworks

DOM abstracted out → DOM-related faults not problematic!

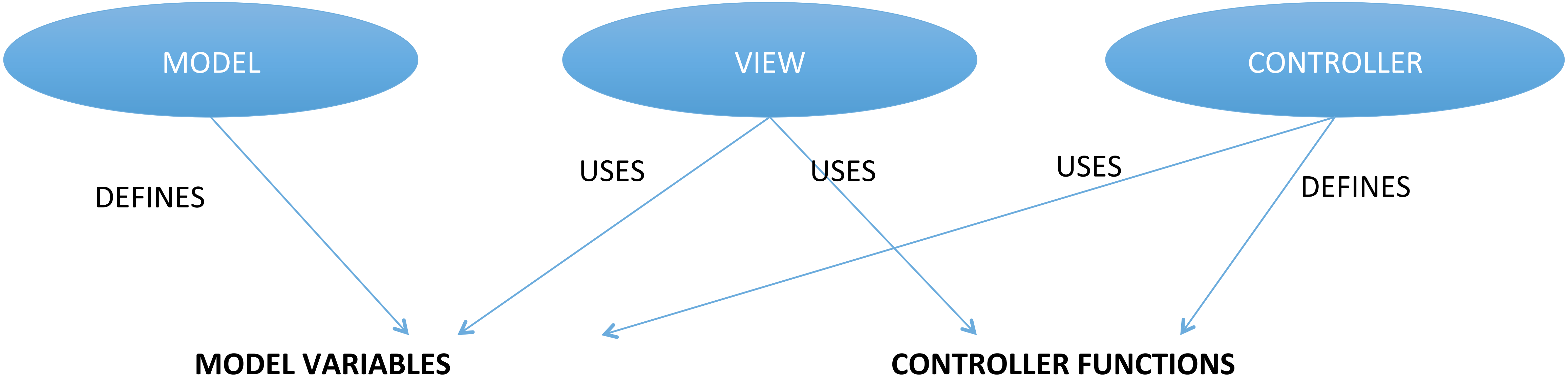
Automatic Fault Detection: Background



Problem: Inconsistencies between identifiers and types in model, view, and controller

Automatic Fault Detection: Background

MVC Frameworks

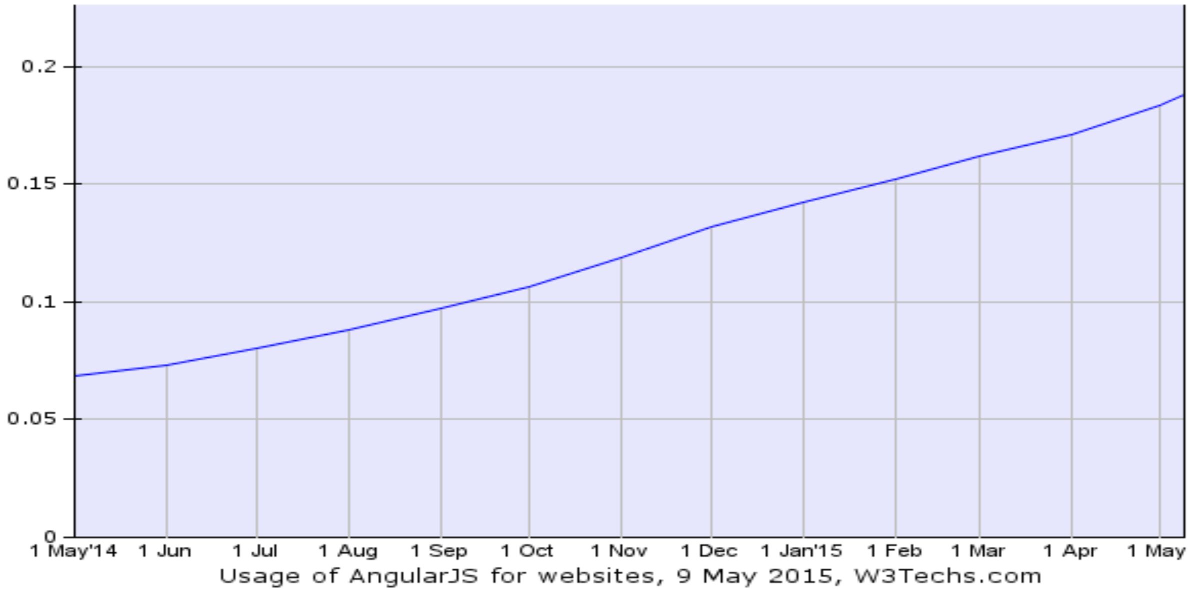


Our Solution: Approach to automatically detect identifier and type inconsistencies

ANGULARJS

Aurebesh: AngularJS

The most popular JS MVC framework in GitHub, StackOverfow, and even YouTube!



300% increase in AngularJS usage in the year 2015

Automatic Fault Detection: Methodology

MODEL

String: a
Boolean: b
Object: c

VIEW

Boolean: e
String: bar()

VIEW

String: d
String: fun()

CONTROLLER

Object: c
Number: foo()

MODEL

String: d

VIEW

String: a
Boolean: e
String: foo()

CONTROLLER

String: bar()

MODEL

Boolean: e

CONTROLLER

String: fun()

Automatic Fault Detection: Methodology

MODEL

String: a
Boolean: b
Object: c

VIEW

Boolean: e
String: bar()

VIEW

String: d
String: fun()

CONTROLLER

Object: c
Number: foo()

MODEL

String: d

VIEW

String: a
Boolean: e
String: foo()

CONTROLLER

String: bar()

MODEL

Boolean: e

“e” is not defined in model!

CONTROLLER

String: fun()

Inconsistent types!

Automatic Fault Detection: Results

TOOL: Aurebesh

Fault injection experiment

Real-world web applications

Aurebesh is 96.1% accurate, with only one false positive

Aurebesh detected 15 *previously undetected* bugs (5 were acknowledged by developers)

http://www.ece.ubc.ca/~frolino/projects/aurebesh

Aurebesh: Screenshot

INDEX.HTML (VIEW)

```
    <pane title="Pluralization">
     <div ng-controller="BeerCounter">
      <div ng-repeat="beerCount in beets">
       <ng-pluralize count="beerCount" when="be
      </div>
     </div>
    </pane>
```

"beets" is undefined!

"beets"

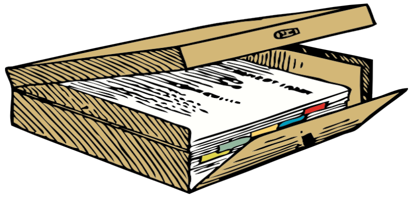
APP.JS (MODEL)

```
    .controller('BeerCounter', function($scope) {
        $scope.beers = [0, 1, 2, 3, 4, 5, 6];
        $scope.beerForms = {
            0: 'no beers',
            one: '{} beer',
            other: '{} beers'
        };
    });
```

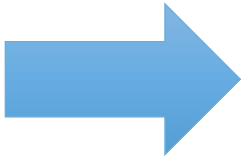
"beers"

"beerForms"

Drawback of Aurebesh



Analyzed 90 MVC bug reports



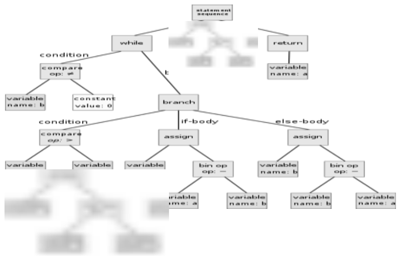
Over **40** inconsistency categories!

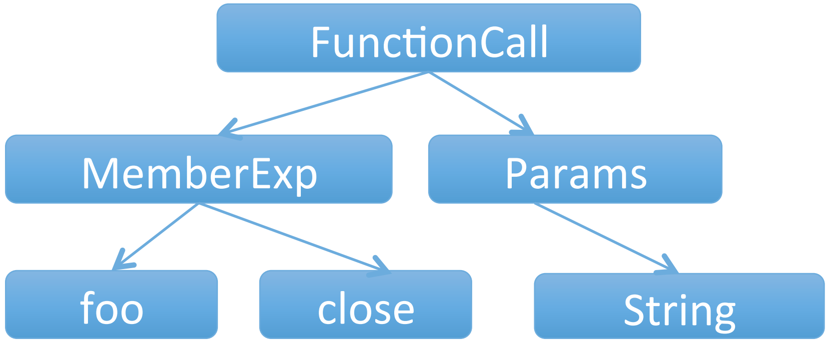
Aurebesh only supports 4 inconsistency categories!

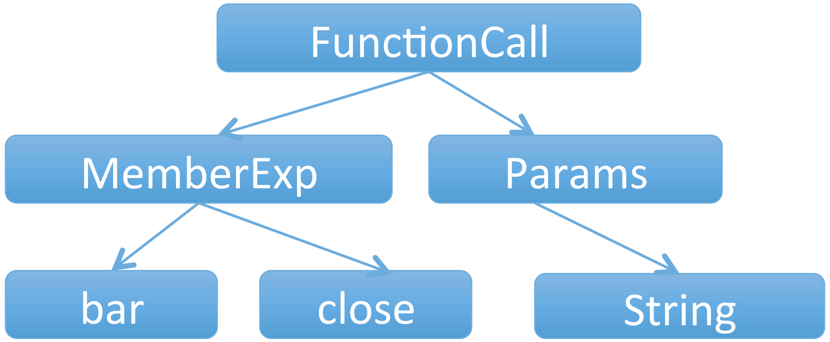
A Generalized Inconsistency Detector

Step 1: Infer Implicit Consistency Rules

From target application:



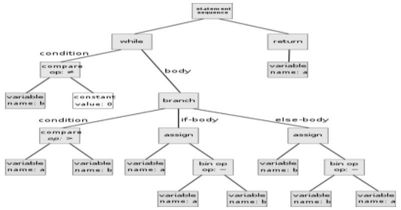


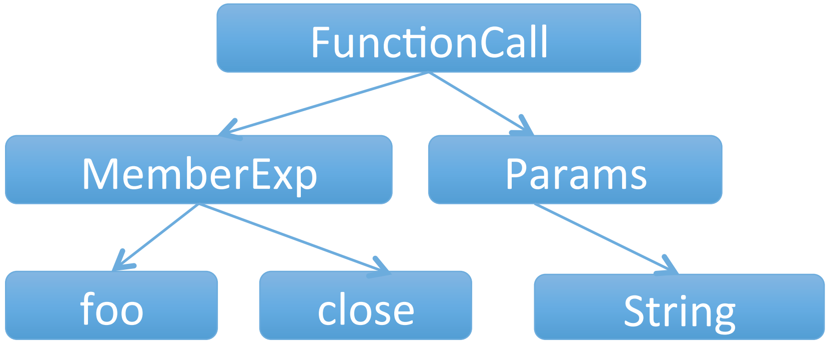


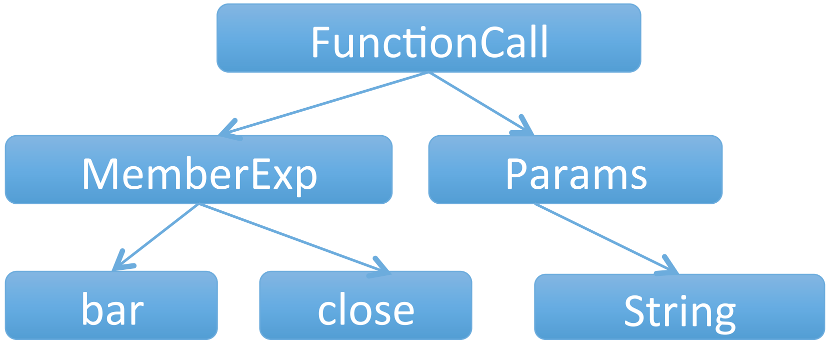
A Generalized Inconsistency Detector

Step 1: Infer Implicit Consistency Rules

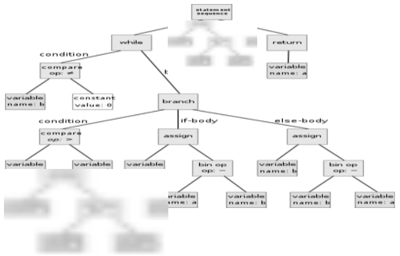
From target application:

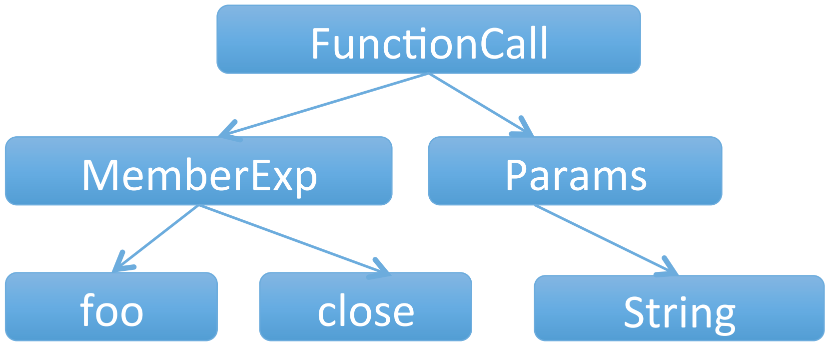


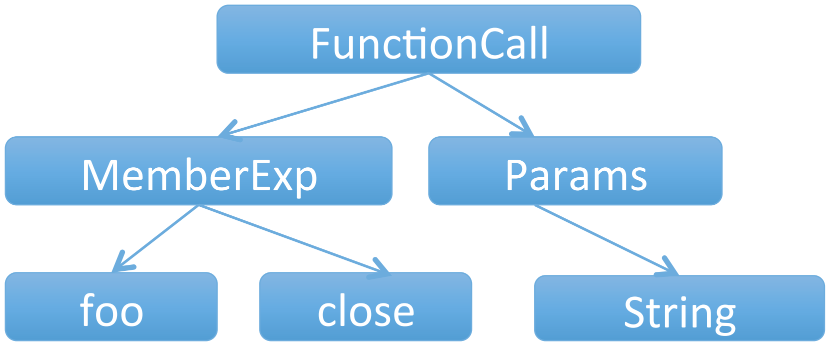




From sample applications:



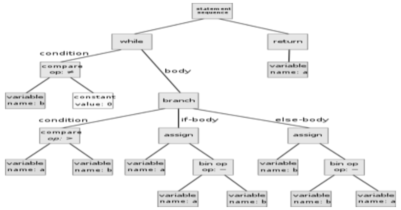


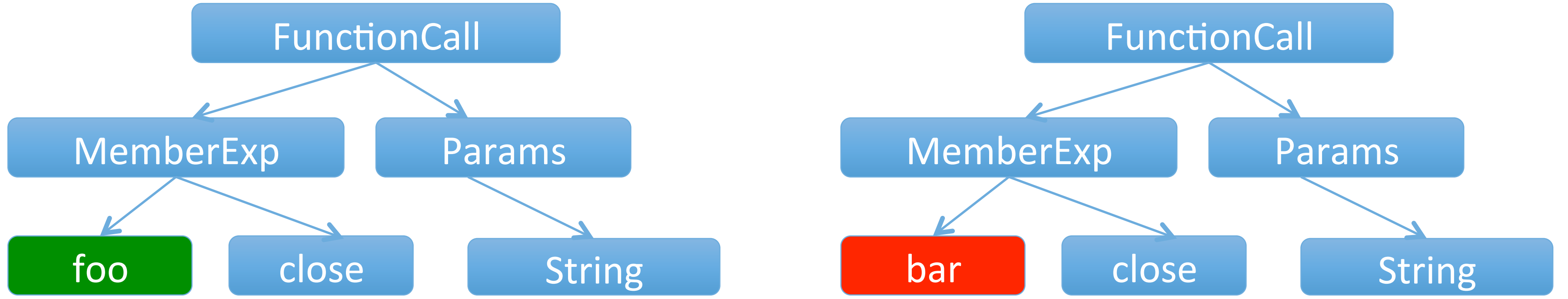


A Generalized Inconsistency Detector

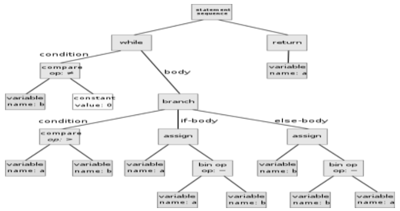
Step 2: Detect Rule Violations

From target application:



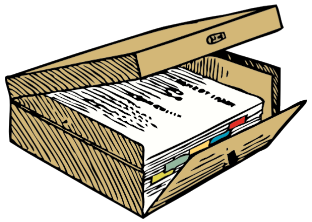


From sample applications:

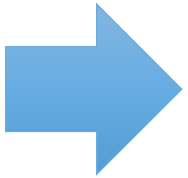


HoloCron: Generalized Inconsistency Detector

Learns inconsistency patterns for any MVC-like framework



Analyzed 90 MVC bug reports



35% of inconsistencies are cross-language

Generalized Detector: Results

Real-world web applications

Holocron detected

18 *previously undetected* bugs

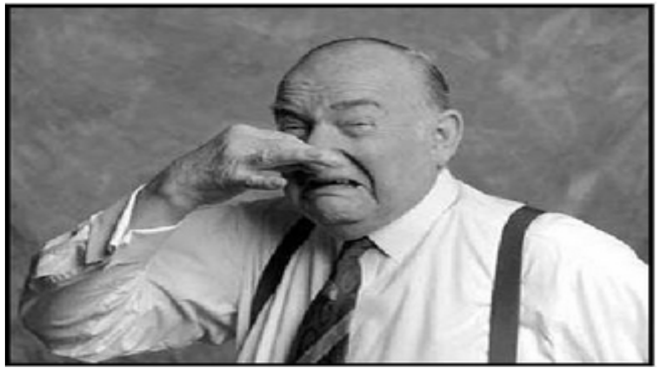
from MVC applications

15 inconsistency categories

5 cross language inconsistencies

Generalized Detector: Results

TOOL: Holocron



Code Smell = not a bug, but makes code more difficult to maintain

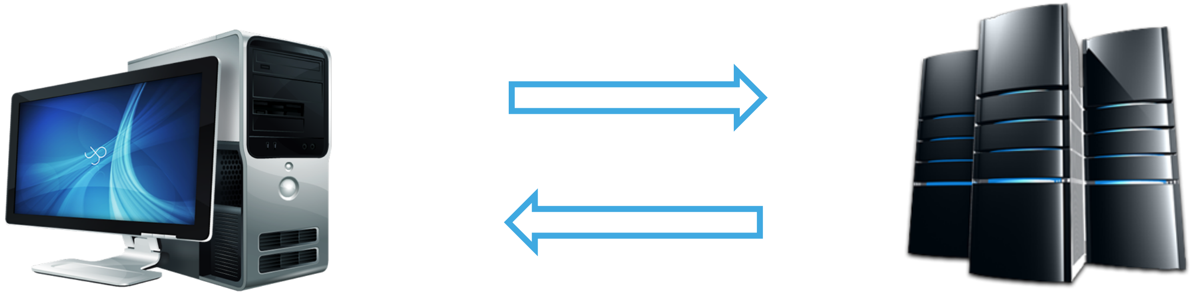
Example: Giving the same name to unrelated variables

1 out of every 2 reports are either real bugs or code smells

Talk Outline

- Motivation and Goals
- Empirical Study of reliability
- Reliability Improvements
- Program Understanding
- IDE Support
- Other Work and Future Directions

Understanding JavaScript Apps



Event-based Interactions

Client/Server Interactions

Asynchronous Server Interactions

JS and DOM Interactions

Clematis [ICSE'14]

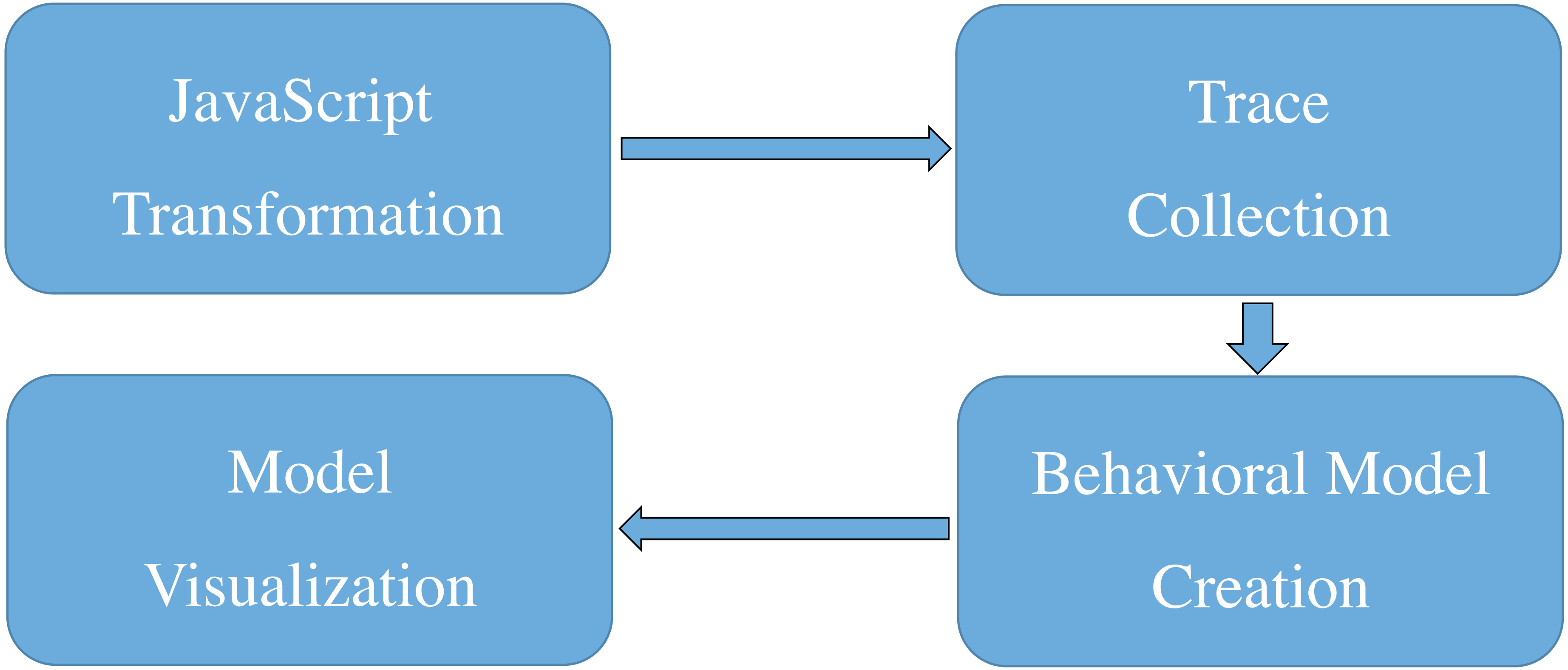
Tochal [ECOOP'15]

Sahand [ICSE'16]

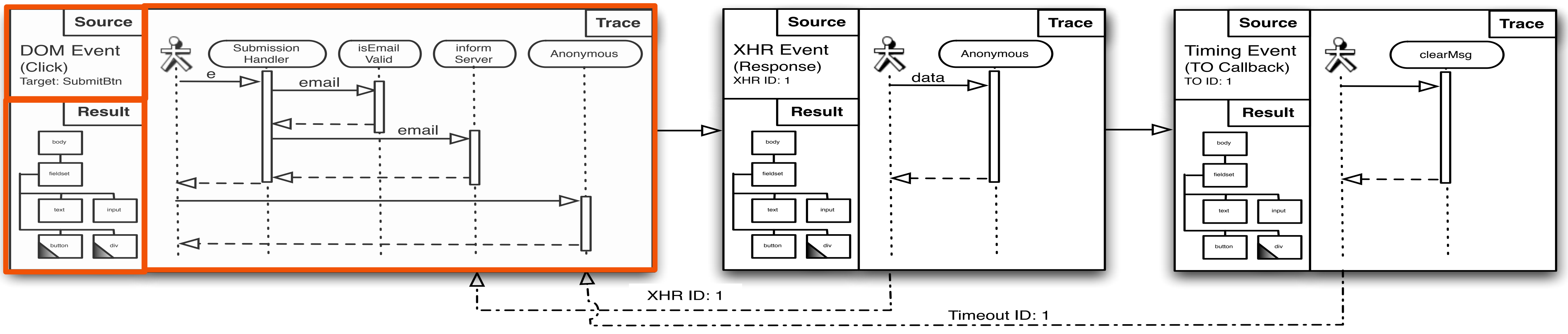
Clematis: Motivation

- **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

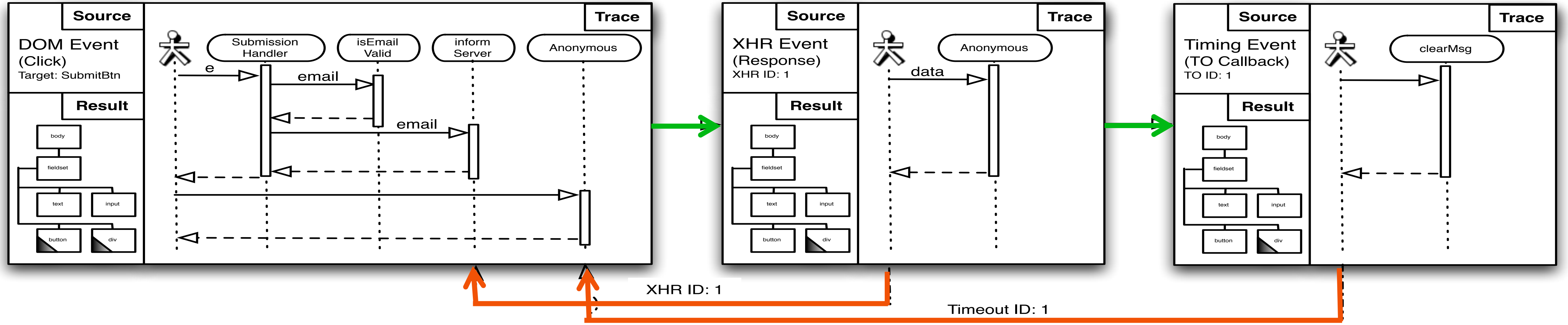
Clematis: Approach



Clematis: Model Episodes



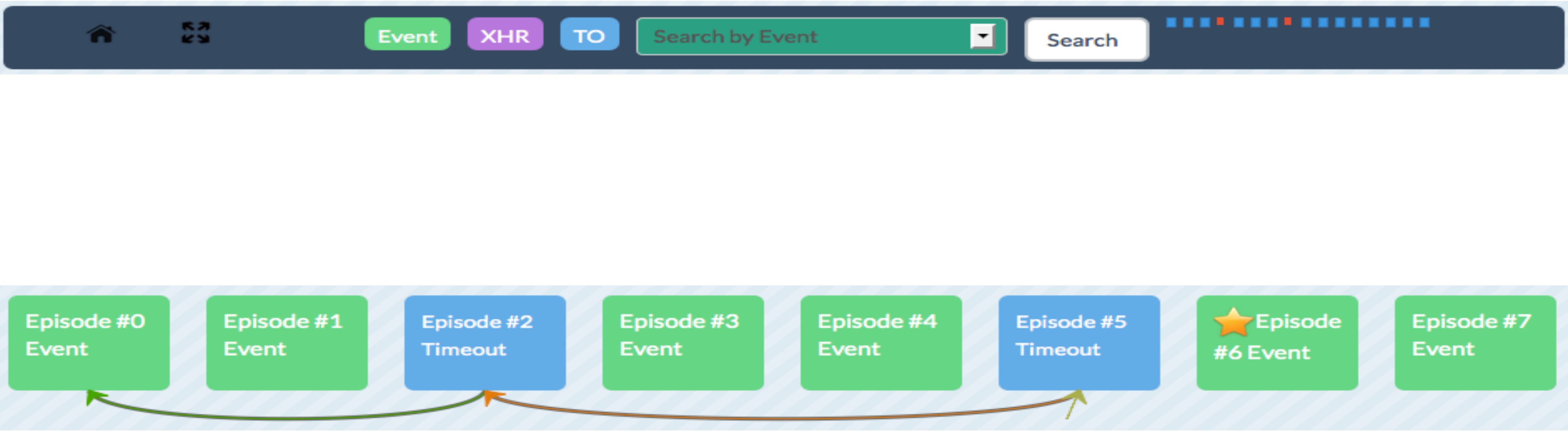
Clematis: Model Links



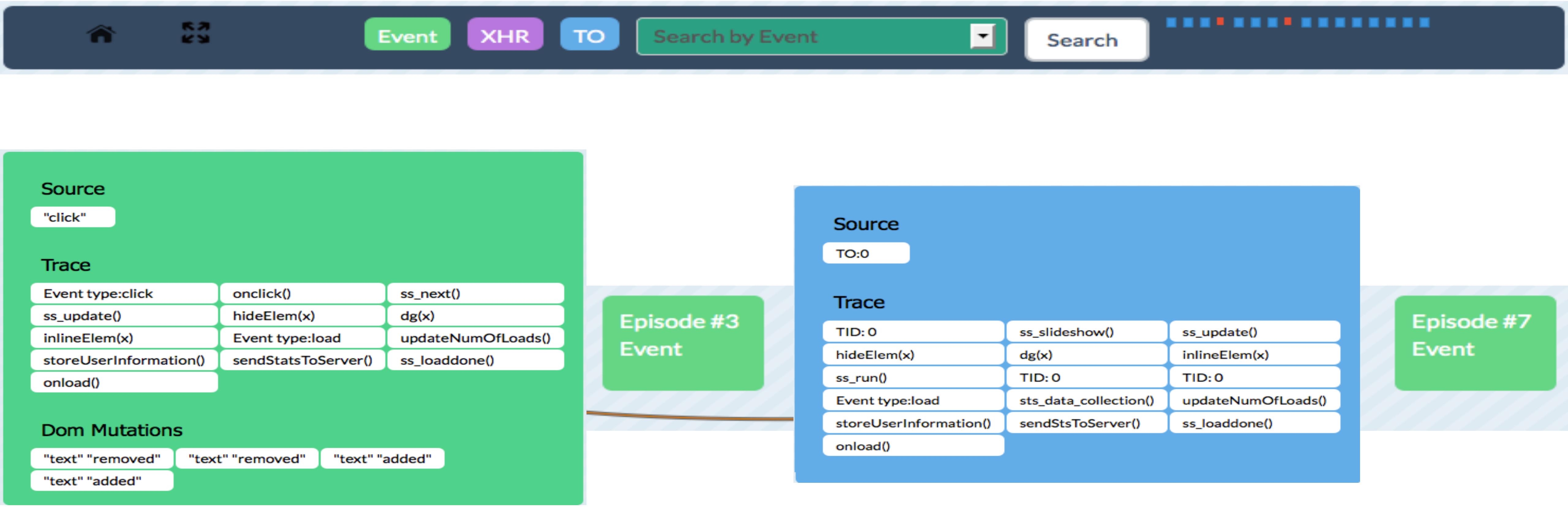
Temporal

Causal

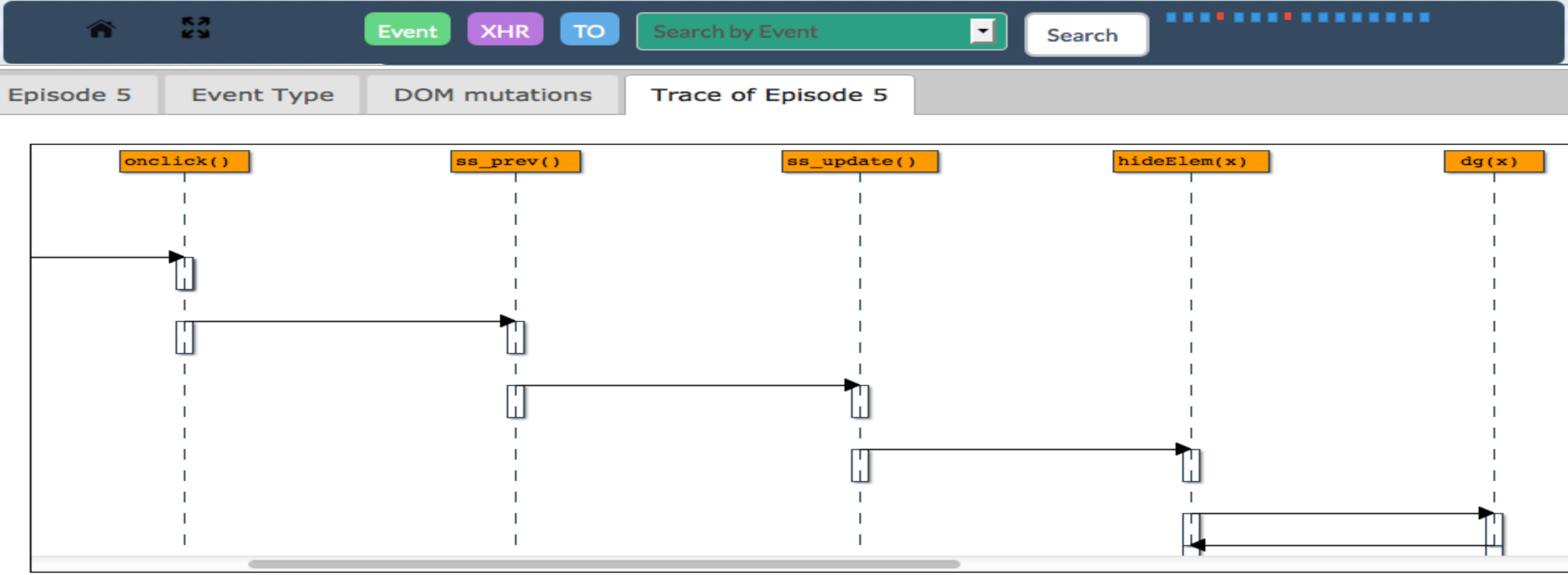
Clematis: Visualization



CLEMATIS: VISUALIZATION



Zoom Level 1



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

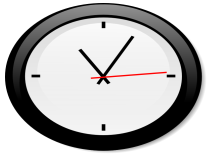
Clematis: User Experiment

- 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

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

Accuracy

Task	Improvement
T1	(67%↑)
T2	(41%↑)
T3	(20%↑)
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

ToChal: Change Impact Analysis (CIA)

- Software must continually change to adapt to the changing environment.
- **Goal**: identifying parts of the program that are potentially affected by a change.

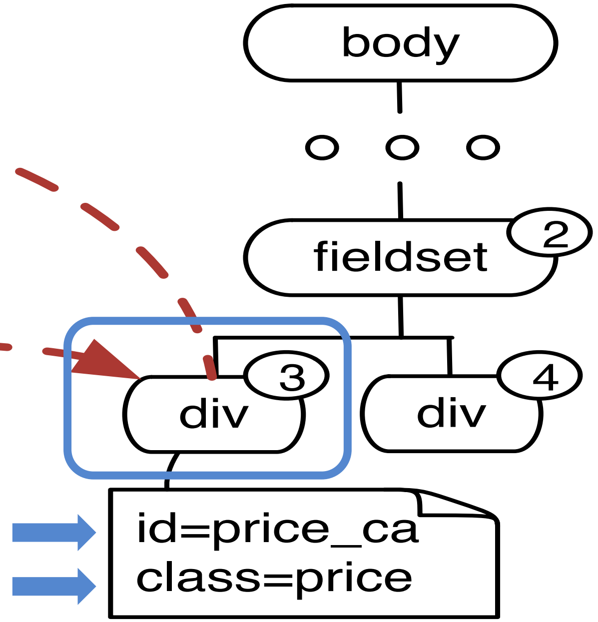
- Hybrid of static and dynamic analyses

ToChal: Impact through the DOM

```
function checkPrice() {
    . . .
    var cad-price = $('#price_ca').innerText();
    . . .
}
```

```
function calculateTax() {
    $('.price').each(function(index) {
        $(this).text(addTaxToPrice(
                $(this).text());
    });
}
```

```
$('#price_ca').bind('click', checkPrice);
```



ToChal: Approach

- Static control-flow and data-flow analysis

- Analyzing the dynamic features
  - Dynamic call graph
  - DOM interactions
  - Event-based impact propagation
  - XHR relations

- Hybrid model for impact analysis

Tochal: Tool Implementation

- Tochal: open source
  - https://github.com/saltlab/tochal

- Proxy (Java, JavaScript)
  - Esprima, Estraverse, Escodegen, Mutation Summary, WALA

- Client-side (Google Chrome extension)
  - Chrome DevTools

ToChal: User Experiment

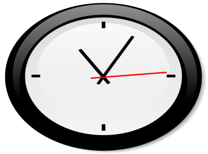
- Question: Does Tochal help developers in practice to perform change impact analysis?

- Design:
  - 12 participants from industry
  - 4 tasks: detecting and analyzing change impact
  - Measured: task completion duration and accuracy

ToChal: User Experiment Results

Accuracy

Task	Improvement
Total	223%↑↑

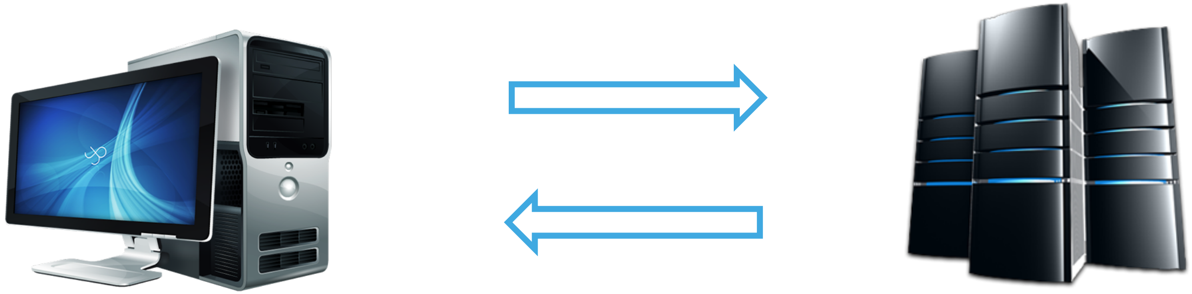


Duration

Task	Improvement
T1	78%↑↑

Task	Description
T1	Finding the potential impact of a DOM element
T2	Finding the potential impact of a JavaScript function
T3	Finding a conflict after making a new change (no ranking)
T4	Finding a bug in JavaScript code

Understanding JavaScript Apps



Event-based Interactions

Client/Server Interactions

Asynchronous Server Interactions

JS and DOM Interactions

Clematis [ICSE'14]

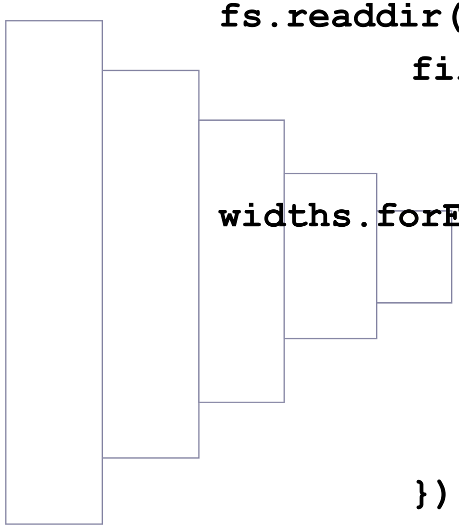
Tochal [ECOOP'15]

Sahand [ICSE'16]

Node.js Challenges

- Asynchronous execution
- Network communication
- Scalability
  - Example: Callback hell

Little pyramid of doom

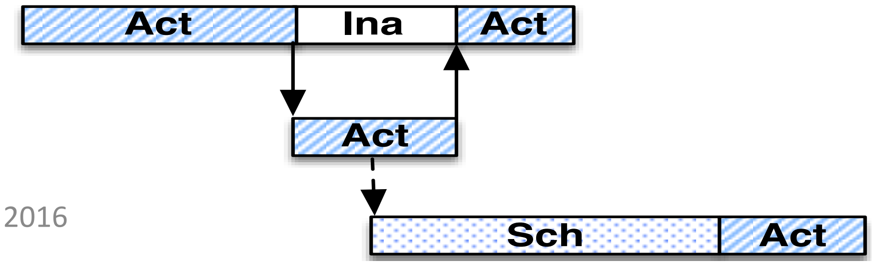


```
fs.readdir(source, function(err, files) {
        files.forEach(function(filename, fileIndex)
                gm(source + filename).size(function(err, values) {
widths.forEach(function(width, widthIndex) {
                                    this.resize(w, h).write(newName, function(err)
                                    })
                                })
                        })
                })
})   // example from callbackhell.com
```

Our Approach: ***Sahand***

Application →

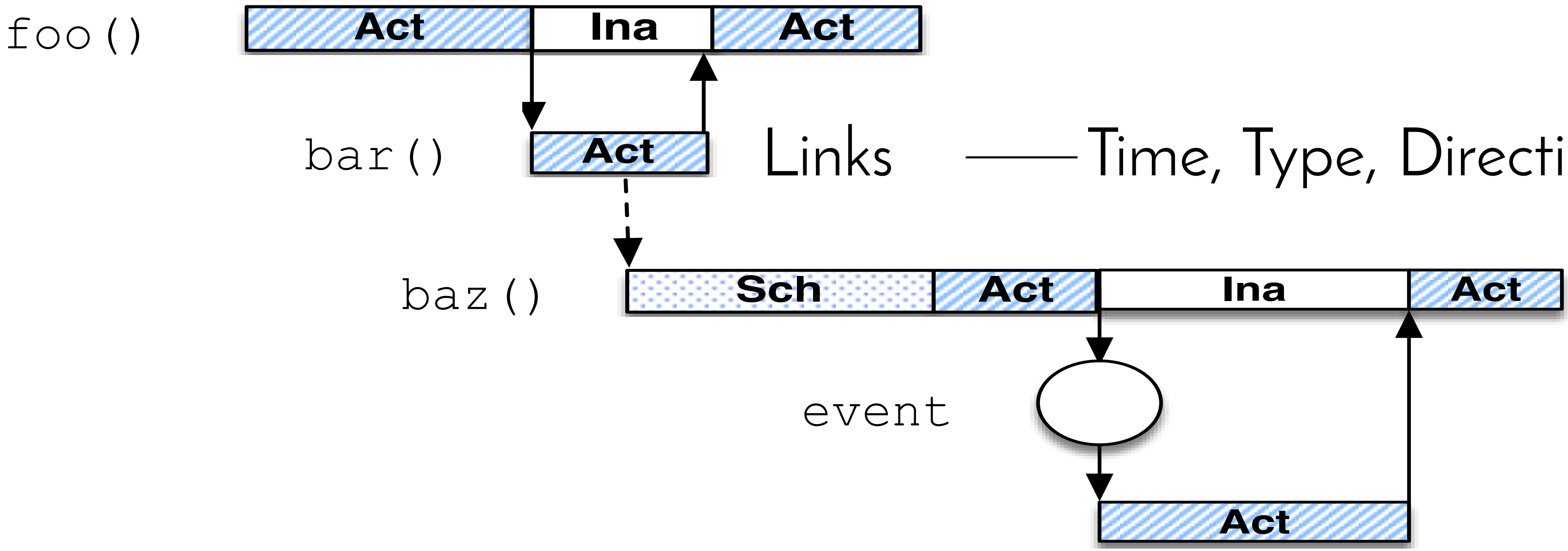
1. Instrument automatically
2. Trace full-stack execution
3. Infer a behavioural model
4. Visualize the model



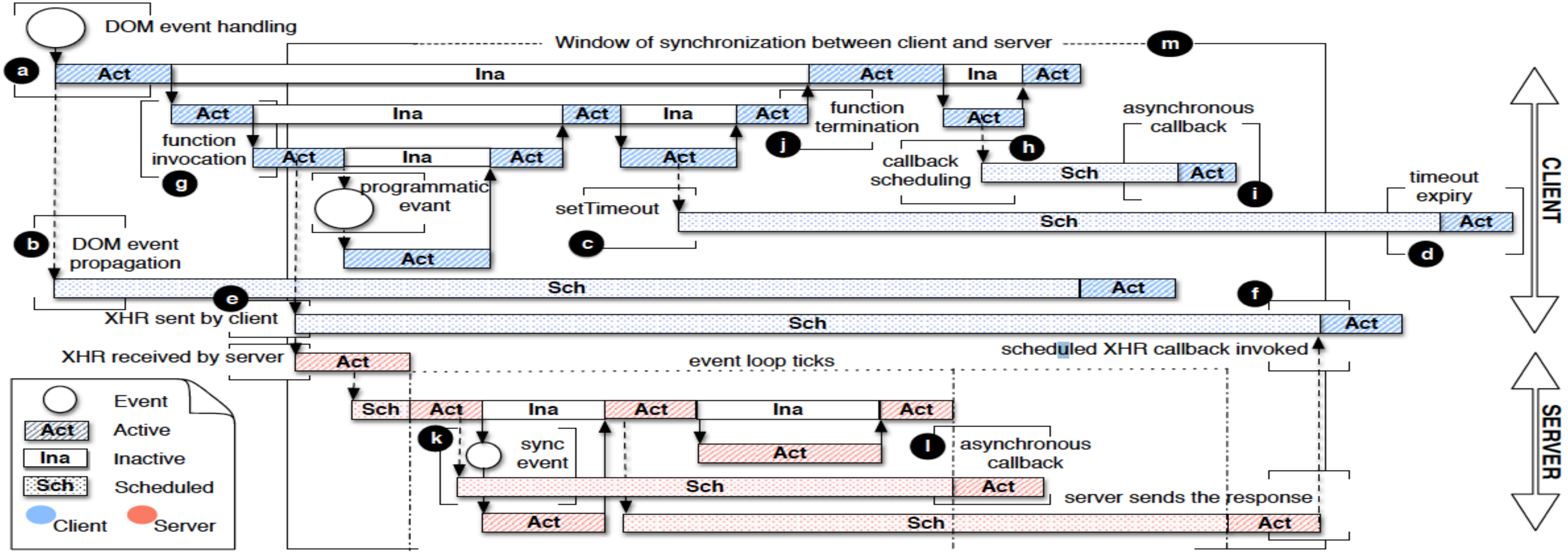
Behavioral Model

Nodes
- Lifelines of function executions
- (A)Synchronous client/server events

Links — Time, Type, Direction



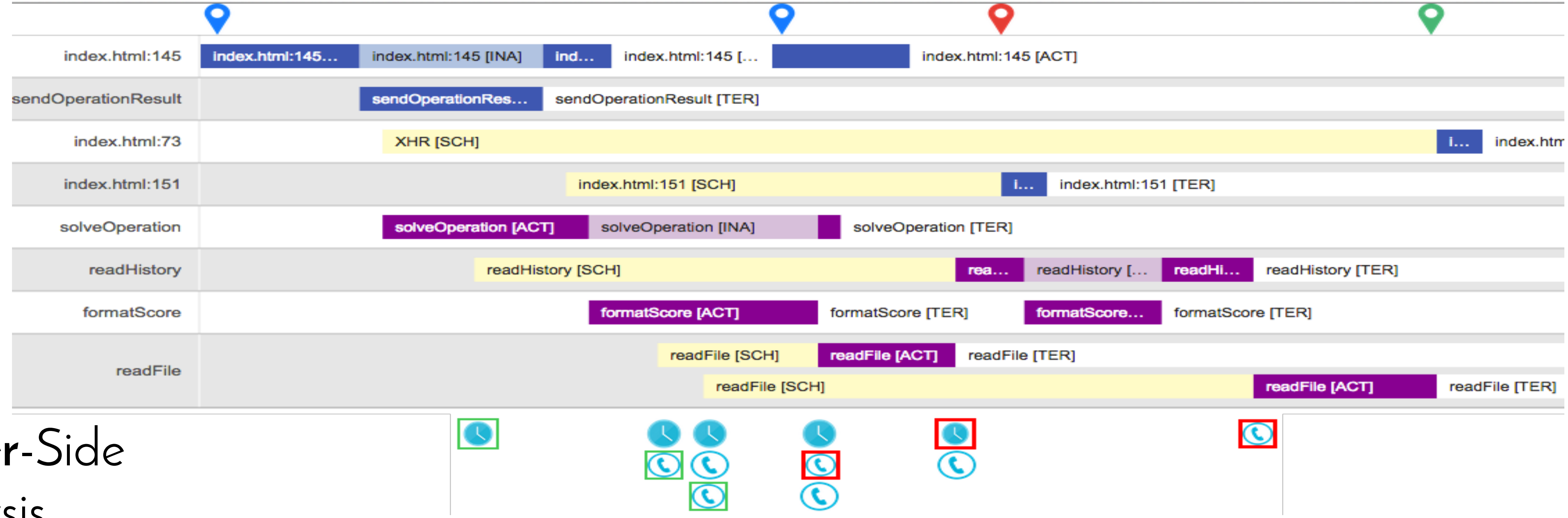
Real Behavioural Models Are Complex



Visualization

Client-Side Analysis

https://github.com/saltlab/sahand



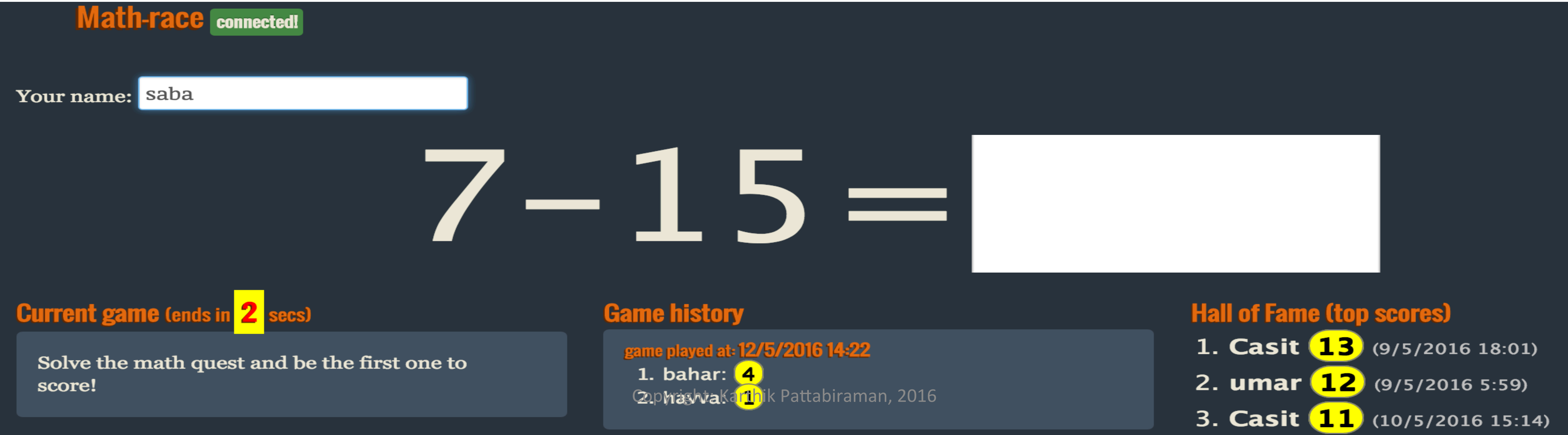
Server-Side Analysis

Evaluation

Does using ***Sahand*** improve developers' performance in program comprehension tasks?

Controlled Experiment

- ***Sahand***'s effect on developers' performance
- 12 Participants
- Object: full-stack JavaScript application

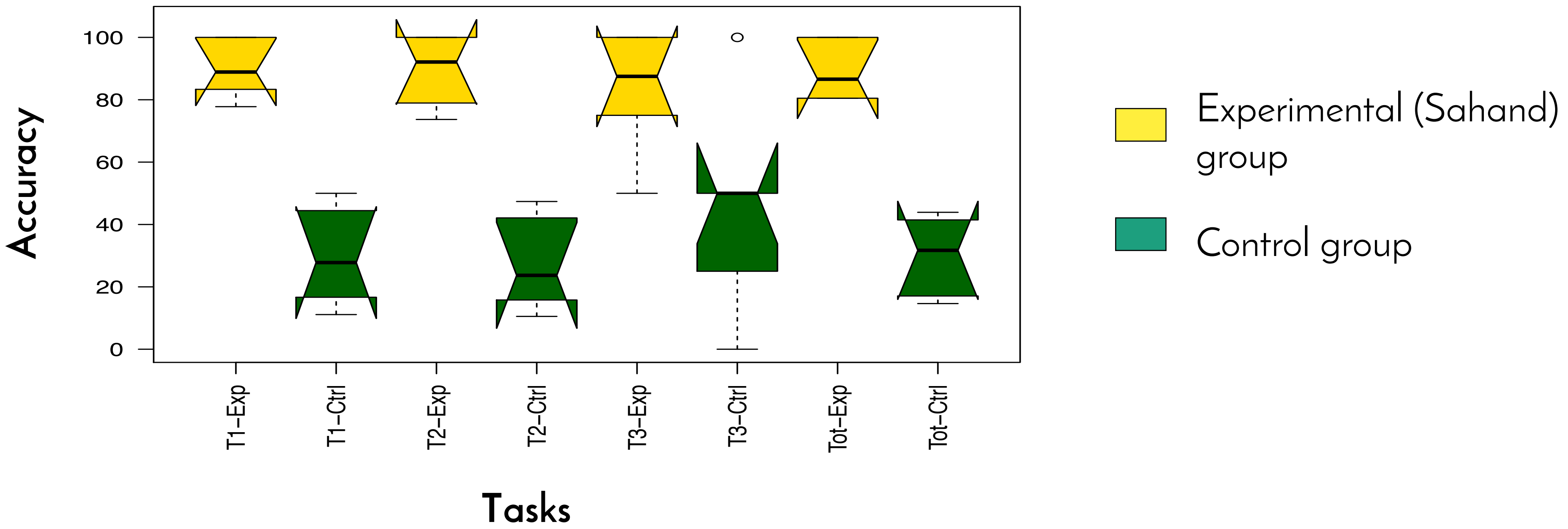


Results Highlight

Using ***Sahand***

3 times more accuracy

In the same time

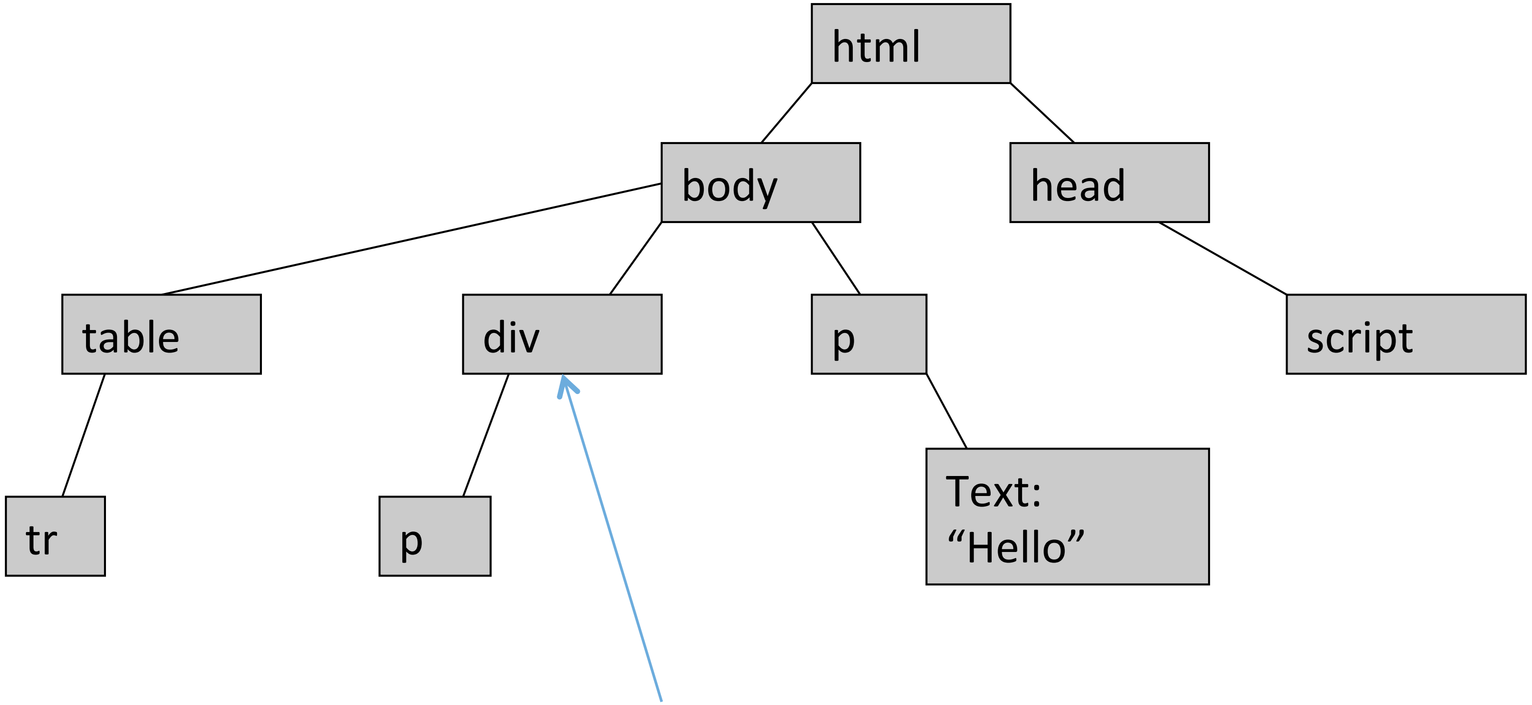


Talk Outline

- Motivation and Goals
- Empirical Study of reliability
- Reliability Improvements
- Program Understanding
- IDE Support
- Other Work and Future Directions

Dompletion: Motivation

- Provide code-completion for DOM-JavaScript interactions

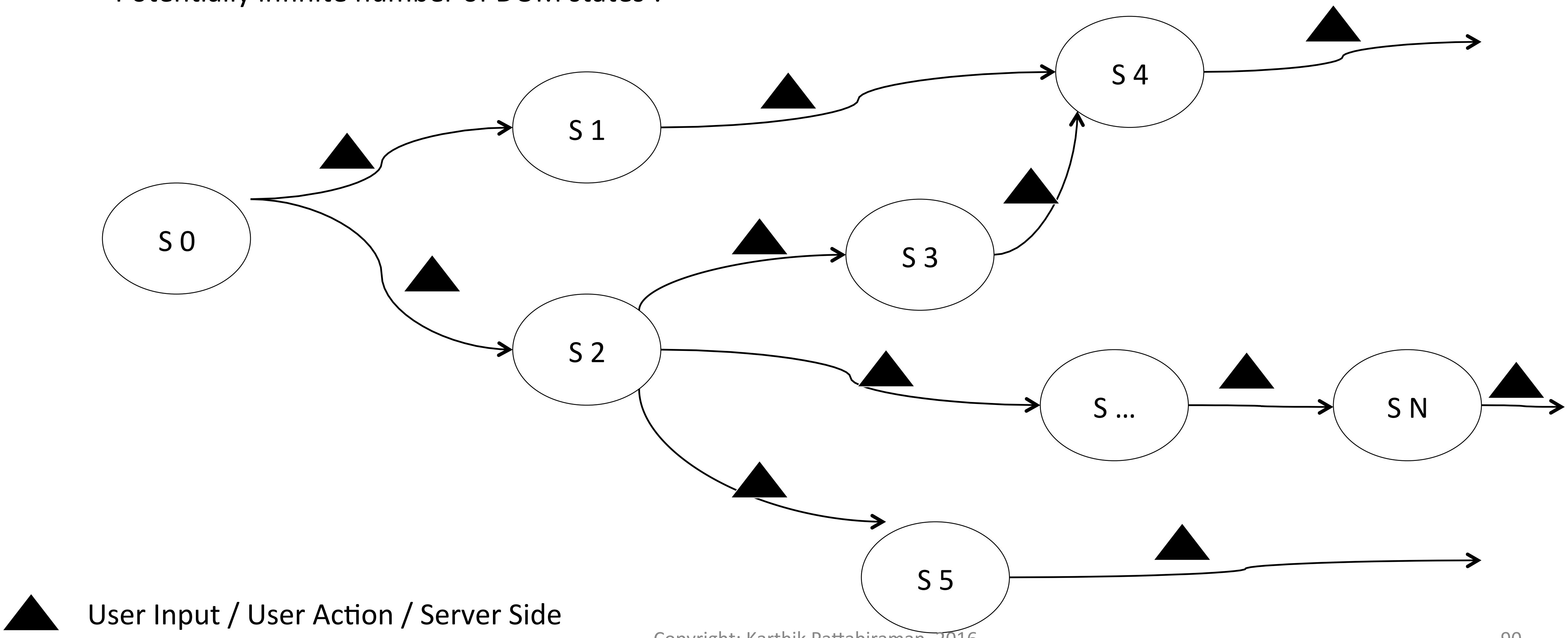


Want to retrieve element with id “elem”

```
var x = document.getElementById(“elem”);
```

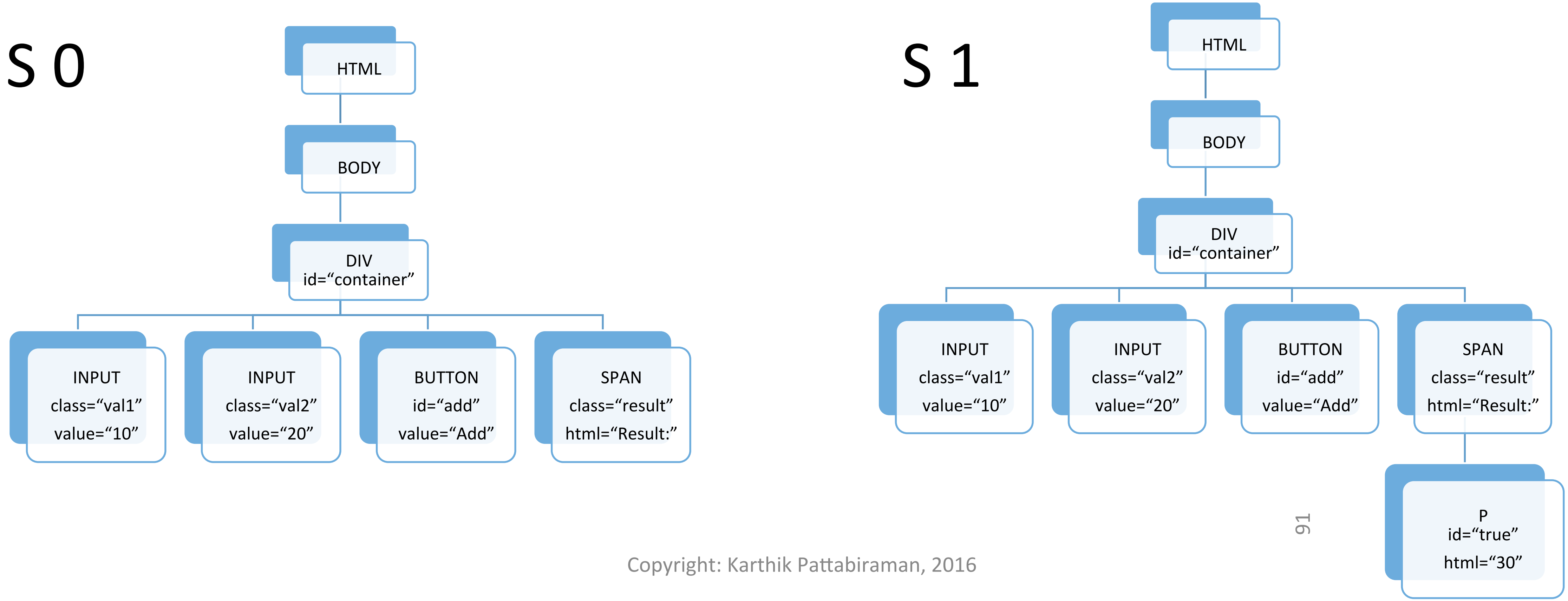
Dompletion: Challenge

Potentially infinite number of DOM states !

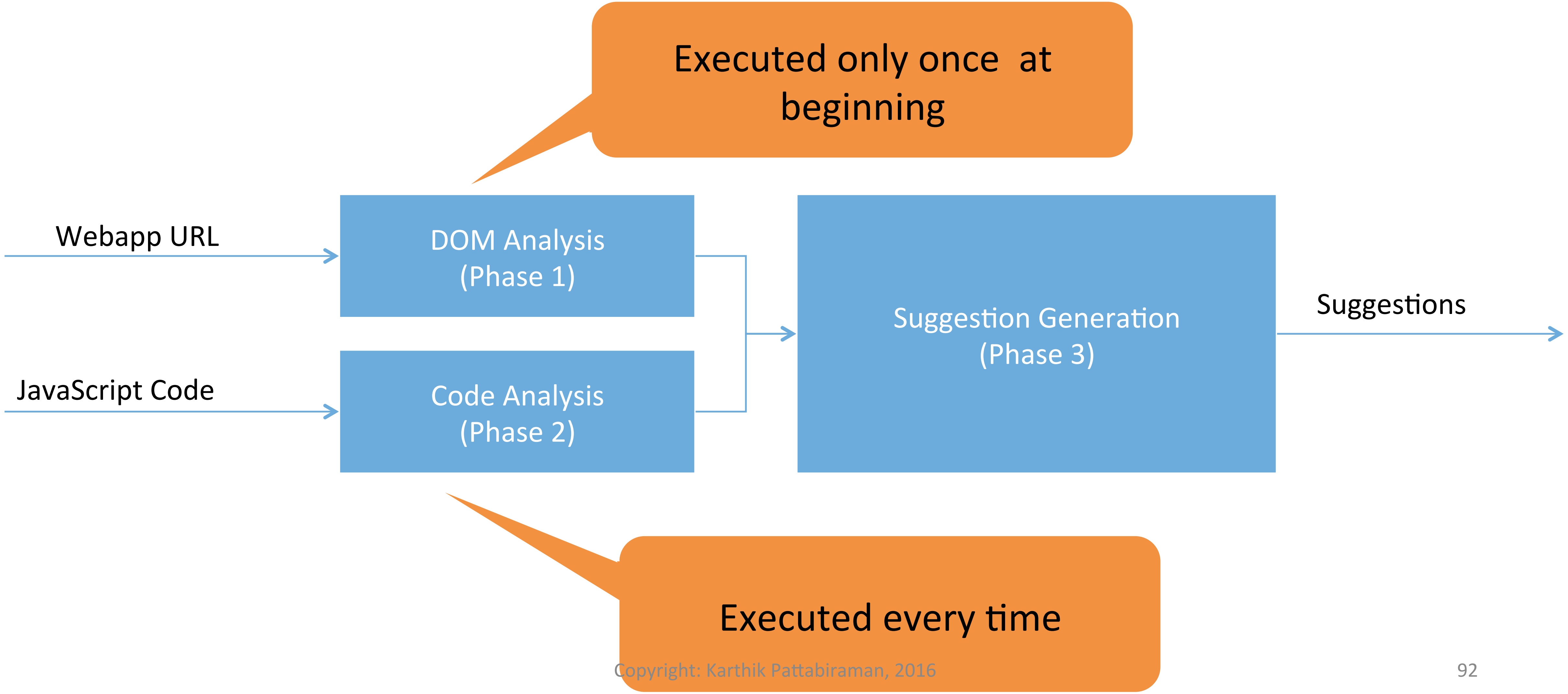


Dompletion: Intuition

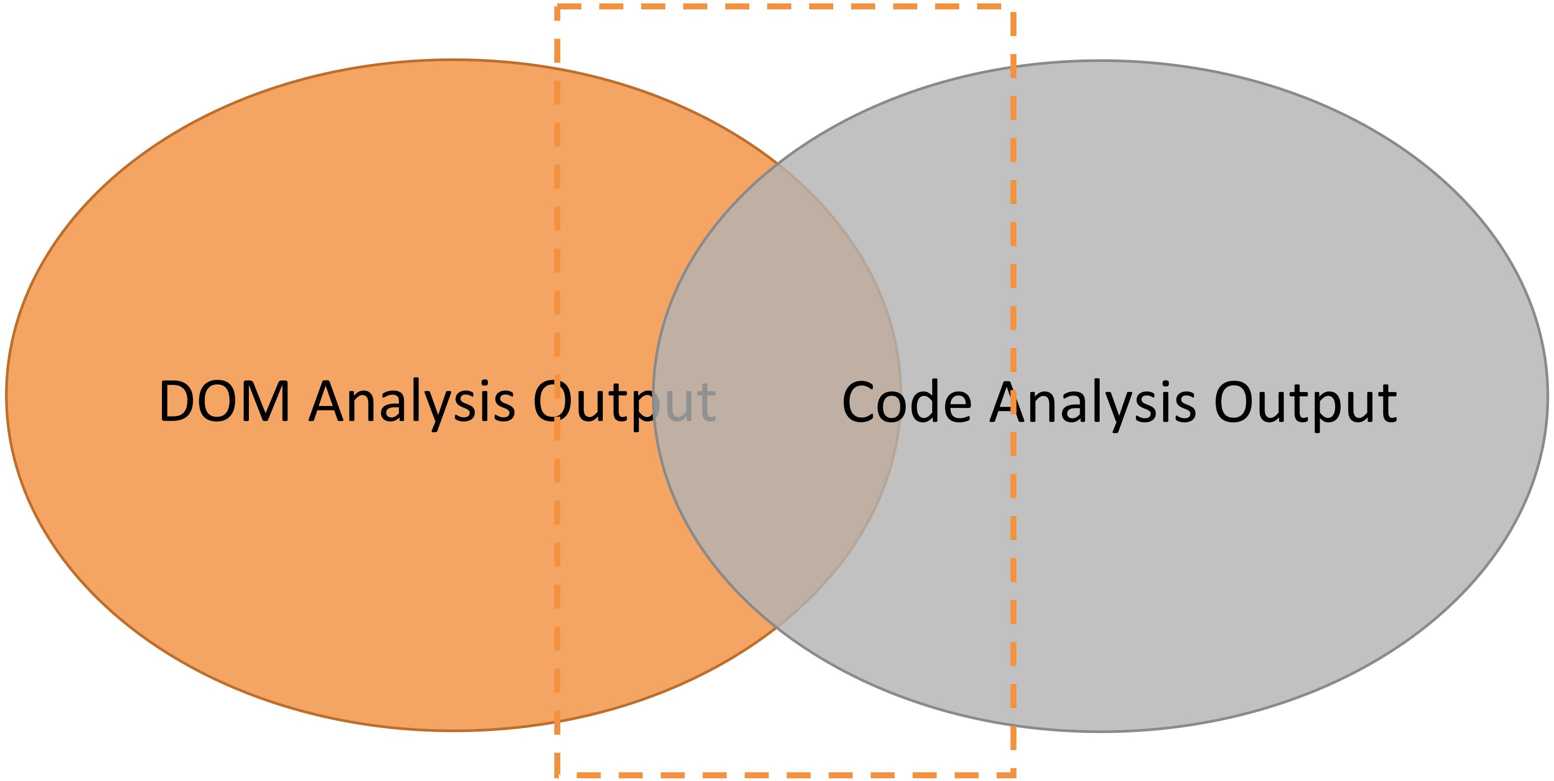
DOM states exhibit patterns



Dompletion: Approach

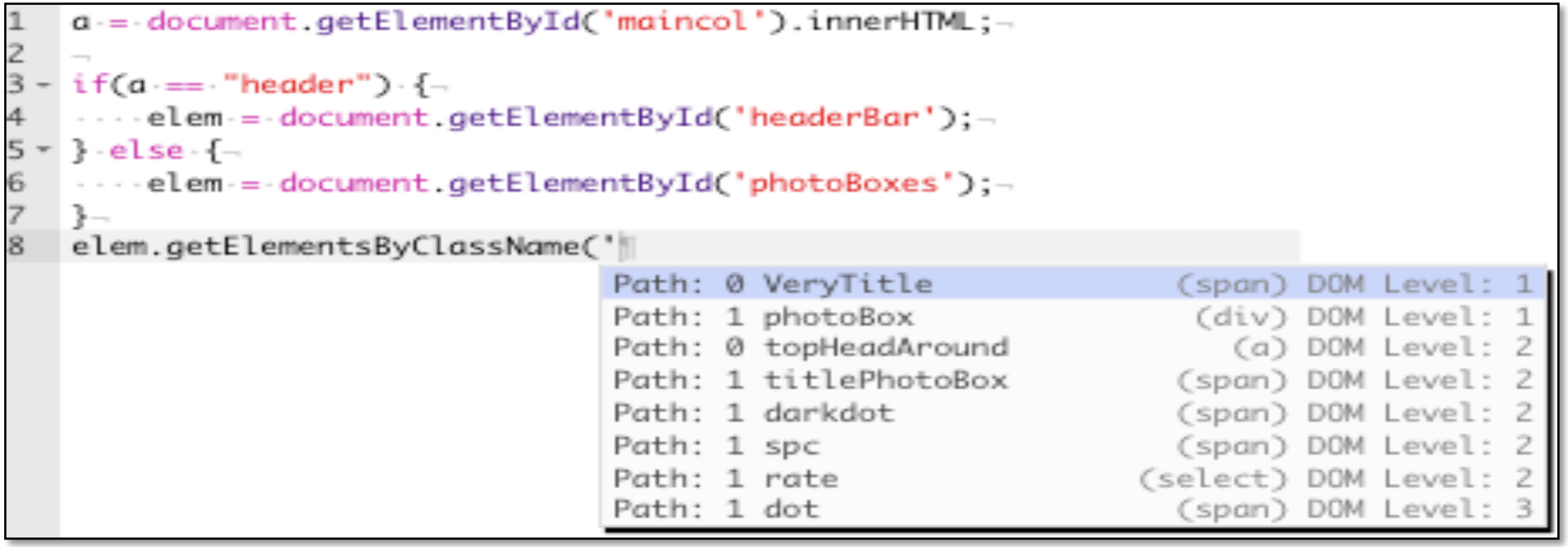


Dompletion: Suggestion Generation



Suggestions

Dompletion: Screenshot

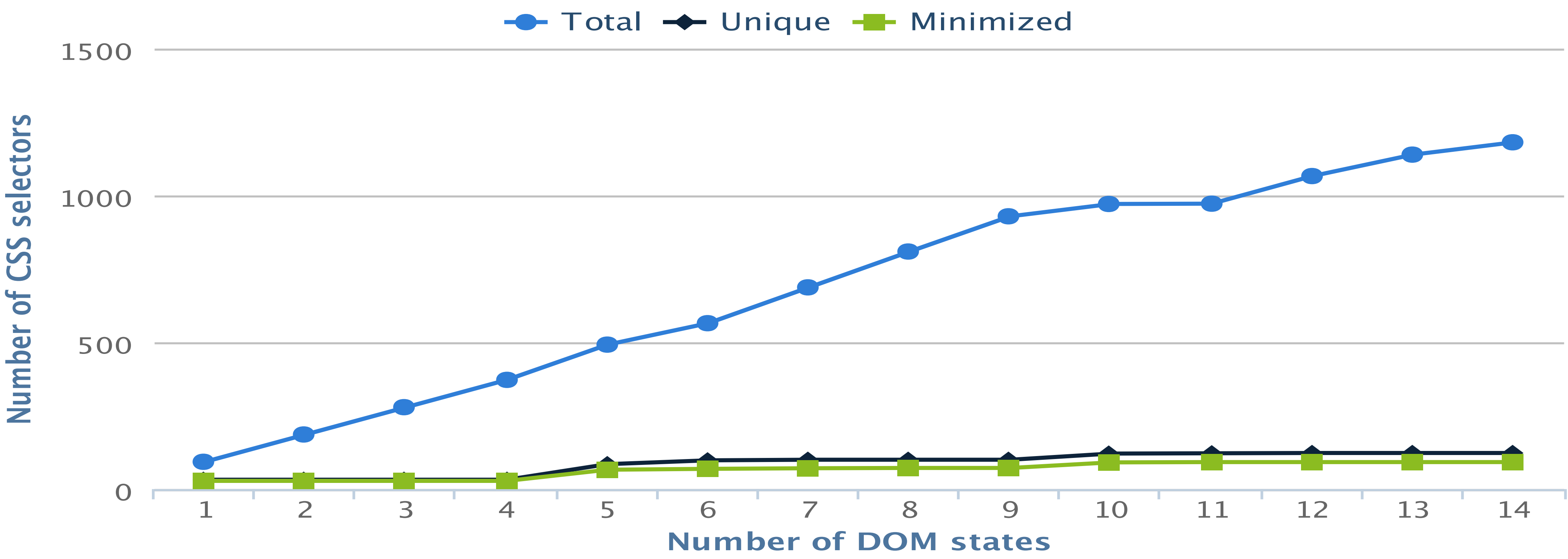


Implemented in the Brackets IDE

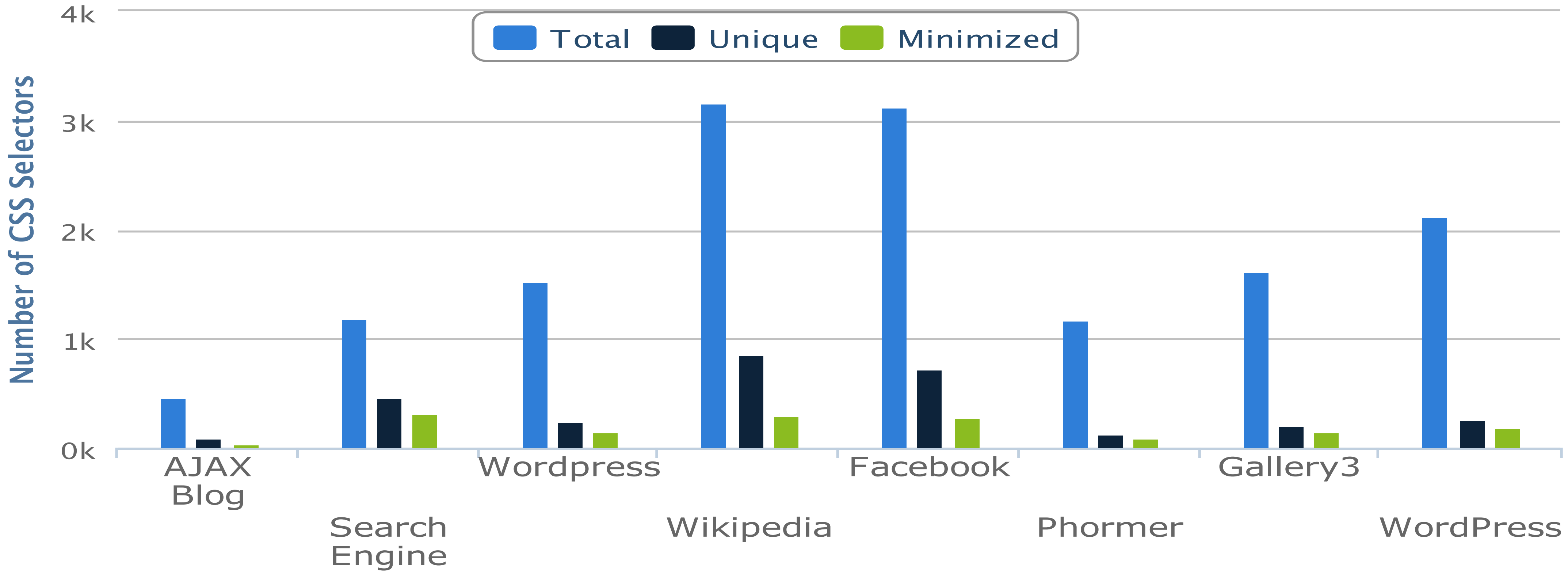
Dompletion: Evaluation

- 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 it Dompletion in helping the web developers with code completion tasks?

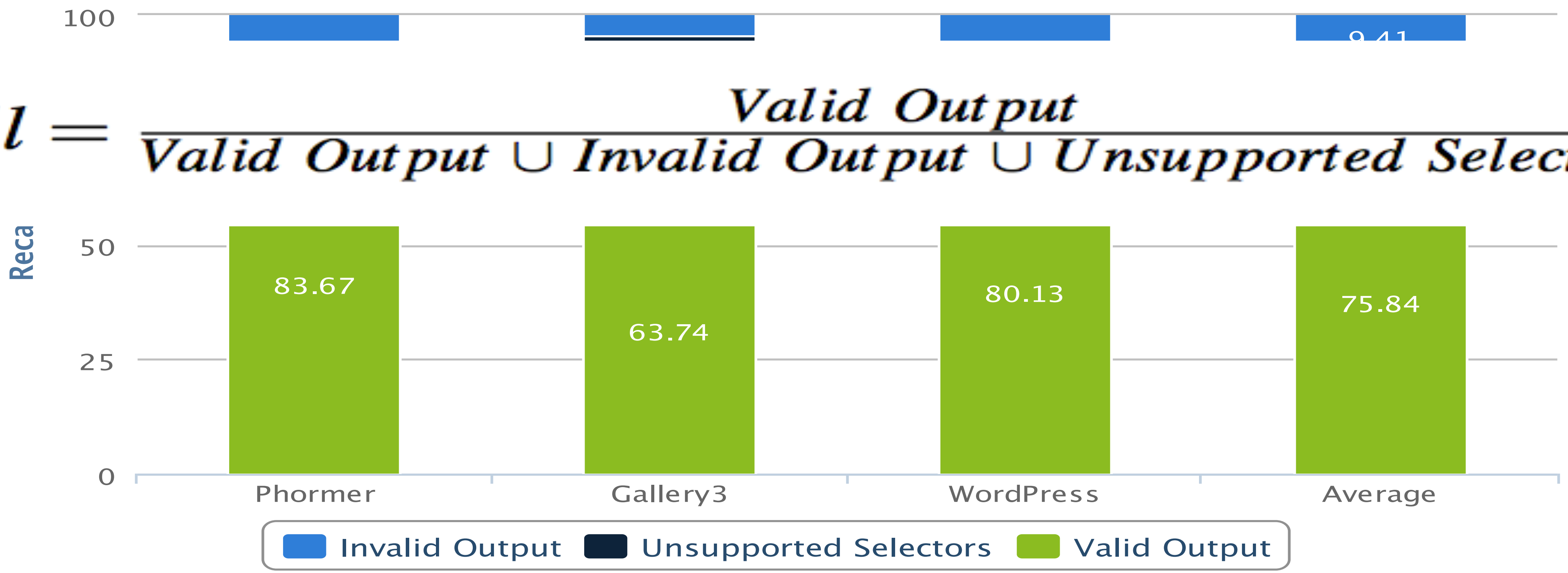
Dompletion: Convergence (RQ1)



Dompletion: Convergence (RQ1)

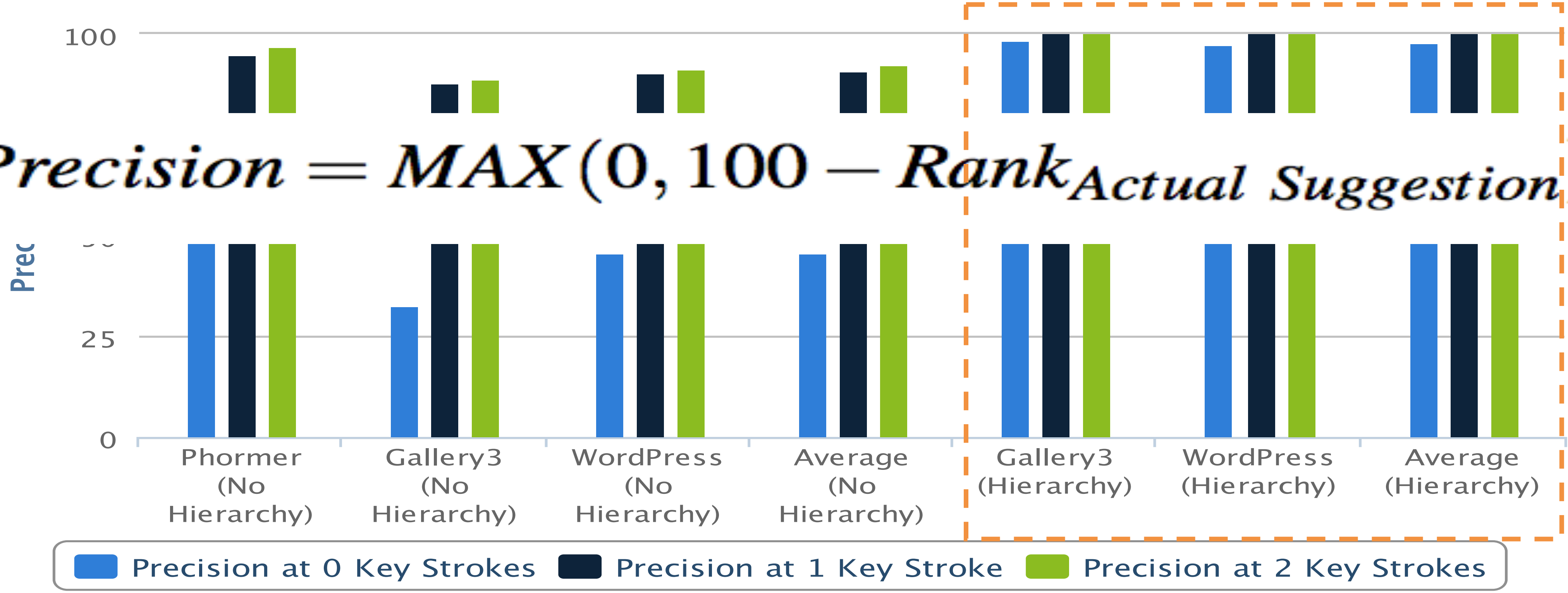


Dompletion: Accuracy (RQ2)



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

Dompletion: Accuracy (RQ2)



$$Precision = MAX(0, 100 - Rank_{Actual\ Suggestion})$$

Dompletion: 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 Dompletion	4	4m 28s	76.25%	47.5%

LED: Motivation

- DOM-JS interactions is a major source of errors [ESEM'13]

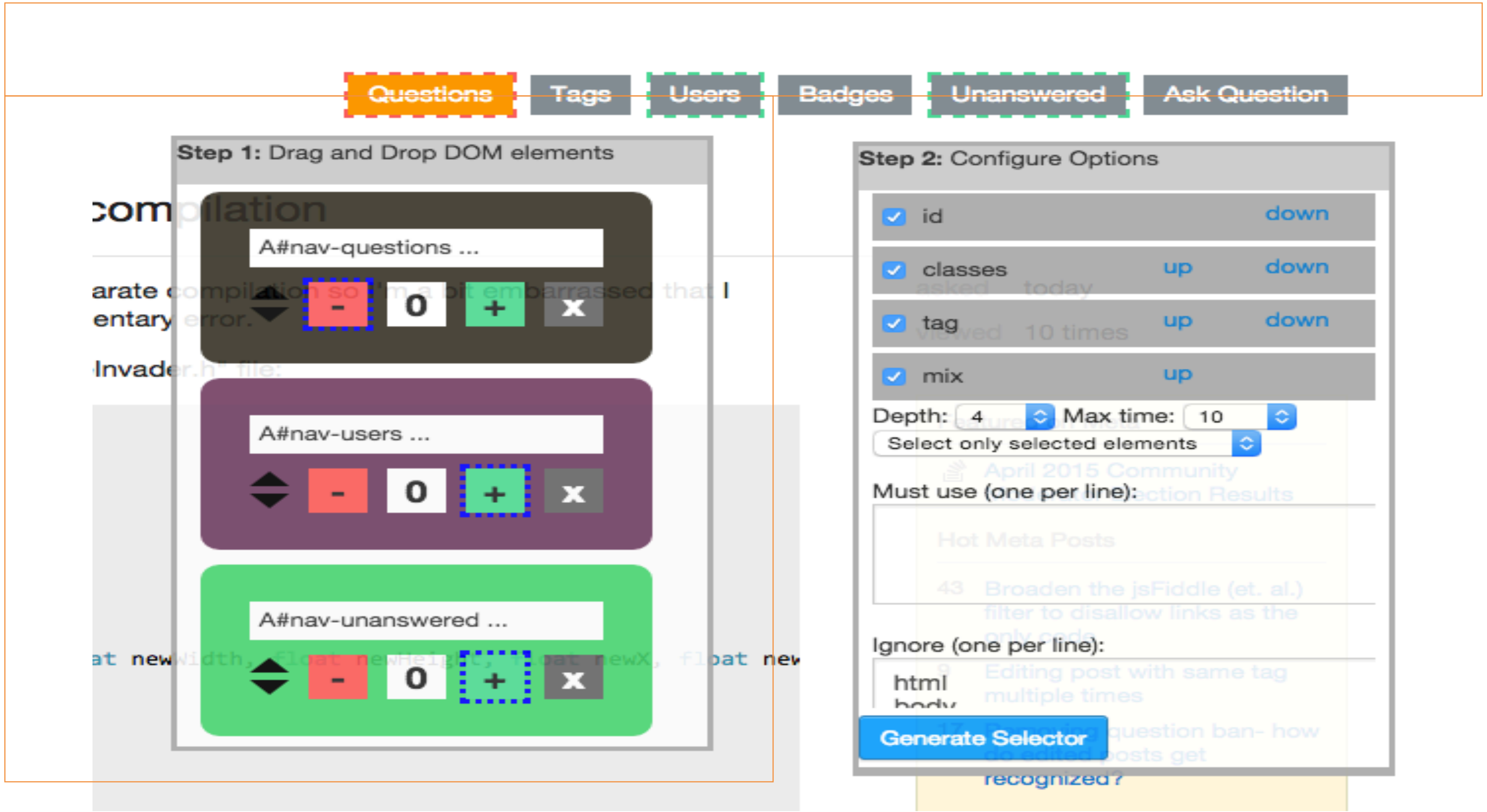
- **Performed through CSS selectors in JS code**
  - 22% of the CSS selectors are used to select **multiple** DOM elements
  - 35% of the CSS selectors are a **combination** of multiple atomic CSS selectors

- **Goal: Automate synthesis of CSS selectors**

LED: Main Idea

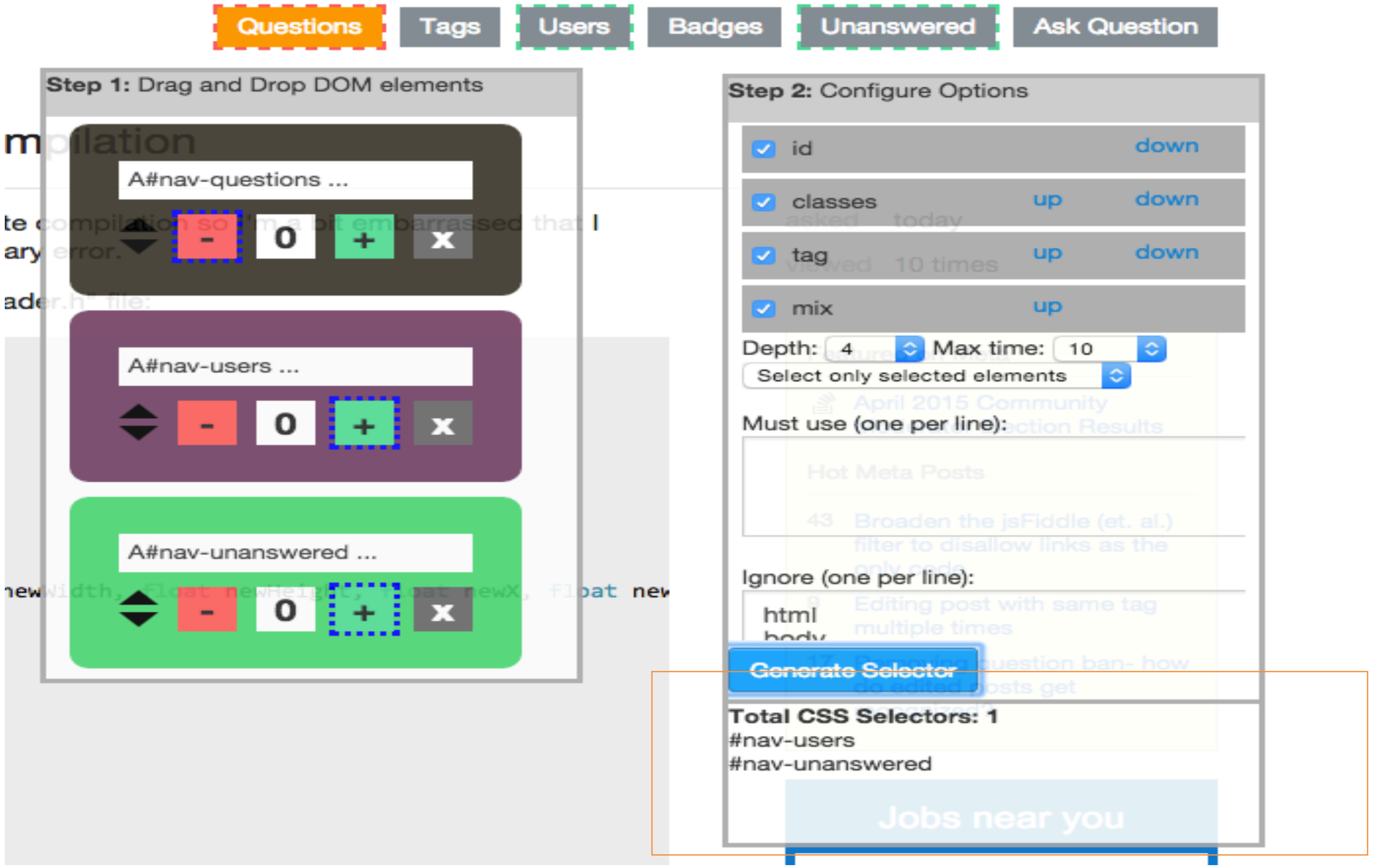
- Ask developer to provide DOM elements as positive and negative examples for selector
- Analyze distinguishing properties of elements and generate constraints for the properties
- Leverage SAT solvers to solve constraints
  - Rank selectors based on “goodness” criteria

LED: Input



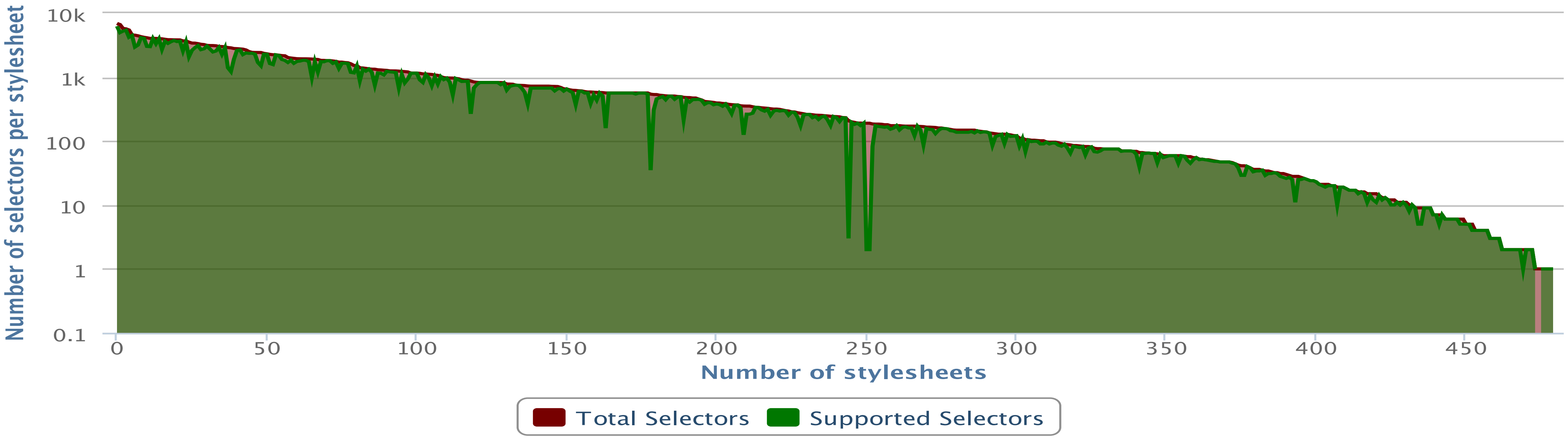
Questions
Tags
Users
Badges
Unanswered
Ask Question
Step 1: Drag and Drop DOM elements
A#nav-questions ...
0
A#nav-users ...
0
A#nav-unanswered ...
0
Step 2: Configure Options
id
down
classes
up
down
tag
up
down
mix
up
Depth: 4
Max time: 10
Select only selected elements
Must use (one per line):
Ignore (one per line):
html
Generate Selector

LED: Output


Questions
Tags
Users
Badges
Unanswered
Ask Question
Step 1: Drag and Drop DOM elements
A#nav-questions ...
A#nav-users ...
A#nav-unanswered ...
Step 2: Configure Options
id
classes
tag
mix
up
down
Depth: 4
Max time: 10
Select only selected elements
Must use (one per line):
Ignore (one per line):
html
Generate Selector
Total CSS Selectors: 1
#nav-users
#nav-unanswered

LED: Evaluation (Coverage)

Analyzed CSS selectors used in Alexa's top 500 web applications



86% of the CSS selectors used in top 500 web applications are supported by LED

LED: Evaluation (Accuracy)

- **Intercepted DOM API calls**
  - Wordpress, Gallery3 and Phormer
- **Analyzed DOM elements**
- **Synthesized CSS selectors**
- 98% Recall and 92% Precision

LED: Evaluation (Performance)

- Average time: 0.22 Seconds
- Max time: 0.5 Seconds
- Tool and video available at

http://ece.ubc.ca/~kbajaj/led.html

Talk Outline

- Motivation and Goals
- Empirical Study of reliability
- Reliability Improvements
- Program Understanding
- IDE Support
- Other Work and Future Directions

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]

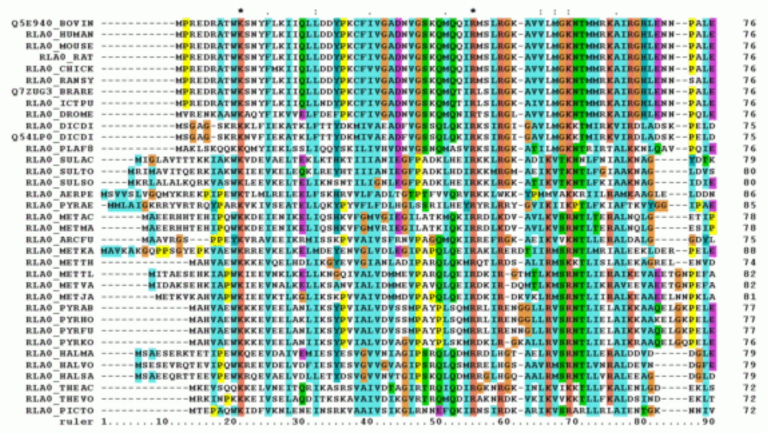
TypeScript and JavaScript Frameworks

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

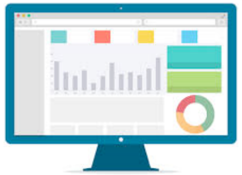
- Frameworks for construction JavaScript Apps
  - Flapajax [Guha-OOPSLA'09]
  - Arrows [Hicks-DLS'09]

Future Work

- **Understanding large-scale applications using trace compression**
  - Current traces are too large for comprehension
  - Use of algorithms inspired by gene sequence matching



- **JavaScript in the IoT Space**
  - Understanding the sources of errors and how they affect the system
  - Targeted techniques for improving reliability subject to resource constraints



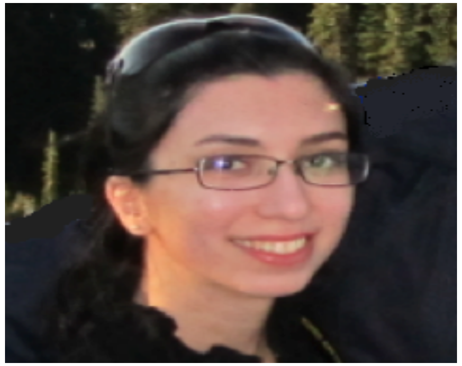
Conclusions

- **JavaScript is one of the most prominent languages today**
  - Five years of research into studying and understanding JavaScript applications
  - Performed empirical studies to identify sources of JavaScript bugs
  - Built tools for improving the reliability and programmability of JavaScript
  - Evaluated tools using real-world applications and case studies

- **Open Challenges**
  - Non-DOM related faults
  - Scalable IDE support
  - JavaScript on IoT

Karthik Pattabiraman

Ali Mesbah



Saba Alimadadi

Kartik Bajaj

Frolin Ocariza

Sheldon Sequira

NSERC CRSNG

Canada Foundation for Innovation
Fondation canadienne pour l'innovation

intel

Microsoft Research