Cleanroom
Edit-Time Error Detection with the Uniqueness Heuristic

Andrew J. Ko and Jacob O. Wobbrock
the web is great for rapid prototyping ...
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>
only after testing was this typo apparent...
Come out to the Olympic Sculpture Park for the last family event of the season!

**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.

**Community Partners:** Duwamish Tribe Long House & Cultural Center, United Indians of All Tribes, Daybreak Star Cultural Center, Pathways to Prosperity, Native People for Cancer Control, Red Eagle Soaring, Washington Rare Plant Care & Conservation, Restore Our Waters, City of Seattle Public Utilities, Burke Museum of Natural History, Colorific Kids, Youth in Focus

Free and open to the public.
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...
the uniqueness heuristic

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

how often is this right?

would warnings based on it be useful?

Need some examples of two name sequences
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

```
page.lastElement = |
```

if it’s an unused variable, developer is reminded

```
page.lastElement = |
```

if declared, developer gets confirmation

```
page.lastElement = |
```
interaction design

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

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
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 calculator would provide correct answers for:

- clear $\rightarrow$ 0
- $9 + 5$
- $9 - 5$
- $9 \times 5$
- $9 / 5$
the participants

94 visitors, 40 start 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

response to “In the past month, I’ve written JavaScript...” was 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
both conditions fixed a similar proportion of warned names \( (p > .05, \text{ ns}) \)
results

Cleanroom developers were significantly more likely to **fixed warned code explicitly** \((p < .001)\)

![Bar chart showing proportion of explicit corrections]

- **Cleanroom**: 100%
- **Control**: 25%

Andrew J. Ko – VL/HCC 2010 – Cleanroom
results

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

Cleanroom developers executed warned names significantly fewer times

median warning executions

- 8 executions
- 6 executions
- 4 executions
- 2 executions
- 0 executions

Cleanroom | control
errors that Cleanroom developers fixed

unddeclared names

unused names

typos (e.g., `parseFLoat`, `getElementByID`, `onlcick`, `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

highlights names and pairs of names that appear only once

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