

defect detection for the wayward web

Andrew J. Ko



supported by Microsoft and the National Science Foundation
under Grant CCF-0952733

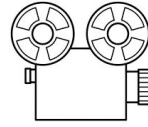


Andrew J. Ko

Assistant Professor
The Information School
University of Washington

computer science
psychology
design

Ph.D from the HCI Institute at
Carnegie Mellon University



01001
10100
10101



software is a
fascinating medium
for human expression

I want to make it
easier to **express**
and **understand**
ideas as code

research I've done

studies of software development as if it were created by people

credit to Rob DeLine at MSR

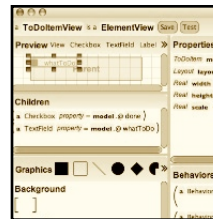
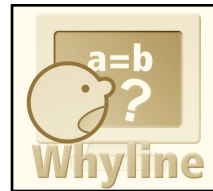
of debugging

of teamwork

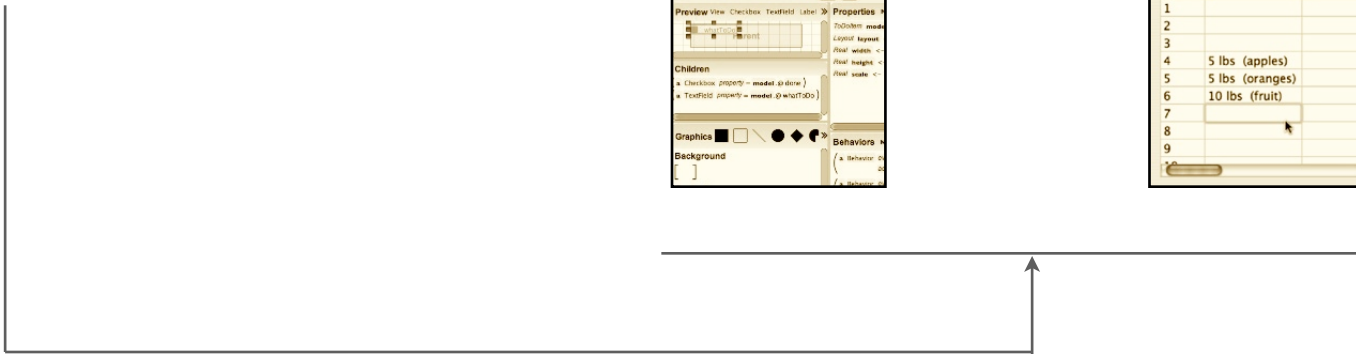
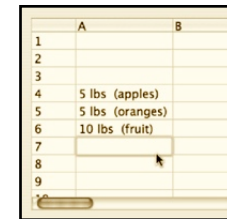
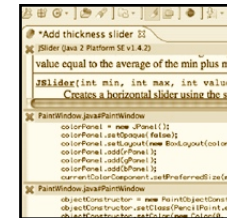
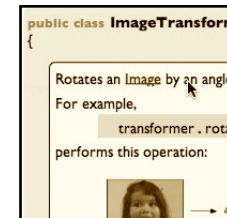
of API learning

of open source

debugging tools



programming tools



research I'm doing with the **usegroup**

studies

open bug reporting



bug triage meetings



Stack Overflow



diagnostic thinking



tools

next generation help

automating bug severity measurements

improved API documentation

teaching debugging skills

defect detection for the web

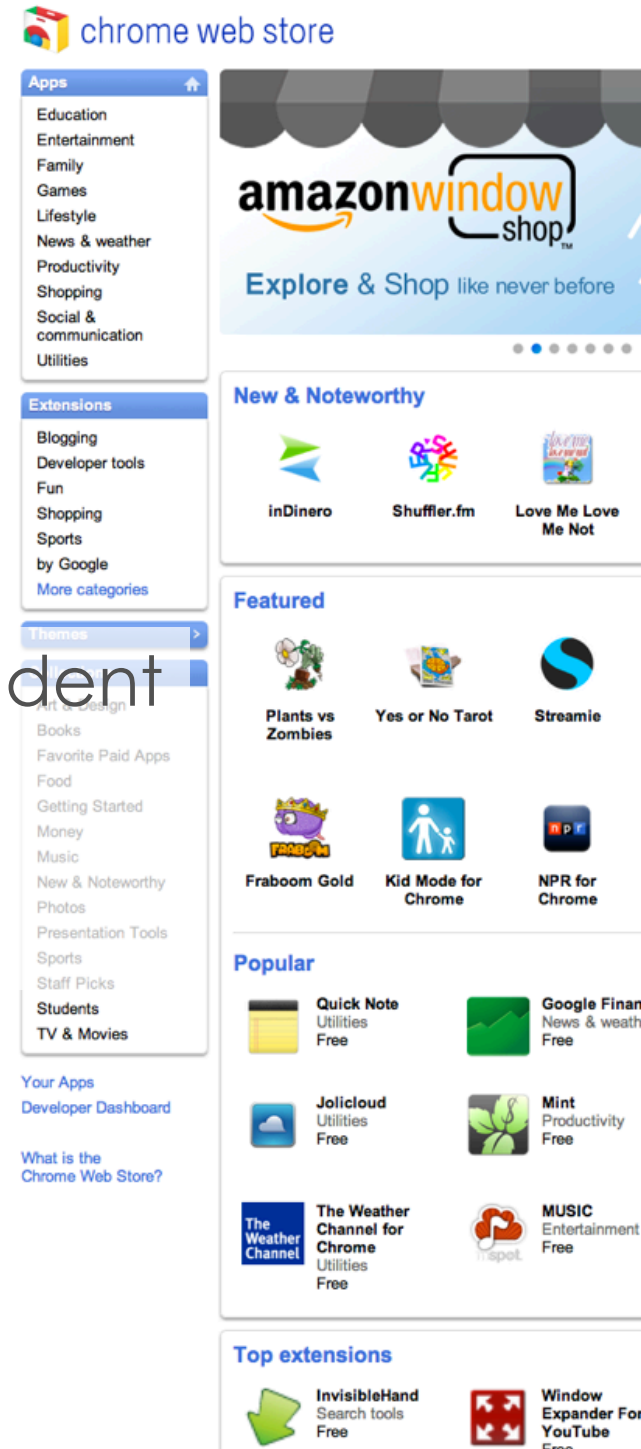
defect detection for the web

an increasingly popular platform for interactive software applications

platform-independent

information rich

highly flexible



defect detection for the web



the very languages that **enable**
this flexibility also impose some
serious **tradeoffs**...

dynamic typing means that many errors aren't found until runtime

noon-3 pm
OSP

Come out to the Olympic Sculpture Park for the last family festival of our big park season se
The day will be filled with F-U-N
music,dancing, story telling, art r
more! Plus, learn how you and y
salmon safe and explore how Na
encourages us to care for our na

Free and open to the public!

PERFORMERS

T'ilishubdub (Duwamish)
Roger Fernandes (Lower Elwha Klallam)
Red Eagle Soaring
Elder Chris Morganroth (Quileute)
Daisy Chain

ART MAKING

Printmaking
Flower Pressing
Sketching
Photo Booth
And More!

FAMILY TOUR Explore the shoreline of the Olympic Sculpture Park and discover a beautiful space that displays art while providing a shoreline habitat for migrating juvenile salmon. See the unique pocket beach featuring native shoreline plantings, marine life, and works of art

COMING UP

Sat. Sep

11

Family Festival Presented
by Target
noon-3 pm

```
Microsoft JScript runtime error '800a138f'  
'strDisplayText' is null or not an object  
/getout/comingup.inc, line 26
```

Sat. Sep

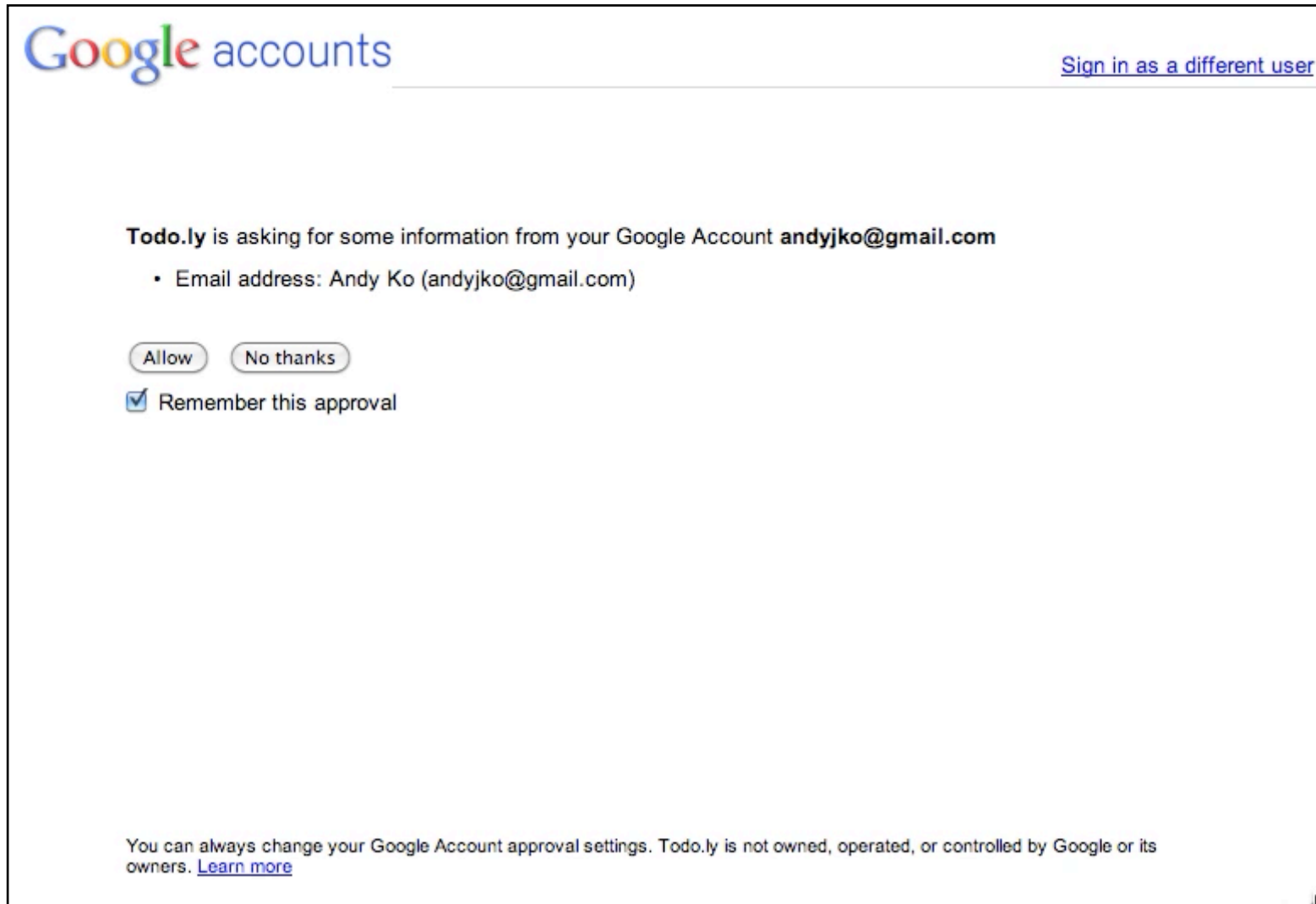
25

Educator Workshop
10:30 am-3:30 pm

Join us for our annual OSP
Educator Workshop! This ... >>

```
Microsoft JScript runtime error '800a138f'  
'strDisplayText' is null or not an object  
/getout/comingup.inc, line 26
```


JavaScript's flexibility in constructing user interfaces **dynamically** makes it easy to overlook broken execution contexts without significant testing



despite all of the **variation** in how web applications are written

there is **uniformity** in developers' mistakes that we can detect and highlight

Cleanroom

```
13 </head>
14
15 <script type="text/javascript" src="code.js"></script>
16 <link href="style.css" type="text/css" rel="stylesheet">
17
18 </head>
19
20
21 <!-- On load, clear the calculator -->
22 <body onload="">
23
24 <div class="calculatorBody">
25
26 <div id="display" class="display"></div>
27
28 <!-- On click, press digit 1 -->
29 <button onclick="">1</button>
30 <!-- On click, press digit 2 -->
31 <button>2</button>
32 <!-- On click, press digit 3 -->
33 <button>3</button>
34 <!-- On click, press operation + -->
35 <button>+</button>
36 <br>
37 <!-- On click, press digit 4 -->
38 <button>4</button>
39 <!-- On click, press digit 5 -->
```

The class calculatorBody only appears once, are you sure it's right?

statically detecting a large class of JavaScript errors at edit time

FeedLack

FeedLack

project discussion

FeedLack found 1 place that appear to be missing feedback:

- ✗ `post(text)` at `index.html` may not produce feedback

FeedLack found 4 places that appear to always produce feedback:

- ✓ `mouseover` at `index.html` always produces output
- ✓ `click` at `index.html` always produces output
- ✓ `keypress` at `index.html` always produces output
- ✓ `mousedown` at `index.html` always produces output

```
5 function isValid(comment) {
6   if(comment == '') $('#post').text('write something')
7   return comment != '';
8 }
9 function post(text) {
10  if(isValid(text)) {
11    $.get('comment.php', { comment: text });
12  }
13  else {
14    alert('Your comment is invalid.');
```

`post(text)` at `index.html`

When the user performs a

- `submit(index.html:1)`, or
- `click(index.html:2)`

this path may fail to produce output:

1. `post()` is entered `index.html`
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called `index.html`
assumes this can `get(comment)`, because no other functions by this name were found
3. `isValid()` is entered `index.html`
assumes this function can produce output because `text()` can produce output
4. the expression of `index.html` is false
5. the expression of `index.html` is true
assumes condition can be true
6. several functions are called that do not affect output
assumes `get()` that found does not affect output
7. `post()` is exited `index.html` without producing output

verifying the presence of feedback in response to user input

Cleanroom

```
13
14 <head>
15
16   <script type='text/javascript' src='code.js'></script>
17   <link href='style.css' type='text/css' rel='stylesheet'>
18
19 </head>
20
21 <!-- On load, clear the calculator -->
22 <body onload=''>
23
24 <div class='calculatorBody'>
25
26   <div id='display' class='display'></div>
27
28   <!-- On click, press digit 1 -->
29   <button onclick=''>1</button>
30   <!-- On click, press digit 2 -->
31   <button>2</button>
32   <!-- On click, press digit 3 -->
33   <button>3</button>
34   <!-- On click, press operation + -->
35   <button>+</button>
36   <br>
37   <!-- On click, press digit 4 -->
38   <button>4</button>
39   <!-- On click, press digit 5 -->
```

The class `calculatorBody` only appears once: are you sure it's right?

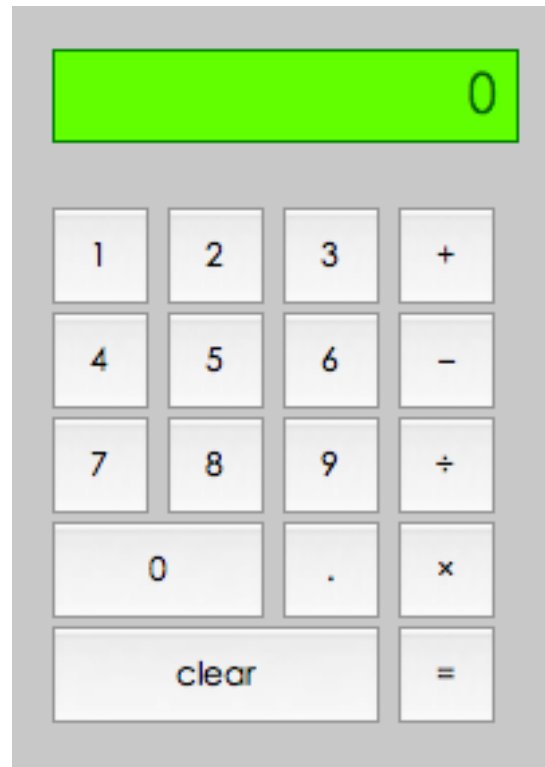
with

Jacob Wobbrock

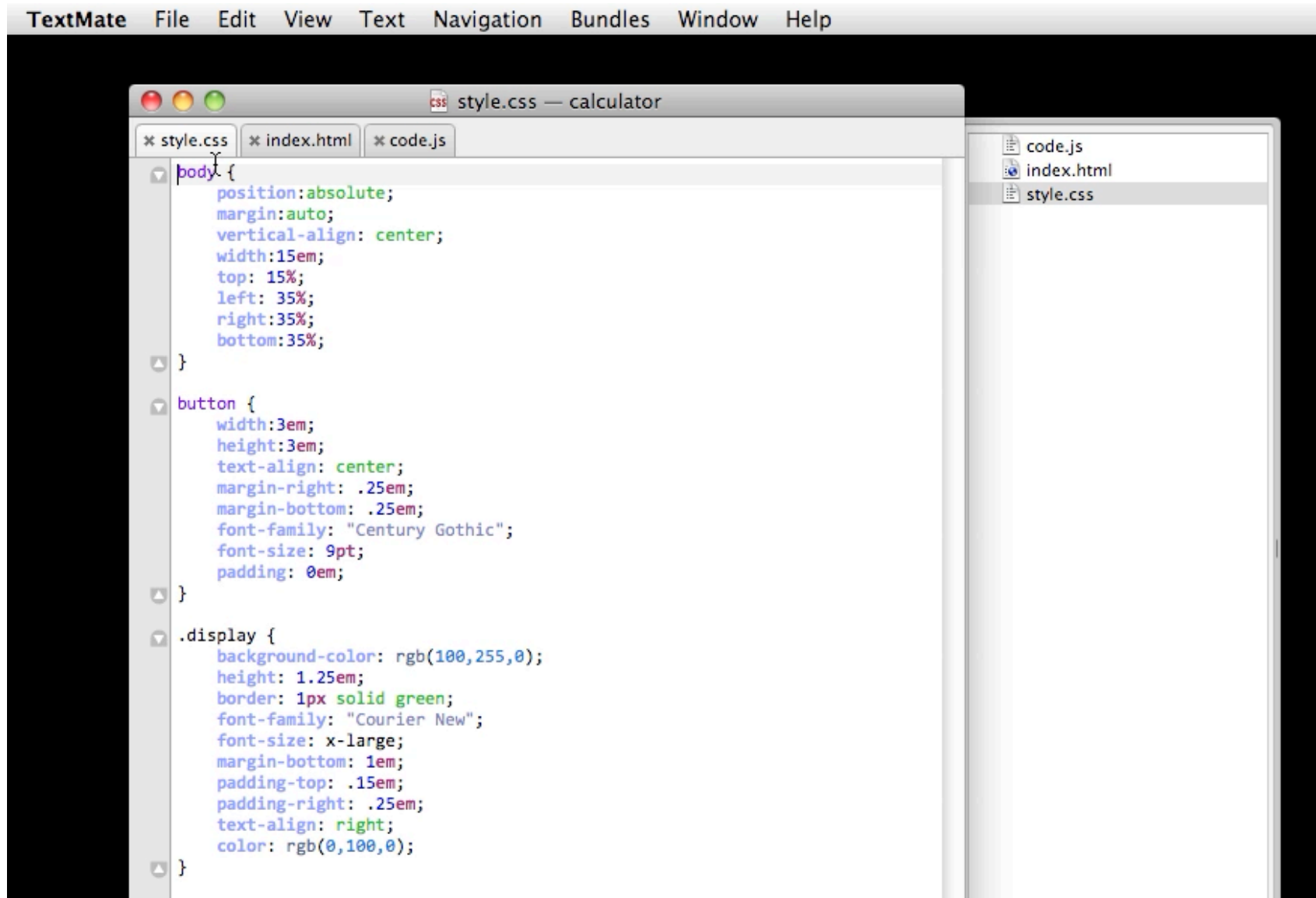
Assistant Professor

The Information School

the web is great for rapid prototyping ...



the web is great for rapid prototyping ...



5 minutes later ...

of testing

of debugging

of reviewing my code

dynamic languages strike again...

```
<!-- On Load, clear the calculator -->
```

```
<body onload=''>
```

```
<div class='claculatorBody'> }
```

```
<div id='display' class='display'></div>
```

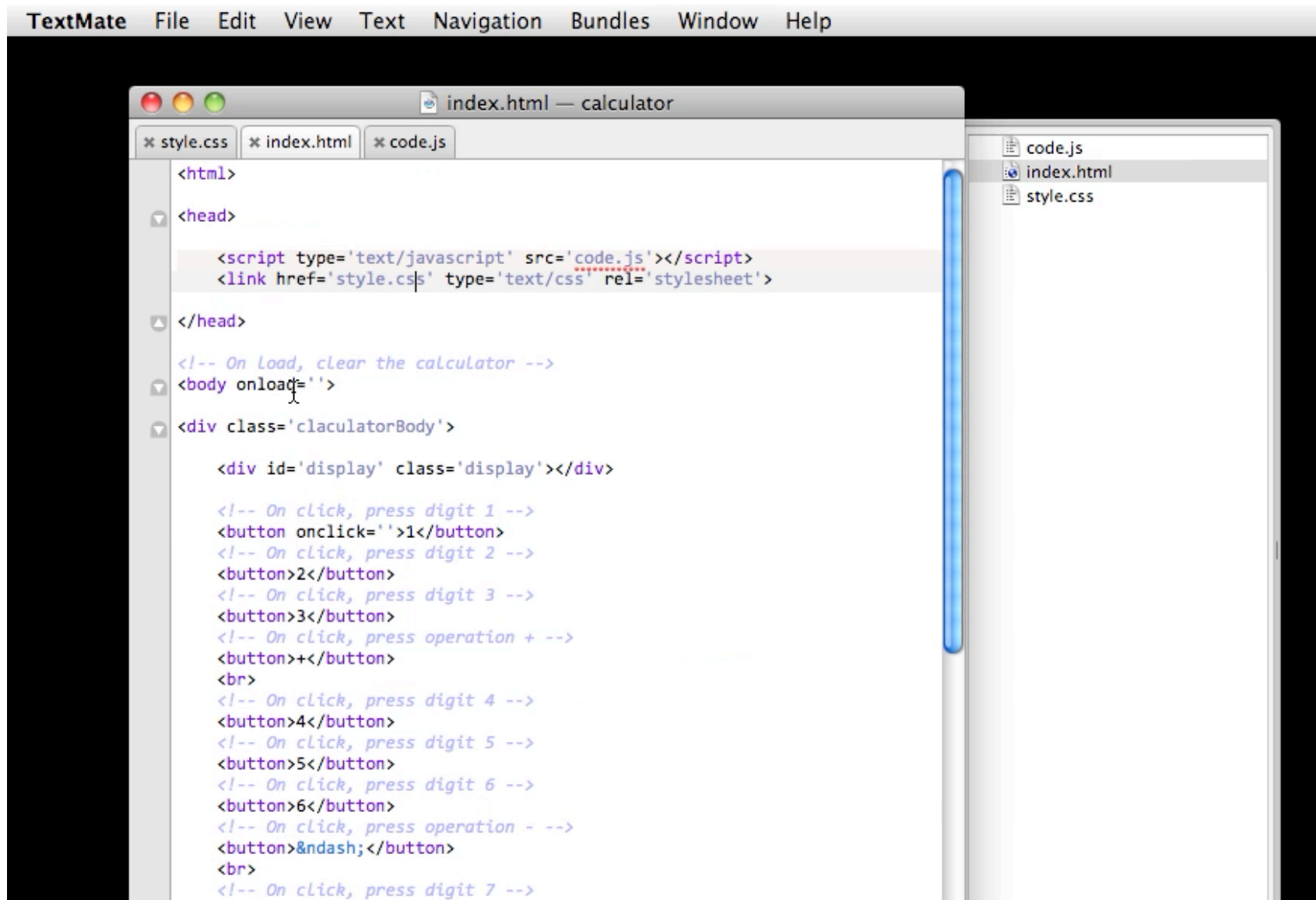
```
<!-- On click, press digit 1 -->
```

```
<button onclick=''>1</button>
```

```
<!-- On click, press digit 2 -->
```

```
<button>2</button>
```


only after testing was this typo apparent...



The screenshot shows the TextMate editor interface. The title bar reads "TextMate File Edit View Text Navigation Bundles Window Help". The main window is titled "index.html — calculator" and contains three tabs: "style.css", "index.html", and "code.js". The "index.html" tab is active, displaying the following HTML code:

```
<html>
<head>
  <script type='text/javascript' src='code.js'></script>
  <link href='style.css' type='text/css' rel='stylesheet'>
</head>
<!-- On Load, clear the calculator -->
<body onload=''>
<div class='claculatorBody'>
  <div id='display' class='display'></div>
  <!-- On click, press digit 1 -->
  <button onclick=''>1</button>
  <!-- On click, press digit 2 -->
  <button>2</button>
  <!-- On click, press digit 3 -->
  <button>3</button>
  <!-- On click, press operation + -->
  <button>+</button>
  <br>
  <!-- On click, press digit 4 -->
  <button>4</button>
  <!-- On click, press digit 5 -->
  <button>5</button>
  <!-- On click, press digit 6 -->
  <button>6</button>
  <!-- On click, press operation - -->
  <button>&ndash;</button>
  <br>
  <!-- On click, press digit 7 -->
```

The file explorer on the right shows the project structure with files "code.js", "index.html", and "style.css". A red squiggly line under the "code.js" attribute in the script tag indicates a typo. The cursor is positioned at the end of the "onload" attribute in the body tag.

current tools do
not detect these
name errors...

```
Microsoft JScript runtime error '800a138f'  
'strDisplayText' is null or not an object  
/getout/comingup.inc, line 26
```

```
<body onload= >  
<div class='claculatorBody'>⌵  
  <div id='display' class='display'></  
  <!-- On click, press digit 1 -->
```

HTML/CSS **validators** don't catch them

JSLint doesn't catch them

Google's **Closure** compiler doesn't catch them

code completion can help prevent them, but
type inference isn't always possible...

what can we do about them?

spell checking?

text entry error detection?

fancy static type inference? (DoctorJS)

we tried all of these...

two observations

in any programming language, names are used to **uniquely refer** to data and behavior

human motor performance with keyboards is prone to **duplication, omission, transposition**, and **substitution** errors leading to “off-by-one” errors in names

the resulting hypothesis

$$\mathbf{frequency(name)} \propto \mathbf{validity(name)}$$

the uniqueness heuristic

any **name** or **name sequence** that appears once in a program is **wrong**

e.g., claculatorBody, consloe.log()

how often is this right?

would warnings based on it be useful?

Cleanroom

highlights violations of the uniqueness heuristic after each keystroke

```
1  .calculatorBody {
2    text-align: left;
3    background-color: lightGray;
4
5  body {
6    position: absolute;
7    margin: auto;
8    vertical-align: center;
9    width: 13em;
10   top: 15%;
11   left: 35%;
12   right: 35%;
13   bottom: 35%;
14 }
15
16 button {
17   width: 3em;
18   height: 3em;
19   text-align: center;
20   margin-right: .25em;
21   margin-bottom: .25em;
22   font-family: "Century Gothic";
23   font-size: 9pt;
24   padding: 0em;
25 }
26
27 display: flex;
```

The CSS class name `calculatorBody` doesn't appear anywhere else in your code. Perhaps you meant `claculatorBody`?

files

code.js (15)

index.html (1)

style.css (2)(1)

add new file

reset the demo

Ko, A.J. and Wobbrock, J.O. (2014) **Cleanroom: Edit-Time Error Detect with the Uniqueness Heuristic**. *IEEE Symposium on Visual Languages Human-Centric Computing*, Madrid, Spain, September 21-24, to appear.

Thanks to the [BespIn](#) team for a great editor and Douglas Crockford for [JSLint](#). Also thanks to the [ANTLR](#) team and the various users who've contributed to HTML/CSS/ECMAScript token grammars. The rest of the content on this site is property of the [University of Washington](#). Thanks to MSIM's Jeroen van den Eijndhof for adding local storage support. Contact [AJ Ko](#) with questions or comments.

dub W ©

interaction design

during typing,
validation that name
isn't complete

```
page.lastEle
```

if it's an error,
developer is warned

```
page.lastElement = |
```

if it's an unused variable,
developer is reminded

```
page.lastElement = |
```

if declared, developer
developer gets
confirmation

```
page.lastElement =
```

interaction design

```
index.html (2) style.css (1)(1) code.js (14)  
onclick='calculator.cle|();'>clear</button>
```



```
index.html (1) style.css (1)(1) code.js (13)  
onclick='calculator.clear();'>clear</button>
```

file-level counts
updated on each
keystroke to notify of
cross-file changes

interaction design

```
52 le='width: 10.2em' onclick='claer();'>clear</butt  
53 cl  
54
```

The function name **clae**r only appears once and doesn't appear
Perhaps you meant **clear**?

alternate names are suggested using
Levenshtein string distance

implementation

after **each keystroke**

incremental tokenization

identifiers tagged with one or more
token types

HTMLTag

HTMLAttributeName

HTMLClass

HTMLID

CSSPropertyName

CSSValue

JSFunction

JSProperty

JSVariable

JSLiteral

implementation

...

string literals are tagged as JavaScript identifiers, HTML ids, HTML classes, CSS values since they are often used to refer to identifiers

Cleanroom has a dictionary of W3C standard API names

works even in the presence of **parsing errors**

implementation

...

table of name tokens by tag is created

table of adjacent **two name sequences** is created.

names or pairs of names that appear once are selected for warnings

names for which **Levenshtein string distance** from warned name < 1 are suggested as alternatives

evaluation

online experiment

Cleanroom + JSLint versus **JSLint** only

developers asked to finish



Cleanroom warnings were tracked in JSLint condition, **but not displayed**

participants asked to finish...

18 inline onclick event handlers

~76 lines of calculator function implementations



the tests

automated test launched the web site and tested whether programmatic clicks on the the calculator would provide correct answers for



clear → 0

9 + 5

9 - 5

9 x 5

9 / 5

Each time you preview, Cleanroom will run these automated tests. When you've passed them all, you can submit your e-mail address for the \$10 gift certificate.

- clear test failed
- + test failed
- test failed
- × test failed
- ÷ test failed

the participants

94 visited

40 started task

22 typed for more than 3 minutes

16 made substantial progress on the task

8 Cleanroom and **8 control** participants

no significant difference in JavaScript
experience

“In the past month, I’ve written JavaScript **weekly**”

data collected

whether a warning was **active** after the last recorded keystroke

the **duration** a warning was active

the **kind** of token warned

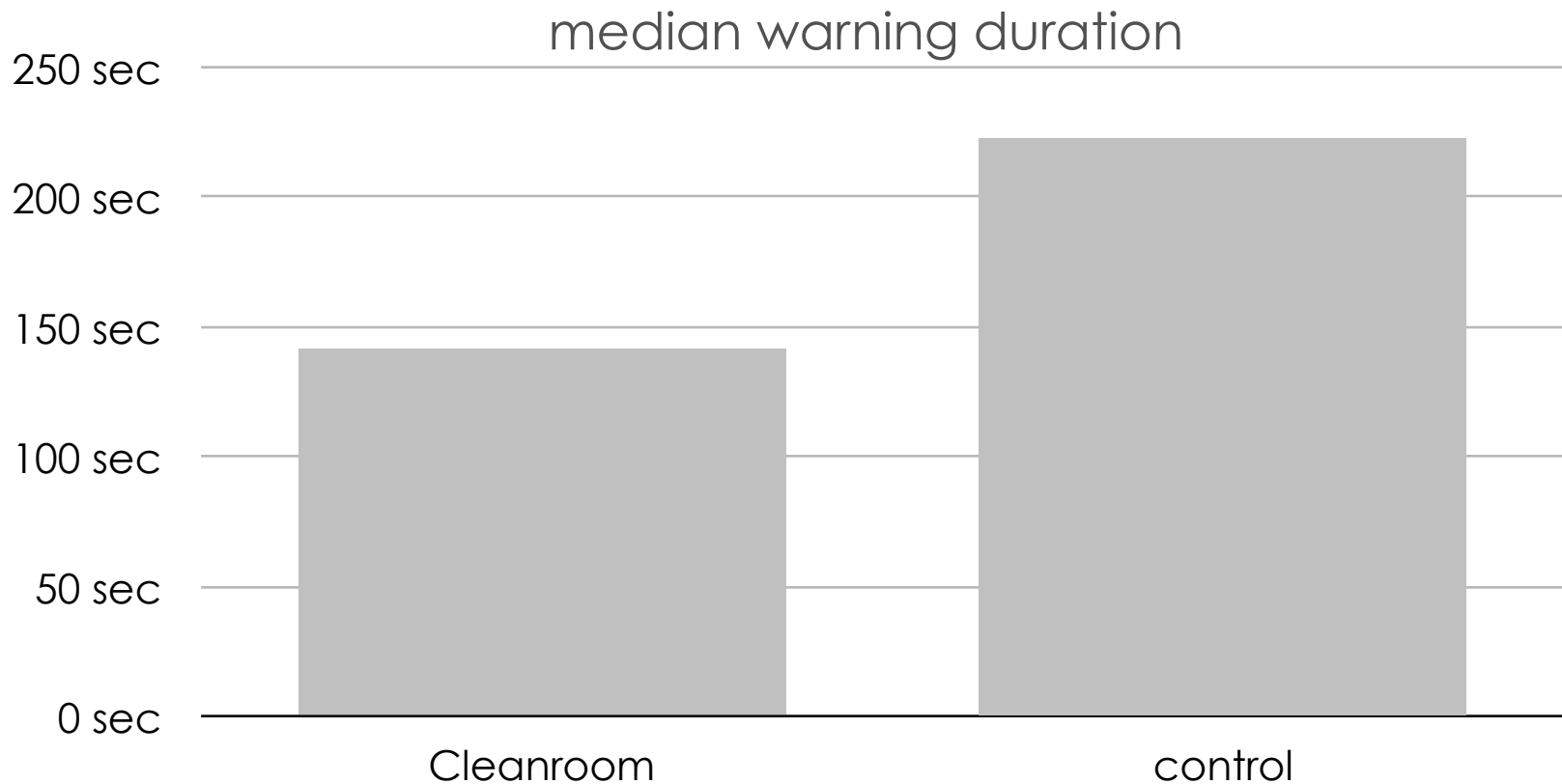
whether the warning was on a **declaration**

whether the warning disappeared because of a **direct** edit on the name

how many times a warning was **executed** while active

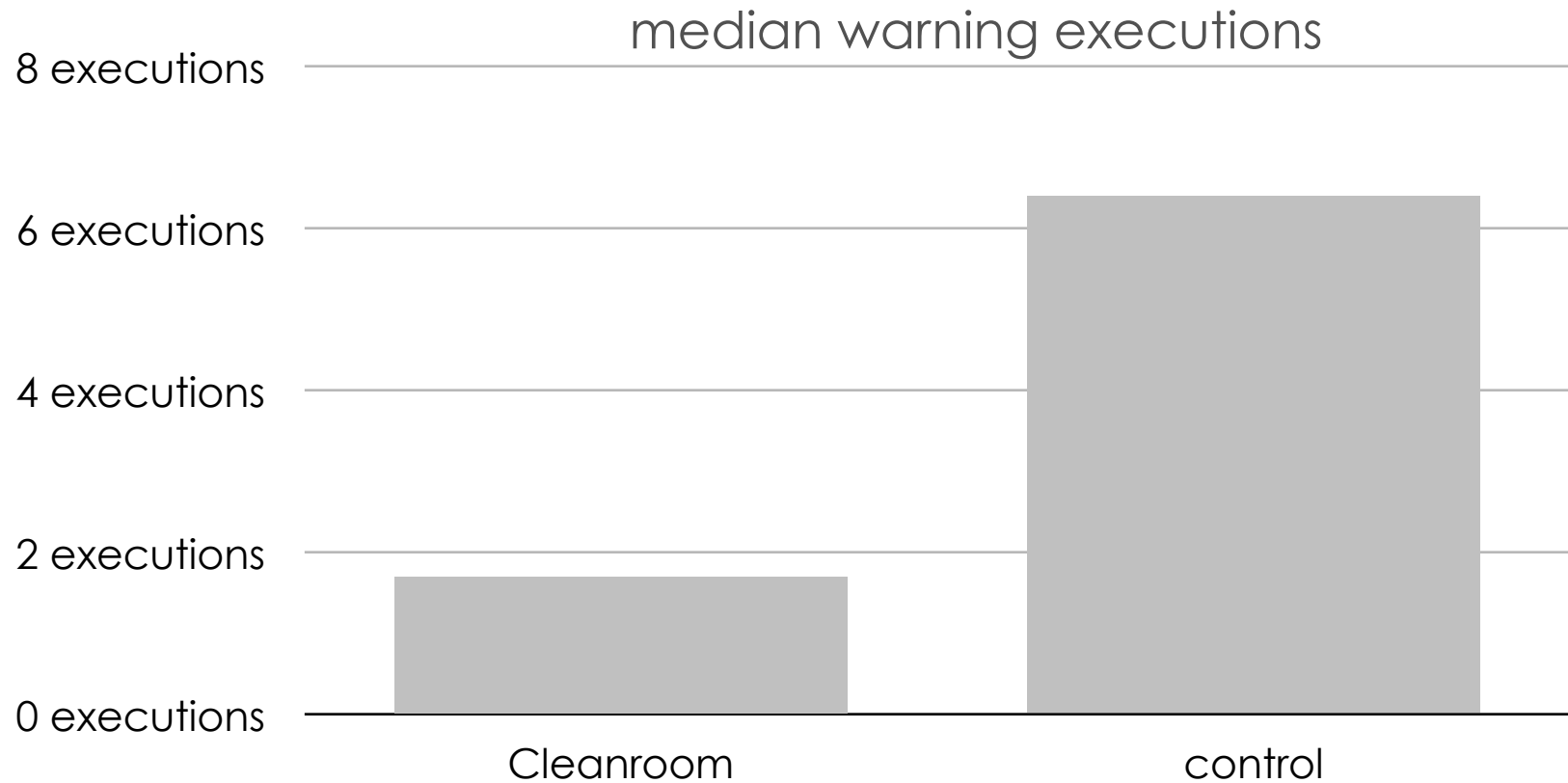
results

warnings were **active for significantly less time** in the Cleanroom condition ($p < .01$)



results

Cleanroom developers **executed** warned names significantly fewer times ($p < .01$)



results

errors that Cleanroom developers fixed

undeclared names

unused names

typos (e.g., `parseFloat`, `getElementByID`, `onclick`, `alert_box`)

syntax from other languages (e.g., `dim` from Visual Basic)

APIs from other languages (e.g., `sum` instead of `add`)

type declarations (e.g., `int`)

results

none of the warnings in the program were false positives

some of the warnings were not severe

e.g., unused variables had no consequence on behavior

limitations

can't detect errors that occur more than once

can't detect errors in dynamically generated names

there are bound to be a variety of false positives in the wild

e.g., pre- and postfix literals of dynamically generated names, as in ("week" + number)

Cleanroom

```
13 </head>
14
15 <script type="text/javascript" src="code.js"></script>
16 <link href="style.css" type="text/css" rel="stylesheet">
17
18 </head>
19
20
21 <!-- On load, clear the calculator -->
22 <body onload="">
23
24 <div class="calculatorBody">
25
26 <div id="display" class="display"></div>
27
28 <!-- On click, press digit 1 -->
29 <button onclick="">1</button>
30 <!-- On click, press digit 2 -->
31 <button>2</button>
32 <!-- On click, press digit 3 -->
33 <button>3</button>
34 <!-- On click, press operation + -->
35 <button>+</button>
36 <br>
37 <!-- On click, press digit 4 -->
38 <button>4</button>
39 <!-- On click, press digit 5 -->
```

The class calculatorBody only appears once, are you sure it's right?

statically detecting a large class of JavaScript errors at edit time

FeedLack

FeedLack

project discussion

FeedLack found 1 place that appear to be missing feedback:

- ✗ `post(text)` at `index.html` may not produce feedback

FeedLack found 4 places that appear to always produce feedback:

- ✓ `mouseover` at `index.html` always produces output
- ✓ `click` at `index.html` always produces output
- ✓ `keypress` at `index.html` always produces output
- ✓ `mousedown` at `index.html` always produces output

```
5 function isValid(comment) {
6   if(comment == '') $('#post').text('write something')
7   return comment != '';
8 }
9 function post(text) {
10  if(isValid(text)) {
11    $.get('comment.php', { comment: text });
12  }
13  else {
14    alert('Your comment is invalid.');
```

`post(text)` at `index.html`

When the user performs a

- `submit` [`index.html`:1].
- `click` [`index.html`:2].

this path may fail to produce output:

1. `post()` is entered `index.html`: assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called `index.html`: assumes this call `isValid(comment)`, because no other functions by this name were found
3. `isValid()` is entered `index.html`: assumes this function can produce output because `text()` can produce output
4. the expression of `index.html` is `false`
5. the expression of `index.html` is `true`: assumes condition can be true
6. several functions are called that do not affect output: assumes `get()` that found does not affect output
7. `post()` is exited `index.html` without producing output

verifying the presence of feedback in response to user input

all over the web, apps are ignoring people

chrome web store

andyjko@gmail.com Search the store

Sorry, we don't support your browser just yet. You'll need Google Chrome to install apps, extensions and themes. [Download Google Chrome](#)

Apps > Productivity > Remember The Milk

remember the milk

Bob T. Monkey | Offline | Overview | Tasks | Locations | Contacts | Settings | Help | Logout

Thursday, October 21, 2010 | 12:20AM

Remember The Milk

✓ from www.rememberthemilk.com - Verified website

Free

Task management goodness used by millions worldwide. Makes managing your to-do list fun!

★★★★★ 152 ratings
23,834 users

Take over the world

- Call Caitlin Today
- Pick up the milk Today
- Buy gift for Larry Friday
- Apply for new passport Monday
- Submit TPS reports Tuesday
- Order 30 Rock DVD
- Prepare presentation
- Get bananas
- Pay electricity bill

Personal (11 tasks)

- 2 due today
- 1 due tomorrow
- 0 overdue
- 10 completed

Key

Priorities: 1 2 3 No

Due today: bold

Overdue: underline

Learn keyboard shortcuts

click!
click!
click!
click!

where's the feedback?

web apps are full of flaws like these

```
if(everything is normal) {  
    provideFeedback();  
} else {} // TODO
```

and the **TODO** is rarely done

FeedLack

FeedLack

project **discussion**

Feedlack found **1** place that appear to be missing feedback:

- ✗ `post(text)` at `index.html` may not produce feedback

Feedlack found **4** places that appear to always produce feedback.

- ✓ `mouseover` at `index.html` `31` always produces output
- ✓ `click` at `index.html` `32` always produces output
- ✓ `keypress` at `index.html` `33` always produces output
- ✓ `mousedown` at `index.html` `34` always produces output

```
7     return comment != '';  
8   }  
9   function post(text) {  
10    if(isValid(text)) {  
11     $.get('comment.php', { comment: text });  
12    }  
13    else {  
14     alert('Your comment is invalid.');15    }  
16  }  
17  </script>  
18  </head>  
19  <body>  
20    ...  
21    <form id='form' onsubmit='post(form.comment.value)'  
22      <input id='comment' type='text' />  
23      <input id='post' onclick='post(form.comment.value)'  
24    </form>  
25    ...  
26  
27
```

post (text) at index.html 9

When the user performs a

- `submit` (`index.html` `21`), or
- `click` (`index.html` `23`)

this path **may fail to produce output**:

1. `post()` is entered `index.html` `9`
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called `index.html` `10`
assumes this calls `isValid(comment)`, because no other functions by this name were found
3. `isValid()` is entered `index.html` `5`
assumes this function can produce output because `text()` can produce output
4. the expression at `index.html` `6` is **false**
5. the expression at `index.html` `10` is **true**
assumes condition can be true
6. several functions are called that **do not affect output**
assumes `get()` (not found) does not affect output
7. `post()` is exited `index.html` `16` **without producing output**

with
Xing Zhang
undergraduate
University of Washington

FeedLack

verifies that
all control flow paths
originating from user input
produce output

for example...

The screenshot displays the FeedLack tool interface. On the left, a sidebar lists several feedback findings: 'Feedback found 1 place that appear to be missing feedback.', 'Feedback found 4 places that appear to always produce feedback.', and three findings related to 'index.html' (click, keypress, mousedown) that 'always produces output'. The main area shows a code editor with PHP and HTML code. The PHP code defines a 'post(text)' function that checks if a comment is valid and either outputs the comment or an error message. The HTML code shows a form with a submit button and a text input. Below the code, a detailed analysis for the 'post(text) of index.html' is shown, listing triggers like 'submit' and 'click', and a sequence of seven steps describing the control flow path from user input to output or no output.

```
7     return comment != ''
8   }
9   function post(text) {
10    if(!isValid(text)) {
11      $.get('comment.php', { comment: text });
12    }
13    else {
14      alert('Your comment is invalid.');
```

post (text) of index.html

When the user performs a

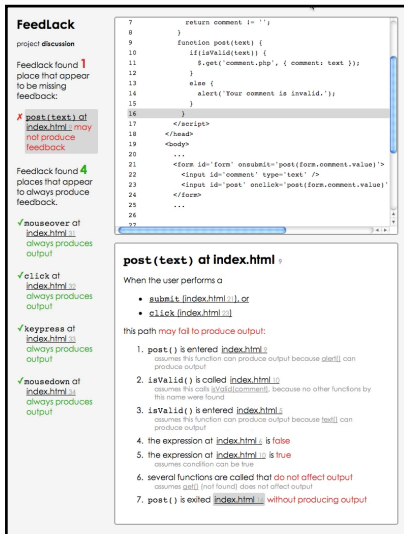
- `submit(index.html)`, or
- `click(index.html)`

This path may fail to produce output:

1. `post()` is entered `index.html`
assumes the function can produce output because `isset()` can produce output
2. `isValid()` is called `index.html`
assumes the call `isset(comment)`, because no other functions by this name were found
3. `isValid()` is entered `index.html`
assumes the function can produce output because `isset()` can produce output
4. the expression of `index.html` is `false`
5. the expression of `index.html` is `true`
assumes condition can be true
6. several functions are called that do not affect output
assumes `get()` that found() does not affect output
7. `post()` is exited `index.html` without producing output

FeedLack

for example...

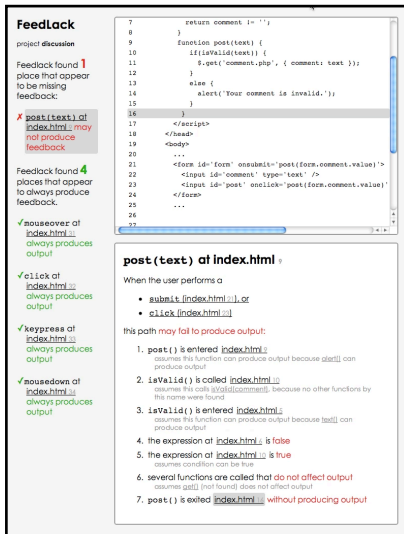


```
<form id='form' onsubmit="post(form.comment.value)">
  <input id='comment' type='text' />
  <input onclick=post(form.comment.value)">
</form>
```

here's a form that posts the value of a comment field when **enter** is typed or **submit** is clicked.

FeedLack

for example...



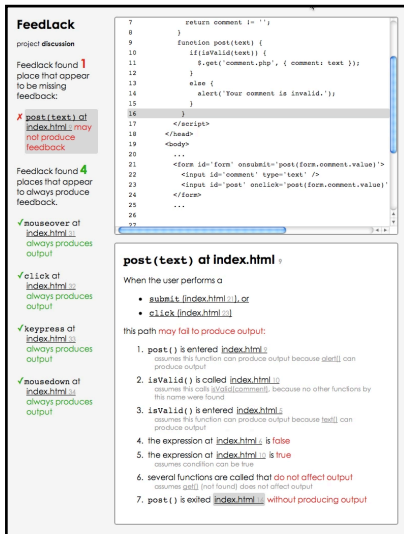
```
<form id='form' onsubmit="post(form.comment.value)">
  <input id='comment' type='text' />
  <input onclick=post(form.comment.value)" />
</form>

<script type='text/javascript'>
  function post(text) {
    if(IsValid(comment))
      $.get("comment.php", { comment: text });
    else
      alert("Your comment is invalid.");
  }
</script>
```

when post() is called, the comment is posted if valid; otherwise, an alert is shown.

FeedLack

for example...



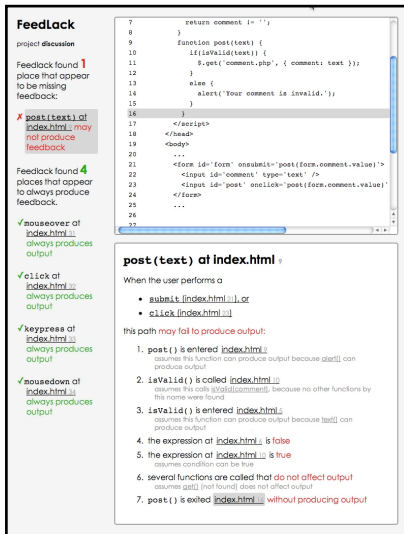
```
<form id='form' onsubmit="post(form.comment.value)">
  <input id='comment' type='text' />
  <input onclick=post(form.comment.value)">
</form>

<script type='text/javascript'>
  function post(text) {
    if(IsValid(comment))
      $.get("comment.php", { comment: text });
    else
      alert("Your comment is invalid.");
  }

  function IsValid(comment) {
    if(comment == '')
      $('#comment').text('write something!');
    return comment != '';
  }
</script>
```

IsValid() provides feedback on empty comments.

FeedLack for example...



```
<form id='form' onsubmit="post(form.comment.value)">
  <input id='comment' type='text' />
  <input onclick=post(form.comment.value)">
</form>

<script type='text/javascript'>
  function post(text) {
    if(IsValid(comment))
      $.get("comment.php", { comment: text });
    else
      alert("Your comment is invalid.");
  }

  function IsValid(comment) {
    if(comment == '')
      $('#comment').text('write something!');
    return comment != '';
  }
</script>
```

what's wrong?

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path may fail to produce output:

1. `post ()` is entered `index.html 9`
Feedback found to events handlers that invoke the same function
2. `isValid (comment)` is entered `index.html 10`
assumes this call `isValid (comment)`, because no other functions by `isValid` were found
3. `isValid ()` is entered `index.html 10`
assumes this function can produce output because `text ()` can produce output
4. the expression at `index.html 6` is `false`
5. the expression at `index.html 10` is `true`
assumes condition can be true
6. several functions are called that `do not affect output`
assumes `get ()` (not found) does not affect output
7. `post ()` is exited `index.html 16` `without producing output`

```
<form id='form' onsubmit="post  
  <input id='comment' type='text'  
  <input onclick=post (form.co  
</form>
```

```
<script type='text/javascript'  
  function post (text) {  
    if (isValid (comment))  
      $.get ("comment.php",  
    else  
      alert ("Your comment  
  }  
  function isValid (comment)  
    if (comment == '')  
      $('#comment').text ('')  
    return comment != '';  
  }  
</script>
```


post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path **may fail to produce output:**

1. `post ()` is entered [index.html 9](#)
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called [index.html 10](#)
assumes this calls `isValid(comment)`, because no other functions by this name were found
3. `isValid()` is entered [index.html 10](#)
assumes this function can produce output because `text()` can produce output
4. the expression at [index.html 6](#) is **false**
5. the expression at [index.html 10](#) is **true**
assumes condition can be true
6. several functions are called that **do not affect output**
assumes `get()` (not found) does not affect output
7. `post ()` is exited [index.html 16](#) **without producing output**

post() handles
the input

```
<form id='form' onsubmit='post  
  <input id='comment' type='text'  
  <input onclick=post(form.co  
</form>  
<script type='text/javascript'  
  function post(text) {  
    if(isValid(comment))  
      $.get("comment.php",  
    else  
      alert("Your comment  
  }  
  function isValid(comment)  
    if(comment == '')  
      $('#comment').text(  
    return comment != '';  
  }  
</script>
```

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path **may fail to produce output:**

1. `post ()` is entered `index.html 9`
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called `index.html 10`
assumes this calls `isValid(comment)`, because no other functions by this name were found
3. `isValid()` is entered `index.html 5`
assumes this function can produce output because `text()` can produce output
4. the expression `isValid(comment)` is true
5. the expression at `index.html 10` is true
assumes `get()` (not found) does not affect output
6. several functions are called that **do not affect output**
assumes `get()` (not found) does not affect output
7. `post ()` is exited `index.html 16` **without producing output**

isValid() might affect input...

```
<form id='form' onsubmit='post
  <input id='comment' type='t
  <input onclick=post(form.co
</form>
<script type='text/javascript'
  function post(text) {
    if (isValid(comment))
      $.get ("comment.php",
    else
      alert ("Your comment
  }
  function isValid(comment)
    if (comment == '')
      $('#comment').text (
    return comment != '';
  }
</script>
```

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path **may fail to produce output:**

1. `post ()` is entered [index.html 9](#)
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called [index.html 10](#)
assumes this calls `isValid(comment)`, because no other functions by this name were found
3. `isValid()` is entered [index.html 5](#)
assumes this function can produce output because `text()` can produce output
4. the expression at [index.html 6](#) is **false**
5. the expression at [index.html 10](#) is **true**
assume `post()` can be found
6. several functions are called that **do not affect output**
assume `post()` (not found) does not affect output
7. `post ()` is exited [index.html 16](#) **without producing output**

isValid() has to be entered to affect input

```
<form id='form' onsubmit='post'  
  <input id='comment' type='text'  
  <input onclick=post(form.co  
</form>  
<script type='text/javascript'  
  function post(text) {  
    if(isValid(comment))  
      $.get("comment.php",  
    else  
      alert("Your comment  
  }  
  function isValid(comment)  
    if(comment == '')  
      $('#comment').text(  
      return comment != '';  
    }  
</script>
```

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path **may fail to produce output:**

1. `post ()` is entered [index.html 9](#)
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called [index.html 10](#)
assumes this calls `isValid(comment)`, because no other functions by this name were found
3. `isValid()` is entered [index.html 5](#)
assumes this function can produce output because `text()` can produce output
4. **the expression at [index.html 6](#) is false**
5. the expression at [index.html 10](#) is true
assumes condition can be true
6. several functions are called that **do not affect output**
assumes `alert()` does not affect output
7. `post ()` is exited [index.html 16](#) **without producing output**

if the
comment is
not empty, it
will skip output

```
<form id='form' onsubmit='post'  
  <input id='comment' type='text'  
  <input onclick=post (form.co  
</form>  
<script type='text/javascript'  
  function post(text) {  
    if(isValid(comment))  
      $.get ("comment.php",  
    else  
      alert ("Your comment  
  }  
  function isValid(comment)  
    if(comment == '')  
      $('#comment').text ('  
    return comment != '';  
  }  
</script>
```

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

if the comment is valid (which it will be, given the previous condition)

1. `isValid()` is entered `index.html 5`
assumes this function can produce output because `text()` can produce output
2. `isValid()` is called `index.html 7`
assumes this calls `isValid(comment)`, because no other functions by this name are entered
3. `isValid()` is entered `index.html 5`
assumes this function can produce output because `text()` can produce output
4. the expression at `index.html 6` is `false`
5. the expression at `index.html 10` is `true`
assumes condition can be true
6. several functions are called that `do not affect output`
assumes `get()` (not found) does not affect output
7. `post()` is exited `index.html 16` `without producing output`

```
<form id='form' onsubmit='post
  <input id='comment' type='t
  <input onclick=post(form.co
</form>
<script type='text/javascript'
  function post(text) {
    if(isValid(comment))
      $.get("comment.php",
    else
      alert("Your comment
  }
  function isValid(comment)
    if(comment == '')
      $('#comment').text(
    return comment != '';
  }
</script>
```

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path **may fail to produce output:**

1. `post ()` is entered `index.html 9`

assumes this function can produce output because `alert()` can

and assuming `$.get()`

2. `isValid()` is called `index.html 10`

produces no output...

3. `isValid()` is entered `index.html 5`

assumes this function can produce output because `text()` can produce output

4. the expression at `index.html 6` is **false**

5. the expression at `index.html 10` is **true**

assumes condition can be true

6. several functions are called that **do not affect output**

assumes `get()` (not found) does not affect output

7. `post ()` is exited `index.html 16` **without producing output**

```
<form id='form' onsubmit='post'  
  <input id='comment' type='text'  
  <input onclick=post(form.co  
</form>
```

```
<script type='text/javascript'  
  function post(text) {  
    if(isValid(comment))  
      $.get("comment.php",  
    else  
      alert("Your comment  
  }  
  function isValid(comment)  
    if(comment == '')  
      $('#comment').text('')  
    return comment != '';  
  }  
</script>
```

post (text) at index.html 9

When the user performs a

- `submit (index.html 21)`, or
- `click (index.html 23)`

this path **may fail to produce output:**

1. `post ()` is entered [index.html 9](#)
assumes this function can produce output because `alert()` can produce output
2. `isValid()` is called [index.html 10](#)
assumes this function can produce output because `alert()` can produce output
3. `isValid()` is entered [index.html 5](#)
assumes this function can produce output because `alert()` can produce output
4. `isValid()` is entered [index.html 6](#) is **false**
5. the expression at [index.html 10](#) is **true**
assumes condition can be true
6. several functions are called that **do not affect output**
assumes `get()` (not found) does not affect output

7. `post ()` is exited [index.html 16](#) **without producing output**

```
<form id='form' onsubmit='post'  
  <input id='comment' type='text'  
  <input onclick=post (form.co  
</form>
```

```
<script type='text/javascript'  
  function post(text) {  
    if(isValid(comment))  
      $.get ("comment.php",  
    else  
      alert ("Your comment  
  }  
  function isValid(comment)  
    if(comment == '')  
      $('#comment').text ('  
    return comment != '';  
  }  
</script>
```

```
<form id='form' onsubmit="post(form.comment.value)">
  <input id='comment' type='text' />
  <input onclick=post(form.comment.value)">
</form>
<script type='text/javascript'>
  function post(text) {
    if(IsValid(comment)) {
      $.get("comment.php", { comment: text })
      .success(function() { alert("submitted!"); })
      .error(function() { alert("didn't work."); })
    }
    else
      alert("Your comment is invalid.");
  }
  function isValid(comment) {
    if(comment == '')
      $('#comment').text('write something!');
    return comment != '';
  }
</script>
```

the obvious
solution is to
add feedback
on success

implementation

ten steps

- 1) identifying and naming functions
- 2) generating function control flow graphs
- 3) propagating type information
- 4) resolving function calls
- 5) identifying output-affecting statements
- 6) identifying input-handling functions
- 7) enumerating paths through input handlers
- 8) expanding paths through input handlers
- 9) Identifying output-lacking paths
- 10) clustering output-lacking paths

implementation

1) identifying and naming functions

only analyze client side JavaScript and HTML

all feedback is ultimately displayed by client

all functions are found

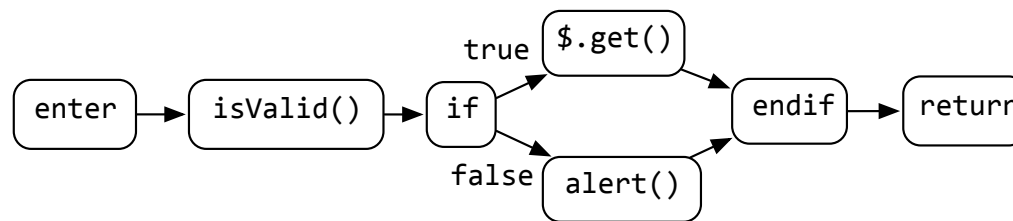
except those generated dynamically

implementation

2) generating function control flow graphs

standard CFGs are created for each function

for example, **post()** from earlier



implementation

3) propagating type information

types of variables and properties are propagated through ASTs from literals, W3C DOM API properties and functions, and object literal declarations

e.g., `document.getElementById()` is assumed to return an `HTMLElement`

implementation

4) resolving function calls

all function calls are resolved using inferred type information

when types aren't available, all functions are searched

to mitigate false positives

apply() and **call()** are assumed to produce output

asynchronous calls are are treated as synchronous

implementation

5) identifying output-affecting statements

any change to the DOM

assignments to W3C DOM properties

e.g., `document.location`, `el.style.top`

jQuery, Prototype, and W3C DOM calls with DOM side effects

e.g., `$(this).hide()`, `el.removeChild()`

implementation

6) identifying input-handling functions

any function directly invoked by W3C input event handlers

includes assignments to properties that represent input handlers

e.g., `el.onclick = goHome`

also includes jQuery and Prototype bindings

e.g., `$(this).click(goHome)`

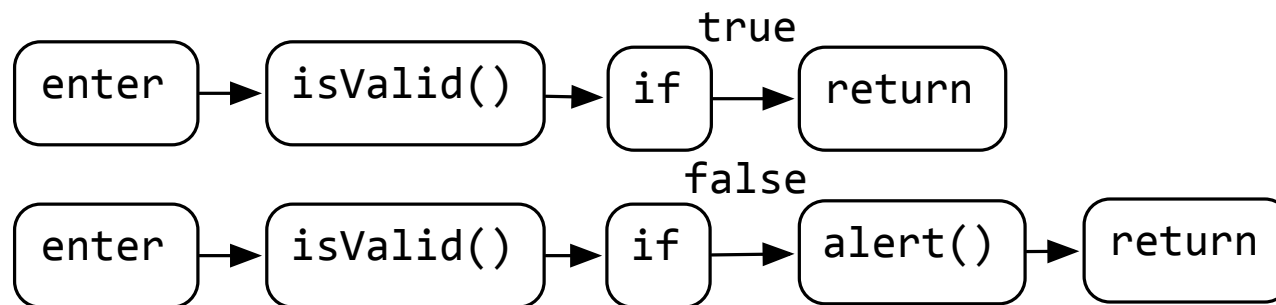
implementation

7) enumerating paths through input handlers

depth-first traversal through each input handler's CFG

only includes calls, returns, conditionals, and output-affecting statements

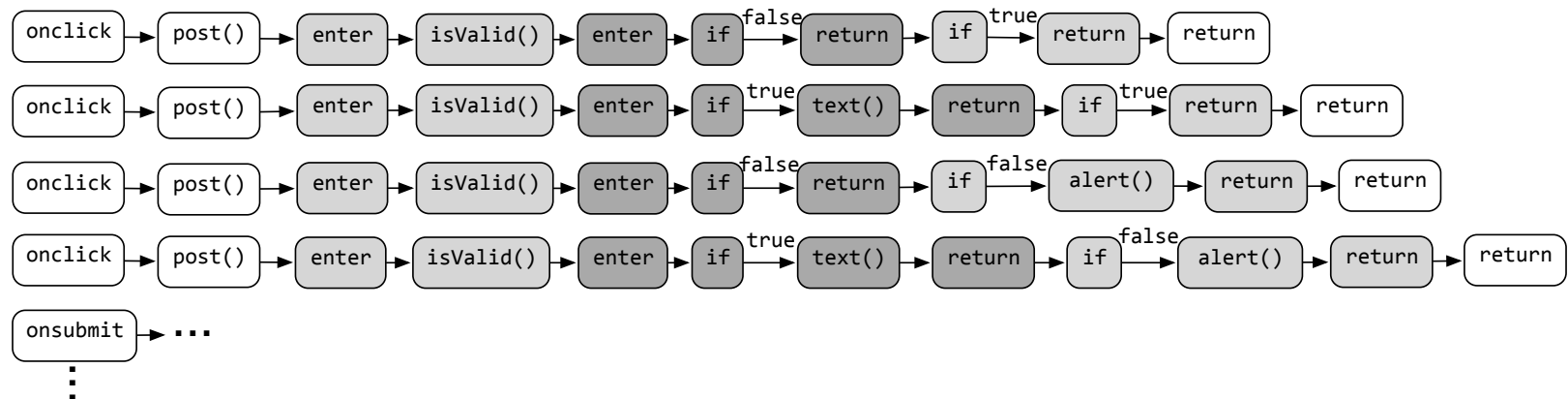
blocks that do not contain output-affecting statement are ignored



implementation

8) expanding paths through input handlers

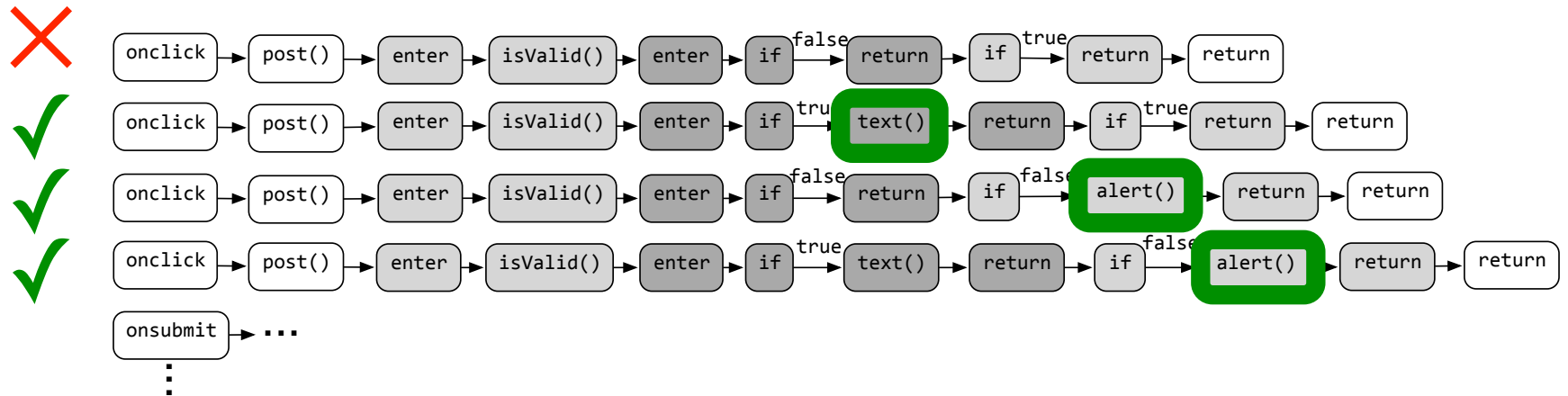
all calls in the resulting paths through input handlers are expanded to all possible resolved functions



implementation

9) Identifying output-lacking paths

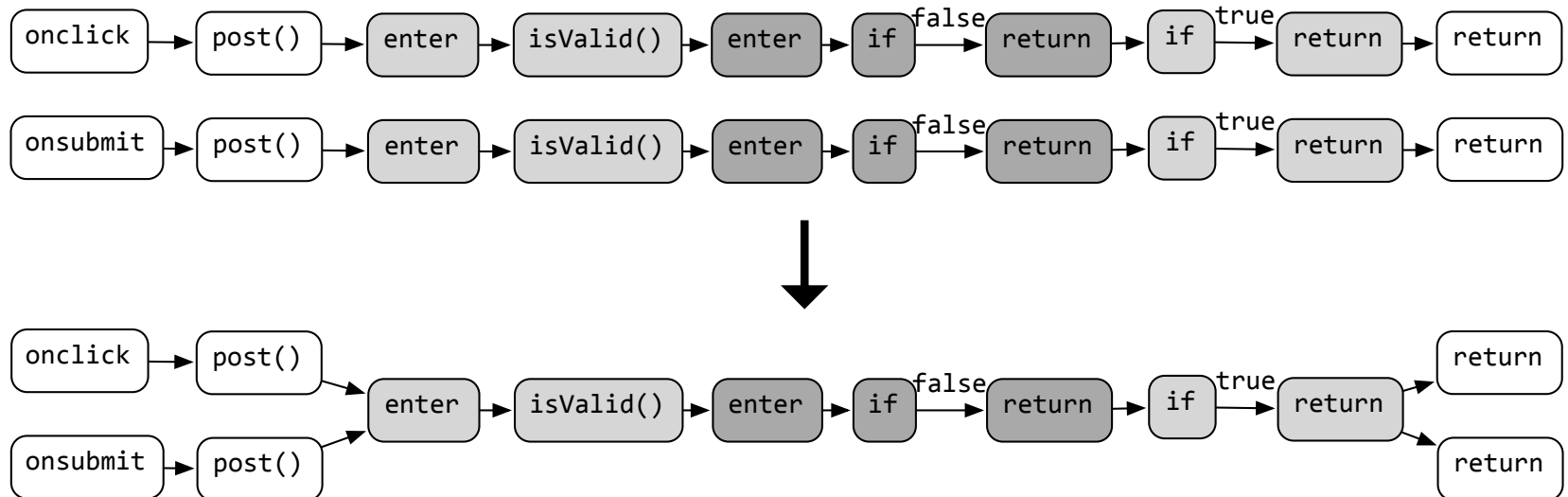
paths lacking an output affecting statement are marked as **output lacking**



implementation

10) clustering output-lacking paths

because handlers often reuse functions that produce output, paths with similar **critical paths** are clustered by identifying largest common subsequences



evaluation

are FeedLack's warnings legitimate?

sampled 129 web application's client-side code

14 failed due to **path explosion**

33/115 applications had no warnings

the 82 remaining had **647 output-lacking paths**

evaluation

classified each of the 647 warnings as one of

12% **infeasible paths**

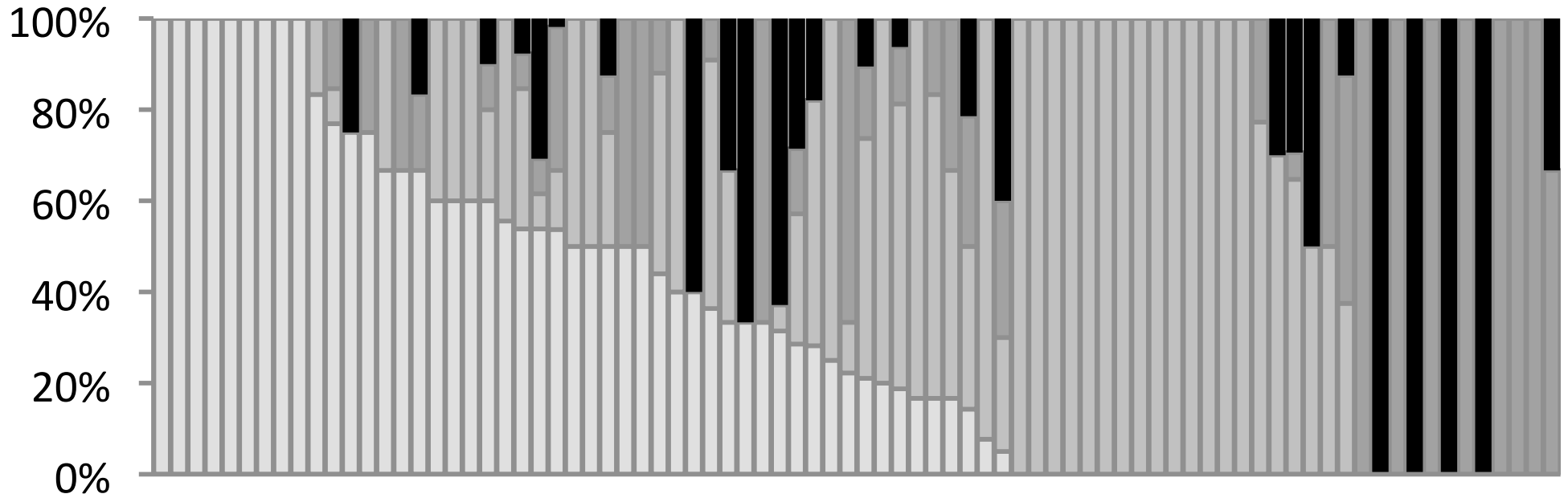
18% **output-producing** false positives

34% **output-missing** true positives that followed standard UI conventions

e.g., buttons that appeared disabled but did not produce feedback

36% **output-deserving** true positives that violated standard UI conventions

deserving missing producing infeasible



proportion of warning types per app

evaluation

how severe were the true positives?

buttons that ignored input in certain modes

text controls that ignored keystrokes

dead links

silent errors

silent success

missing hover feedback

significantly delayed asynchronous feedback

limitations

many false positives

due primarily to **imprecision** in type inference
and call graph construction

many true negatives

paths that produce output that is **imperceptible**

despite all of the **variation** in how web applications are written

there is **uniformity** in developers' mistakes that we can detect and highlight

there is **uniformity** in
developers' mistakes that
we can detect and highlight

developers mistype names

developers overlook execution contexts that
deserve user feedback

**developers rarely comprehend the full extent of
contexts in which their programs execute**

what other details do developers overlook in web development?

control flow paths they've never executed

the full set of dependencies on the code they're changing

silent failure of changes to the DOM

the device an app is being viewed on

the vision impairments of app users

the context in which user interface string literals appear

variations in the meaning of data

user interface dead ends

defect detection for the web



the very languages that **enable**
this flexibility also impose some
~~serious~~ **tradeoffs...**
acceptable

the result may be dynamic
languages that have **some** of
the benefits of static ones

... *without* imposing undue
burden on developers

questions?

Cleanroom

FeedLack

etc.