defect detection for the wayward web

Andrew J. Ko
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

tools

debugging
tools

of debugging

of teamwork

of API learning

of open source

credit to Rob DeLine at MSR

programming tools
<table>
<thead>
<tr>
<th>studies</th>
<th>tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>open bug reporting</td>
<td>next generation help</td>
</tr>
<tr>
<td>bug triage meetings</td>
<td>automating bug severity measurements</td>
</tr>
<tr>
<td>Stack Overflow</td>
<td>improved API documentation</td>
</tr>
<tr>
<td>diagnostic thinking</td>
<td>teaching debugging skills</td>
</tr>
<tr>
<td></td>
<td>defect detection for the web</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
</tr>
</thead>
</table>
| Sat. Sep   | Family Festival Presented by Target  
noon–3 pm  
OSP  
Come out to the Olympic Sculpture Park for the last family festival of our big park season series for summer 2010. The day will be filled with F-U-N, including live music, dancing, story telling, art making, kids tours and more! Plus, learn how you and your family can keep salmon safe and explore how Native American culture encourages us to care for our natural resources. Free and open to the public! |
| Thu. Sep   | Art and the Environment: A Panel Discussion  
11:30 am–1 pm  
SAM, the Cascade Land Conservancy, and the Mayor’s  
>> |
| Sat. Sep   | Educator Workshop  
10:30 am–3:30 pm  
Join us for our annual OSP Educator Workshop! This  
>> |

**PERFORMERS**  
Tilishubdub (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.
**dynamic typing** means that many errors aren't found until runtime.
JavaScript’s flexibility in constructing user interfaces \textit{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

statically detecting a large class of JavaScript errors at edit time

FeedLack

verifying the presence of feedback in response to user input
Cleanroom

with

Jacob Wobbrock
Assistant Professor
The Information School

```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='calculatorBody'>
  <div id='display' class='display'></div>
  <!-- On click, press digit 1 -->
  <button onclick='1'></button>
  <!-- On click, press digit 2 -->
  <button onclick='2'></button>
  <!-- On click, press digit 3 -->
  <button onclick='3'></button>
  <!-- On click, press operation + -->
  <button onclick='+'></button>
  <!-- On click, press digit 4 -->
  <button onclick='4'></button>
  <!-- On click, press digit 5 -->
  <button onclick='5'></button>
</body>
```

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

<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 -->

only after testing was this typo apparent...
current tools do not detect these name errors...

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

\[
\text{frequency}(\text{name}) \propto \text{validity}(\text{name})
\]
the uniqueness heuristic

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

e.g., calculatorBody, console.log()  
how often is this right?  
would warnings based on it be useful?
Cleanroom highlights violations of the uniqueness heuristic after each keystroke.
interaction design

during typing, validation that name isn’t complete

if it’s an error, developer is warned

if it’s an unused variable, developer is reminded

if declared, developer gets confirmation
interaction design

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

The function name `claer` only appears once and doesn't appear. Perhaps you meant `clear`?

alternate names are suggested using Levenstein 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
... 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 that 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 calculator would provide correct answers for:

- clear → 0
- 9 + 5
- 9 – 5
- 9 x 5
- 9 / 5
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

statically detecting a large class of JavaScript errors at edit time

FeedLack

verifying the presence of feedback in response to user input
Cleanroom

statically detecting a large class of JavaScript errors at edit time

FeedLack

verifying the presence of feedback in response to user input
all over the web, apps are ignoring people
all over the web, apps are ignoring people

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

post(text) at index.html

When the user performs a

- submit [index.html 2], or
click [index.html 2]

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 11]
   assumes this function can produce output because isValid() can produce output

4. the expression at [index.html 12] is false

5. the expression at [index.html 13] is true
   assumes condition can be true

6. several functions are called that do not affect output
   assumes post() (not found) does not affect output

7. post() is exited [index.html 14] without producing output

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

with

Xing Zhang
undergraduate
University of Washington
 verifies that all control flow paths originating from user input produce output for example...
for example...

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

for example...

```html
<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...

```html
<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?
FeedLack found to events handlers that invoke the same function

post(text) at index.html

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
2. is_valid() is entered index.html1
   assumes this call of valid(comment), because no other functions by name exist
3. is_valid() is entered index.html4
   as valid() is not found, assumes text() can
4. the expression at index.html 1 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 14 without producing output

<script type='text/javascript'>

function post(text) {
    if(isValid(comment))
        $.get("comment.php")
    else
        alert("Your comment is invalid.");

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

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

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

post() handles the input

<form id='form' onsubmit="post(form)">
  <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")
    else
      alert("Your comment
      function isValid(comment)
      if(comment == '')
        $('#comment').text('',
      return comment != '';
  }
</script>
post(text) at index.html

When the user performs a
- submit(index.html 2), or
- click(index.html 23)

this path may fail to produce output:

1. post() is entered index.html 2
   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

isValid() might
affect input...

3. isValid() is entered index.html 9
   assumes this function can produce output because text() can
   produce output

4. the expression at index.html 10 is true

5. the expression at index.html 10 is true

6. several functions are called that do not affect output
   assumes alert() (not found) does not affect output

7. post() is exited index.html 14 without producing output

<form id='form' onsubmit="post(
<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"
        else
            alert("Your comment
        function isValid(comment)
            if(comment == '')
                $('#comment').text(
                    return comment !="'';
    }
</script>
post(text) at index.html

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 2
   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 4 is false
5. the expression at index.html 10 is true
6. several functions are called that do not affect output
   assume null, not because they are returned empty
7. post() is exited index.html 25 without producing output

isValid() has to be entered to affect input
post(text) at index.html

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 2
   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
   if the
   comment is
   not empty, it
   will skip output
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 function (index.html 9) does not affect output
7. post() is exited index.html 14 without producing output

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

  function isValid(comment)
  if (comment == '')
    $('#comment').text('return comment != "';

</script>

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

5. the expression at index.html is true

6. several functions are called that do not affect output

7. post() is exited index.html without producing output
post (text) at index.html

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 2
   assumes this function can produce output because alert() can

2. isValid() is called index.html 10
   assumes this function can produce output because $.get() can
   produce output by
3. isValid() is entered index.html 5
   assumes this function can produce output because text() can
   produce output
4. the expression at index.html 4 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

and assuming $.get() produces no output...

<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>
the input handler will exit without producing feedback

7. post() is exited without producing output
the obvious solution is to add feedback on success
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 treated as synchronous
implementation

5) identifying output-affecting statements

output-affecting statements include 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

![Diagram showing paths lacking output](image-url)
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

```plaintext
onclick ➔ post() ➔ enter ➔ isValid() ➔ enter ➔ if false ➔ return ➔ if true ➔ return ➔ return

onsubmit ➔ post() ➔ enter ➔ isValid() ➔ enter ➔ if false ➔ return ➔ if true ➔ return ➔ return

.onclick ➔ post() ➔ enter ➔ isValid() ➔ enter ➔ if false ➔ return ➔ if true ➔ return ➔ return

.onsubmit ➔ post() ➔ enter ➔ isValid() ➔ enter ➔ if false ➔ return ➔ if true ➔ return ➔ return
```
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
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
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...

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

...without imposing undue burden on developers
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
e tc.