# DLint: Dynamically Checking Bad Coding Practices in JavaScript

Liang Gong[1], Michael Pradel[2], Manu Sridharan[3] and Koushik Sen[1]

[1] UC Berkeley [2] TU Darmstadt [3] Samsung Research America

# Why JavaScript?

- The RedMonk Programming Language Rankings (***1st***)
  - Based on GitHub and StackOverflow
- Web assembly language
- Web applications, DSL, Desktop App, Mobile App

NativeScript

emscripten

CoffeeScript

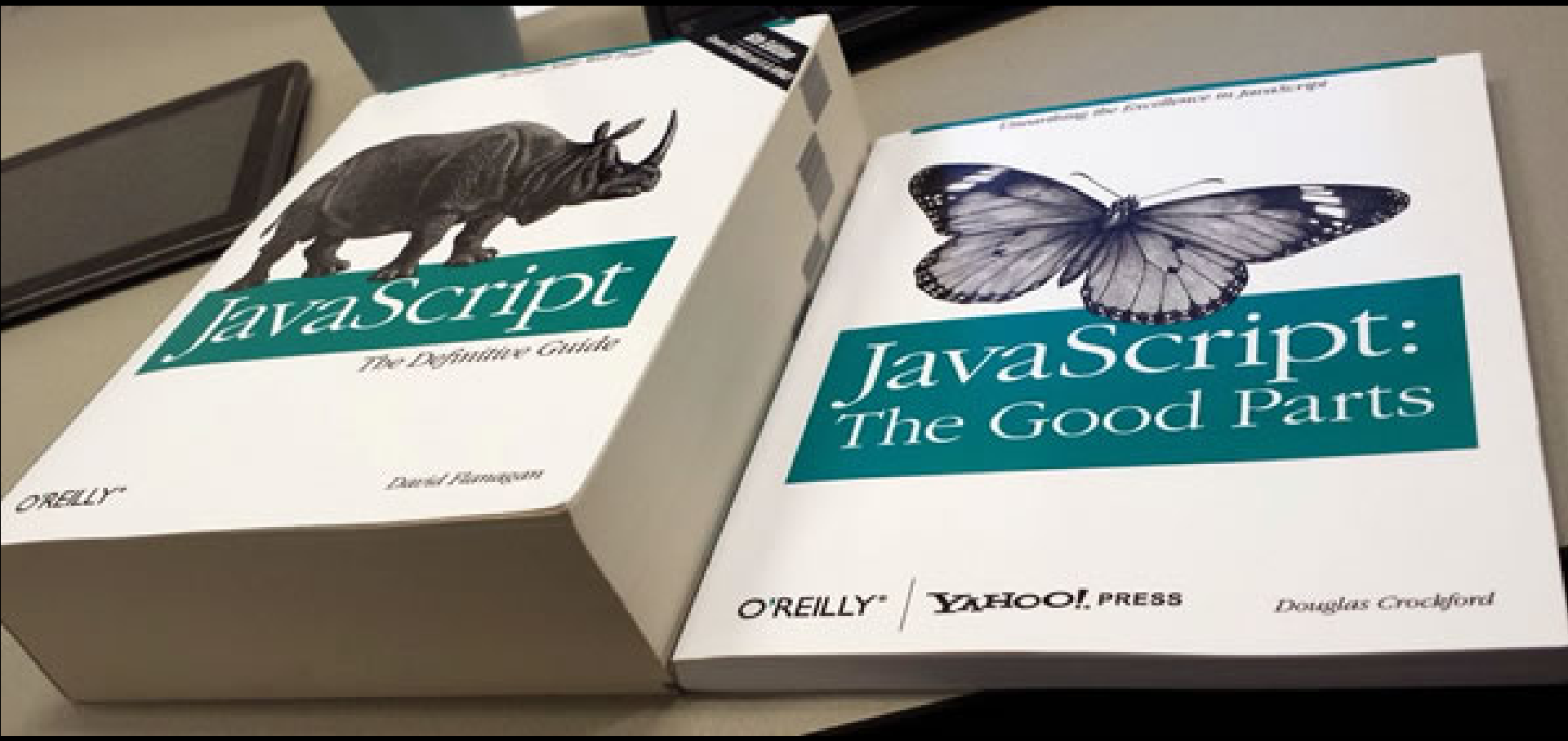
TypeScript

...

# Problematic JavaScript

- Designed and Implemented in 10 days →
- Not all decisions were well thought →
- Problematic language features
  - Error prone
  - Inefficient code
  - Security loophole
- Problematic features are retained
  - backward compatibility

# Problematic JavaScript



# What is coding practice?

- Good coding practices
  - informal rules
  - improve quality

- Better quality means:
  - Less correctness issues
  - Better performance
  - Better usability
  - Better maintainability
  - Less security loopholes
  - Less surprises
  - …

# Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

# Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

array index
(not array value)

```
11 + 22 + 33 => 66
```

array index : string

```
0 + 1 + 2 => 3
```

```
0+"0"+"1"+"2" => "0012"
```

# Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

array index
(not array value)

```
11 + 22 + 33 => 66
```

```
0 + 1 + 2 => 3
```

array index : string

```
0+"0"+"1"+"2" => "0012"
```

- Cross-browser issues 

`> "0012indexOftoString..."`
- Result depends on the Array prototype object

# Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

```
for (i=0; i < array.length; i++) {
    sum += array[i];
}
```

```
function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```

# Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

```
for (i=0; i < array.length; i++) {
    sum += array[i];
}
```

```
function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```

# Coding Practices and Lint Tools

- Existing Lint-like checkers
  - Inspect source code
  - Rule-based checking
  - Detect common mistakes
  - Enforce coding conventions
- Limitations:
  - Approximates behavior
  - Unknown aliases
  - Lint tools favor precision over soundness
- Difficulty: Precise static program analysis

# DLint

- Dynamic Linter checking code quality rules for JS
- Open-source, robust and extensible framework
- Formalized and implemented 28 rules
  - Counterparts of static rules
  - Additional rules
- Empirical study
  - Compare static and dynamic checking

# Jalangi: A Dynamic Analysis Framework for JavaScript

Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs

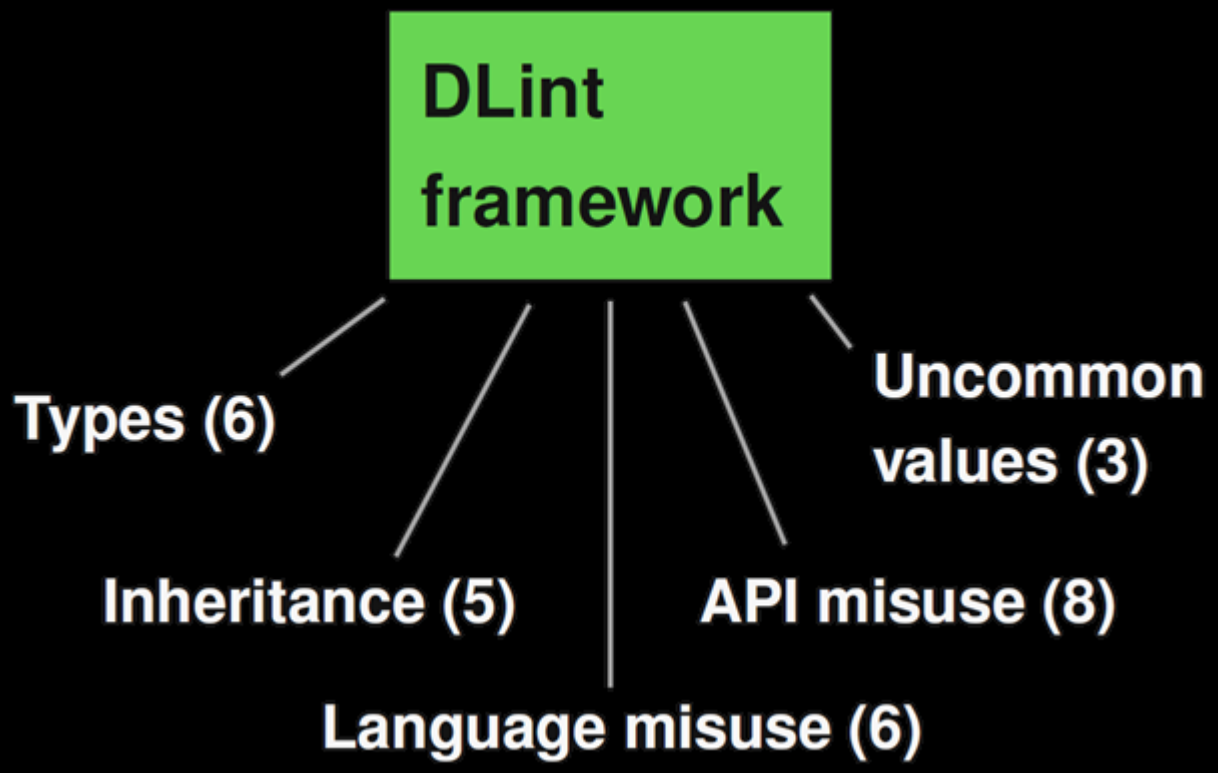
```
a.f = b.g          ➡ PutField(Read("a", a), "f", GetField(Read("b", b), "g"))

if (a.f()) ...     ➡ if (Branch(Method(Read("a", a), "f")()) ...

x = y + 1          ➡ x = Write("x", Binary('+',Read("y", y), Literal(1))
```

---

```
analysis.Literal(c)                      analysis.Branch(c)
analysis.Read(n, x)                      analysis.Write(n, x)
analysis.PutField(b, f, v)               analysis.Binary(op, x, y)
analysis.Function(f, isConstructor)      analysis.GetField(b,f)
analysis.Method(b, f, isConstructor)     analysis.Unary(op, x)
...
```

# Runtime Patterns

- Single-event: Stateless checking
- Multi-event: Stateful checking



# Language Misuse

Avoid setting properties of primitives, which has no effect.

```
var fact = 42;
fact.isTheAnswer = true;
console.log(fact.isTheAnswer);

> undefined
```

DLint Checker Predicate:

$$propWrite(base, *, *) \wedge isPrim(base)$$

# Uncommon Values

Avoid producing NaN (Not a Number).

```
var x = 23 - "five";

> NaN
```

DLint Checker Predicate:

$$unOp(*, val, NaN) \wedge val \neq NaN$$

$$binOp(*, left, right, NaN) \wedge left \neq NaN \wedge right \neq NaN$$

$$call(*, *, args, NaN, *) \wedge NaN \neq args$$

# Uncommon Values

Avoid concatenating undefined to string.

```
var value;
...
var str = "price: ";
...
var result = str + value;

> "price: undefined"
```

DLint Checker Predicate:

$$binOp(+, left, right, res)$$
$$\wedge\ (left = undefined \vee right = undefined)$$
$$\wedge\ isString(res)$$

# API Misuse

Beware that all wrapped primitives coerce to true.

```
var b = false;
if (new Boolean(b)) {
  console.log("true");
}

> true
```

DLint Checker Predicate:

$$cond(val) \wedge isWrappedBoolean(val) \wedge val.valueOf() = false$$

**Table 1: Inheritance-related code quality rules and runtime patterns (all are single-event patterns).**

| ID | Name | Code quality rule | Runtime event predicate(s) |
|---|---|---|---|
| I1 | Enumerable-ObjProps | Avoid adding enumerable properties to `Object`. Doing so affects every for-in loop. | $propWrite(Object, *, *)$<br>$call(Object, f, args, *, *) \mid f.name = \text{“}defineProperty\text{”} \ \wedge$<br>$args.length = 3 \ \wedge \ args[2].enumerable = true$ |
| I2 | Inconsistent-Constructor | `x.constructor` should yield the function that has created `x`. | $propRead(base, constructor, val) \mid val \neq$ function that has created $base$ |
| I3 | NonObject-Prototype | The prototype of an object must be an object. | $propWrite(*, name, val) \mid name \in \{\text{“}prototype\text{”}, \text{“}\_\_proto\_\_\text{”}\} \ \wedge$<br>$!isObject(val)$ |
| I4 | Overwrite-Prototype | Avoid overwriting an existing prototype, as it may break the assumptions of other code. | $propWrite(base, name, *) \mid name \in \{\text{“}prototype\text{”}, \text{“}\_\_proto\_\_\text{”}\} \ \wedge$<br>$base.name$ is a user-defined prototype before the write |
| I5 | Shadow-ProtoProp | Avoid shadowing a prototype property with an object property. | $propWrite(base, name, val) \mid val$ is defined in $base's$ prototype chain $\wedge$<br>$!isFct(val) \ \wedge \ (base, name) \notin shadowingAllowed$ |

**Table 2: Code quality rules and runtime patterns related to type errors.**

| ID | Name | Code quality rule | Runtime event predicate(s) |
|---|---|---|---|
| *Single-event patterns:* | | | |
| T1 | FunctionVs-Prim | Avoid comparing a function with a primitive. | $binOp(relOrEqOp, left, right, *) \mid isFct(left) \ \wedge \ isPrim(right)$<br>$binOp(relOrEqOp, left, right, *) \mid isPrim(left) \ \wedge \ isFct(right)$ |
| T2 | StringAnd-Undefined | Avoid concatenating a string and `undefined`, which leads to a string containing “undefined”. | $binOp(+, left, right, res) \mid (left = \text{“}undefined\text{”} \ \vee \ right = \text{“}undefined\text{”}) \ \wedge$<br>$isString(res)$ |
| T3 | ToString | `toString` must return a string. | $call(*, f, *, ret, *) \mid f.name = \text{“}toString\text{”} \ \wedge \ !isString(ret)$ |
| T4 | Undefined-Prop | Avoid accessing the “undefined” property. | $propWrite(*, \text{“}undefined\text{”}, *)$<br>$propRead(*, \text{“}undefined\text{”}, *)$ |
| *Multi-event patterns:* | | | |
| T5 | Constructor-Functions | Avoid using a function both as constructor and as non-constructor. | $call(*, f, *, *, false) \ \wedge \ call(*, f, *, *, true)$ |
| T6 | TooMany-Arguments | Pass at most as many arguments to a function as it expects. | $call(*, f, args, *, *) \mid \lvert args \rvert > f.length \ \wedge$<br>$\nexists \ varRead(arguments, *)$ during the call |

**Table 4: Code quality rules and runtime patterns related to incorrect API usage (single-event patterns).**

| ID | Name | Code quality rule | Runtime event predicate(s) |
|---|---|---|---|
| A1 | Double-Evaluation | Avoid `eval` and other ways of runtime code injection. | $call(builtin, eval, *, *, *)$<br>$call(builtin, Function, *, *, *)$<br>$call(builtin, setTimeout, args, *, *) \mid isString(args[0])$<br>$call(builtin, setInterval, args, *, *) \mid isString(args[0])$<br>$call(document, f, *, *, *) \mid f.name = \text{“}write\text{”}$ |
| A2 | EmptyChar-Class | Avoid using an empty character class, `[]`, in regular expressions, as it does not match anything. | $lit(val) \mid isRegExp(val) \ \wedge \ val$ contains “[]”<br>$call(builtin, RegExp, args, *, *) \mid isString(args[0]) \ \wedge \ args[0]$ contains “[]” |
| A3 | FunctionTo-String | Avoid calling `Function.toString()`, whose behavior is platform-dependent. | $call(base, f, *, *, *) \mid f.name = \text{“}toString\text{”} \ \wedge \ isFct(base)$ |
| A4 | FutileWrite | Writing a property should change the property's value. | $propWrite(base, name, val) \mid base[name] \neq val$ after the write |
| A5 | MissingRadix | Pass a radix parameter to `parseInt`, whose behavior is platform-dependent otherwise. | $call(builtin, parseInt, args, *, *) \mid args.length = 1$ |
| A6 | SpacesIn-Regexp | Prefer “ `{N}`”2 over multiple consecutive empty spaces in regular expressions for readability. | $lit(val) \mid isRegExp(val) \ \wedge \ val$ contains “ ”<br>$call(builtin, RegExp, args, *, *) \mid args[0]$ contains “ ” |
| A7 | StyleMisuse | CSS objects are not strings and should not be used as if they were. | $binOp(eqOp, left, right) \mid isCSSObj(left) \ \wedge \ isString(right)$<br>$binOp(eqOp, left, right) \mid isString(left) \ \wedge \ isCSSObj(right)$ |
| A8 | Wrapped-Primitives | Beware that all wrapped primitives coerce to `true`. | $cond(val) \mid isBooleanObj(val) \ \wedge \ val.valueOf() = false$ |

**Table 3: Code quality rules and runtime patterns related to language misuse (all are single-event patterns).**

| ID | Name | Code quality rule | Runtime event predicate(s) |
|---|---|---|---|
| L1 | Arguments-Variable | Avoid accessing non-existing properties of `arguments`. | $propRead(arguments, name, *) \mid name \notin argumentProps$<br>$propWrite(arguments, *, *)$<br>$call(arguments, f, *, *, *) \mid f.name = \text{“}concat\text{”}$ |
| L2 | ForInArray | Avoid for-in loops over arrays, both for efficiency and because it may include properties of `Array.prototype`. | $forIn(val) \mid isArray(val)$ |
| L3 | GlobalThis | Avoid referring to `this` when it equals to `global`. | $varRead(this, global)$ |
| L4 | Literals | Use literals instead of `new Object()` and `new Array()`[1] | $call(builtin, f, args, *, *) \mid (f = Array \ \vee \ f = Object) \ \wedge \ args.length = 0$ |
| L5 | NonNumeric-ArrayProp | Avoid storing non-numeric properties in an array. | $(propWrite(base, name, *) \ \vee \ propRead(base, name, *)) \mid isArray(base) \ \wedge$<br>$!isNumeric(name) \ \wedge \ name \notin arrayProps)$ |
| L6 | PropOf-Primitive | Avoid setting properties of primitives, which has no effect. | $propWrite(base, *, *) \mid isPrim(base)$ |

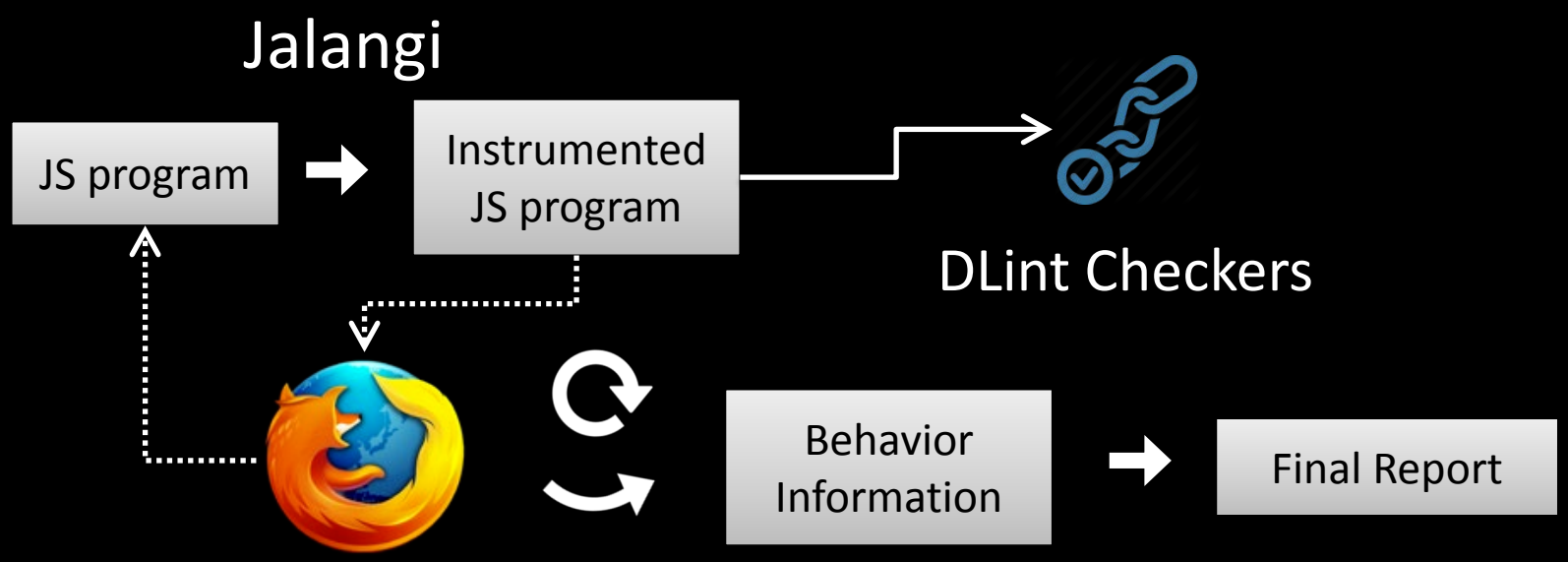
[1] Note that it is legitimate for performance reasons to call these constructors with arguments [24].

**Table 5: Code quality rules and runtime patterns related to uncommon values (all are single-event patterns).**

| ID | Name | Code quality rule | Runtime event predicate(s) |
|---|---|---|---|
| V1 | Float-Equality | Avoid checking the equality of similar floating point numbers, as it may lead to surprises due to rounding.[2] | $binOp(eqOp, left, right, *) \mid isFloat(left) \ \wedge \ isFloat(right) \ \wedge$<br>$\lvert left - right \rvert < \epsilon$ |
| V2 | NaN | Avoid producing `NaN` (not a number). | $unOp(*, val, NaN) \mid val \neq NaN$<br>$binOp(*, left, right, NaN) \mid left \neq NaN \ \wedge \ right \neq NaN$<br>$call(*, *, args, NaN, *) \mid NaN \notin args$ |
| V3 | Overflow-Underflow | Avoid numeric overflow and underflow. | $unOp(*, val, \infty) \mid val \neq \infty$<br>$binOp(*, left, right, \infty) \mid left \neq \infty \ \wedge \ right \neq \infty$<br>$call(builtin, *, args, \infty, *) \mid \infty \notin args$ |

[2] It is a notorious fact that the expression `0.1 + 0.2 === 0.3` returns `false` in JavaScript.

# DLint Overview




- Instrument SpiderMonkey to intercept JavaScript files
- Transpile JavaScript code with Jalangi [Sen et al. FSE 2013]
- DLint checks runtime states and find issues
- Report reason and code location

# Evaluation

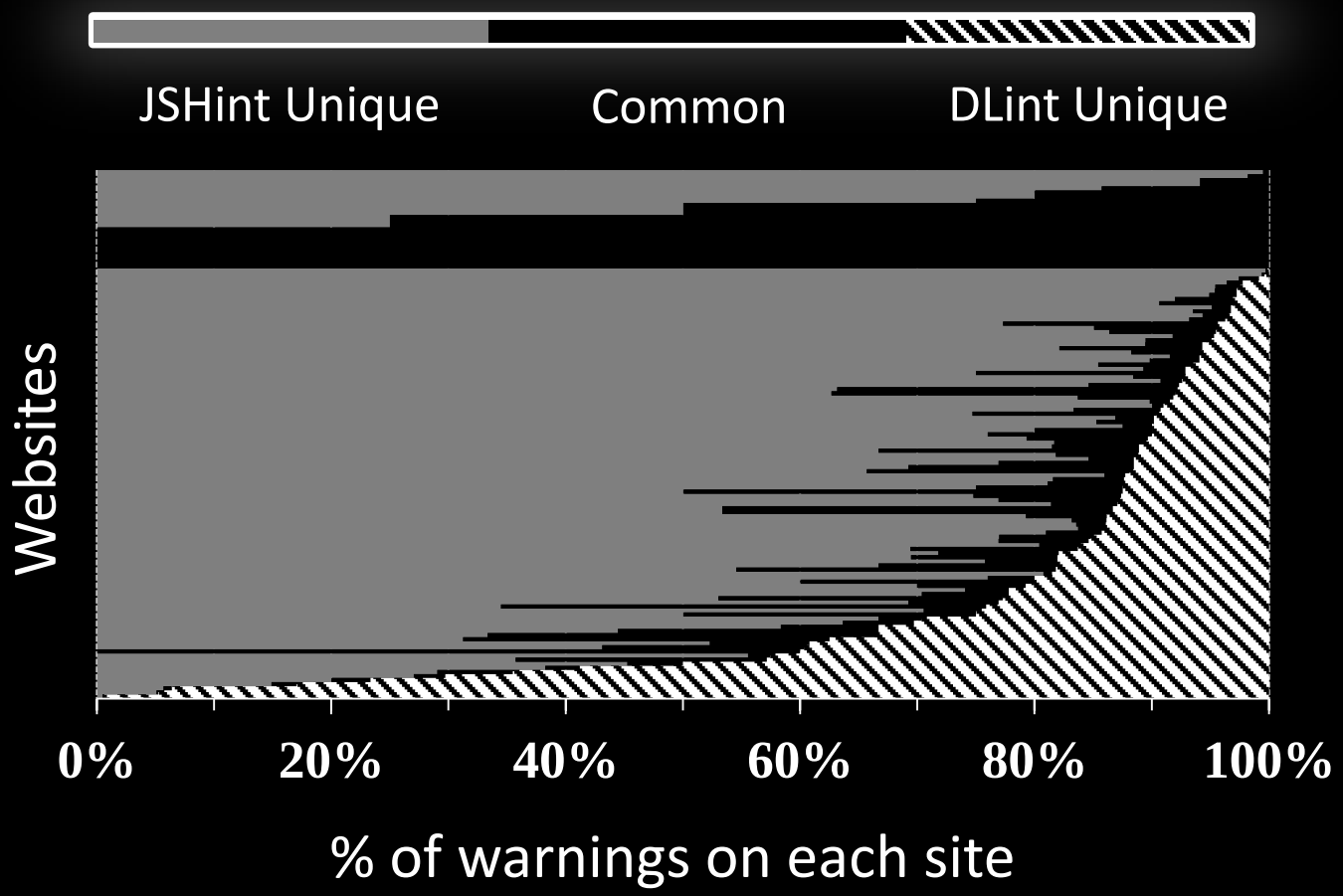
## Research Questions

- DLint warning vs. JSHint warning?
- Additional warnings from DLint?
- Coding convention vs. page popularity?

## Experimental Setup

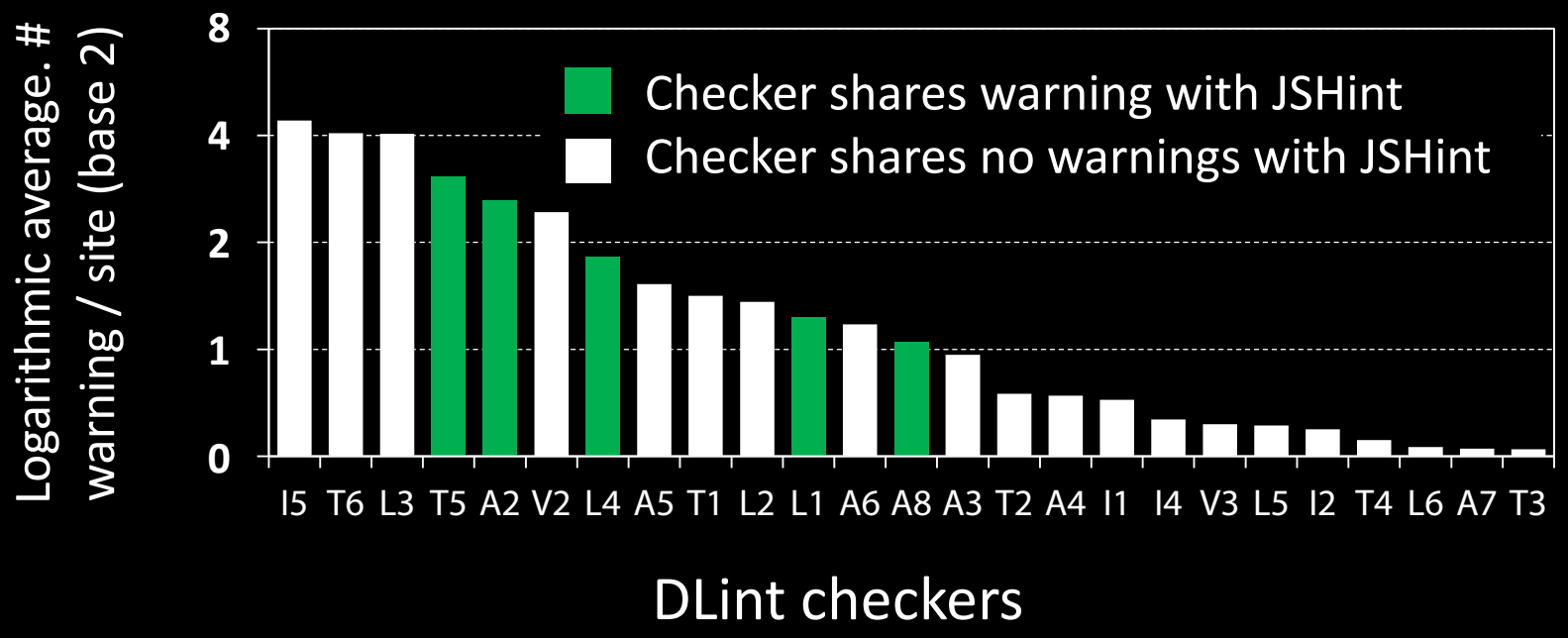
- 200 web sites (top 50 + others)
- Comparison to JSHint

# % of Warnings: DLint vs. JSHint



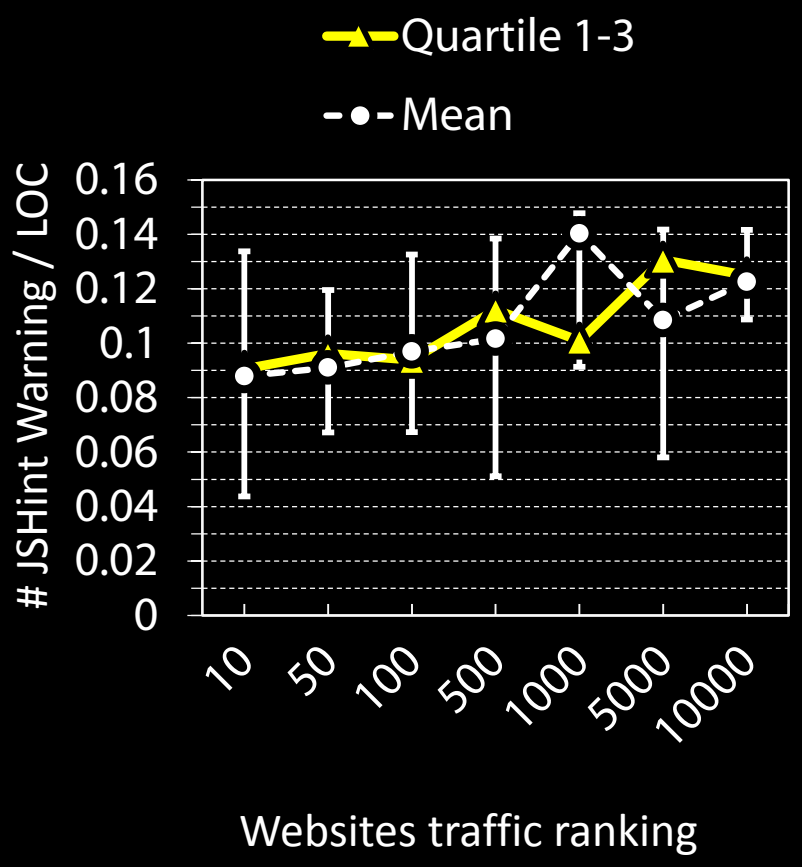

- Some sites: One approach finds all
- Most sites: Better together

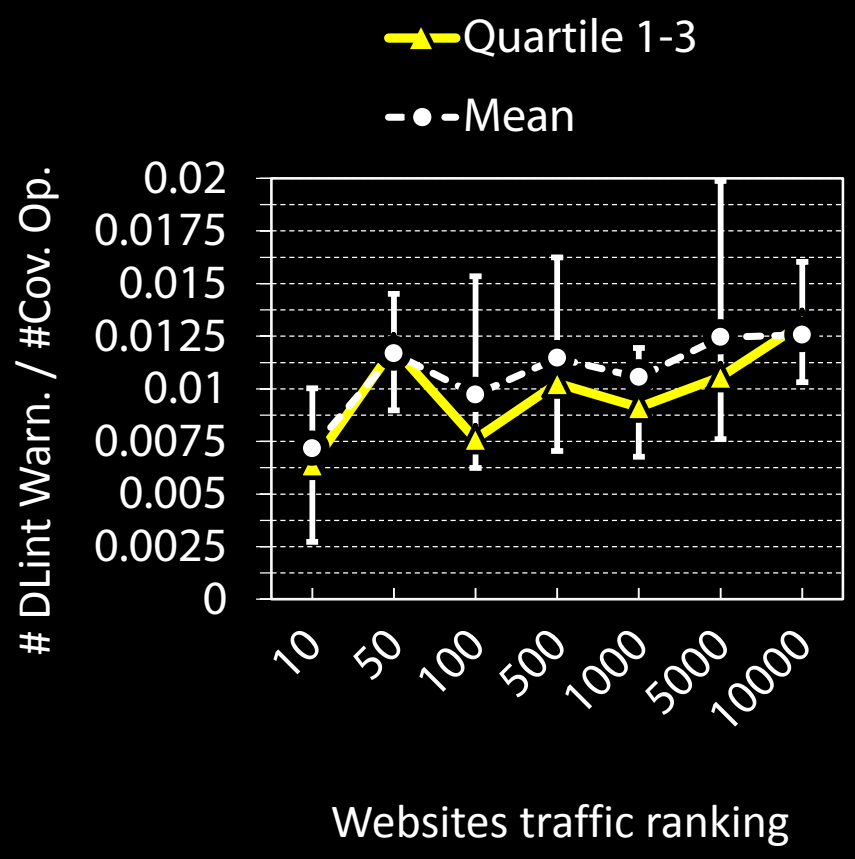
# Additional Warnings Reported by DLint




- 53 warnings per page
- 49 are missed by JSHint

# Coding Convention vs. Page Popularity




Correlation between Alexa popularity and number of DLint warnings: 0.6

Hilton
NIAGARA FALLS / FALLSVIEW
HOTEL & SUITES

Please Select Arrival Date | 2 Please Select Departure Date

Next Month >>

December

| SUNDAY | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY | SATURDAY |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 Starting At $undefinec | 26 Starting At $129 | 27 Starting At $169 |
| 28 Starting At $149 | 29 Starting At $149 | 30 Starting At $129 | 31 Starting At $499 | | | |

TOURS | AMENITIES | ATTRACTIONS | US VISITORS | SHUTTLE SERVICE | FIREWORKS & ILLUMINATION | GROUP SALES

WINNER CERTIFICATE OF EXCELLENCE 2014 tripadvisor

Expedia Insiders' Select 2014

CAA Four Diamond Award AAA

HILTON HHONORS

Hilton
NIAGARA FALLS / FALLSVIEW
HOTEL & SUITES

Please Select Arrival Date | 2 Please Select Departure Date

Next Month >>

December

| SUNDAY | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY | SATURDAY |
|---|---|---|---|---|---|---|
| | | | 3 | 4 | 5 | 6 |
| 7 | | | 10 | 11 | 12 | 13 |
| 14 | | | 17 | 18 | 19 | 20 |
| 21 | | | 24 | 25 Starting At $undefinec | 26 Starting At $129 | 27 Starting At $169 |
| 28 Starting At $149 | 29 Starting At $149 | 30 Starting At $129 | 31 Starting At $499 | | | |

25
Starting At
$undefinec

TOURS | AMENITIES | ATTRACTIONS | US VISITORS | SHUTTLE SERVICE | FIREWORKS & ILLUMINATION | GROUP SALES

WINNER CERTIFICATE OF EXCELLENCE 2014 tripadvisor

Expedia Insiders' Select 2014

CAA Four Diamond Award AAA

HILTON HHONORS

**ALGOT**
Frame/6 wire baskets/top shelf
**$64**
**Last year's price: $NaN**

**ALGOT**
Frame/wire baskets/rod
**$204**
**Last year's price: $220**

**ALGOT**
Frame/4 wire baskets/top shelf
**$44**
**Last year's price: $48**

**ALGOT**
Frame/4 wire baskets/top shelf
**$60**
**Last year's price: $NaN**

**ALGOT**
Frame/4 wire baskets
**$35**
**Last year's price: $39**

**ALGOT**
Frame/4 wire baskets
**$51**
**Last year's price: $NaN**

**ALGOT**
Frame/4 mesh baskets/top shelf
**$56**
**Last year's price: $60**

**ALGOT**
Frame with rod/wire baskets
**$74**
**Last year's price: $87**

**ALGOT**
Frame/6 wire baskets/top shelf
**$64**
**Last year's price: $NaN**

**ALGOT**
Frame/wire baskets/rod
**$204**

**ALGOT**
Frame/4 wire baskets/top shelf
**$44**

**ALGOT**
Frame/4 wire baskets/top shelf
**$60**
**Last year's price: $NaN**

**ALGOT**
Frame/4 wire baskets/top shelf
**$60**
**Last year's price: $NaN**

**ALGOT**
Frame/4 wire baskets
**$35**
**Last year's price: $39**

**ALGOT**
Frame/4 wire baskets
**$51**
**Last year's price: $NaN**

**ALGOT**
Frame/4 mesh baskets/top shelf
**$56**
**Last year's price: $60**

**ALGOT**
Frame with rod/wire baskets
**$74**
**Last year's price: $87**

# Rule: avoid setting field on primitive values

From Google Octane Game Boy Emulator benchmark:

```
var decode64 = "";
if (dataLength > 3 && dataLength % 4 == 0) {
  while (index < dataLength) {
    decode64 += String.fromCharCode(...);
  }
  if (sixbits[3] >= 0x40) {
    decode64.length -= 1;
  }
}
```

# Rule: avoid setting field on primitive values

From Google Octane Game Boy Emulator benchmark:

```
var decode64 = "";
if (dataLength > 3 && dataLength % 4 == 0) {
  while (index < dataLength) {
    decode64 += String.fromCharCode(...);
  }
  if (sixbits[3] >= 0x40) {
⚠️  decode64.length -= 1;
  }
}
```

No effect because ***decode64*** is a primitive string.

# Rule: avoid no effect operations


```
window.onbeforeunload=
"Twitch.player.getPlayer().pauseVideo();"
```

```
window.onunload=
"Twitch.player.getPlayer().pauseVideo();"
```

# Rule: avoid no effect operations


⚠ `window.onbeforeunload= "Twitch.player.getPlayer().pauseVideo();"`

⚠ `window.onunload= "Twitch.player.getPlayer().pauseVideo();"`

```
window.onbeforeunload = function () {
    Twitch.player.getPlayer().pauseVideo();
}
```

# Takeaways

Dynamic lint-like checking for JavaScript

- Static checkers are not sufficient, DLint complements
- DLint is a open-source, robust and extensible tool
  - Works on real-world websites
  - Found 19 clear bugs on most popular websites

More information:

- Paper: "DLint: Dynamically Checking Bad Coding Practices in JavaScript"
- Source Code: https://github.com/Berkeley-Correctness-Group/DLint
- Google "DLint Berkeley"

# Takeaways

Dynamic lint-like checking for JavaScript

- Static checkers are not sufficient, DLint complements
- DLint is a open-source, robust and extensible tool
  - Works on real-world websites
  - Found 19 clear bugs on most popular websites

More information:

- Paper: "DLint: Dynamically Checking Bad Coding Practices in JavaScript"
- Source Code: https://github.com/Berkeley-Correctness-Group/DLint
- Google "DLint Berkeley"

Thanks!

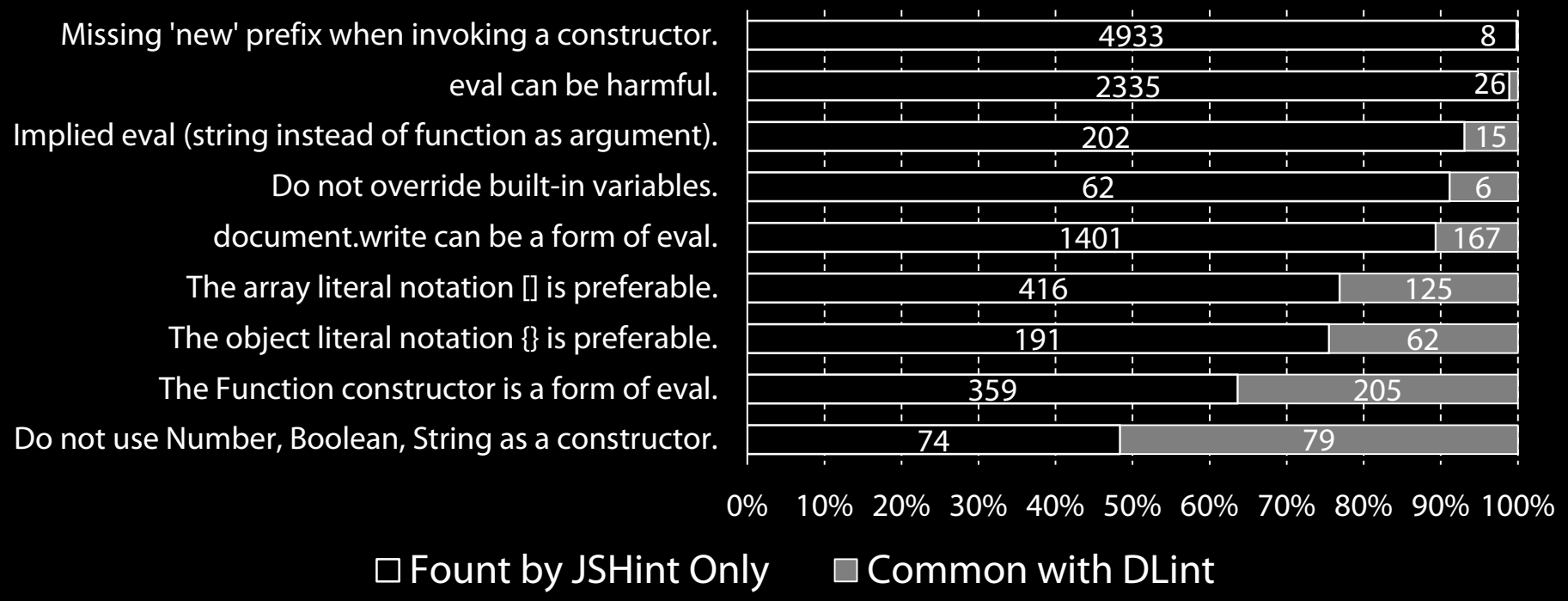
# Formalization: declarative specification

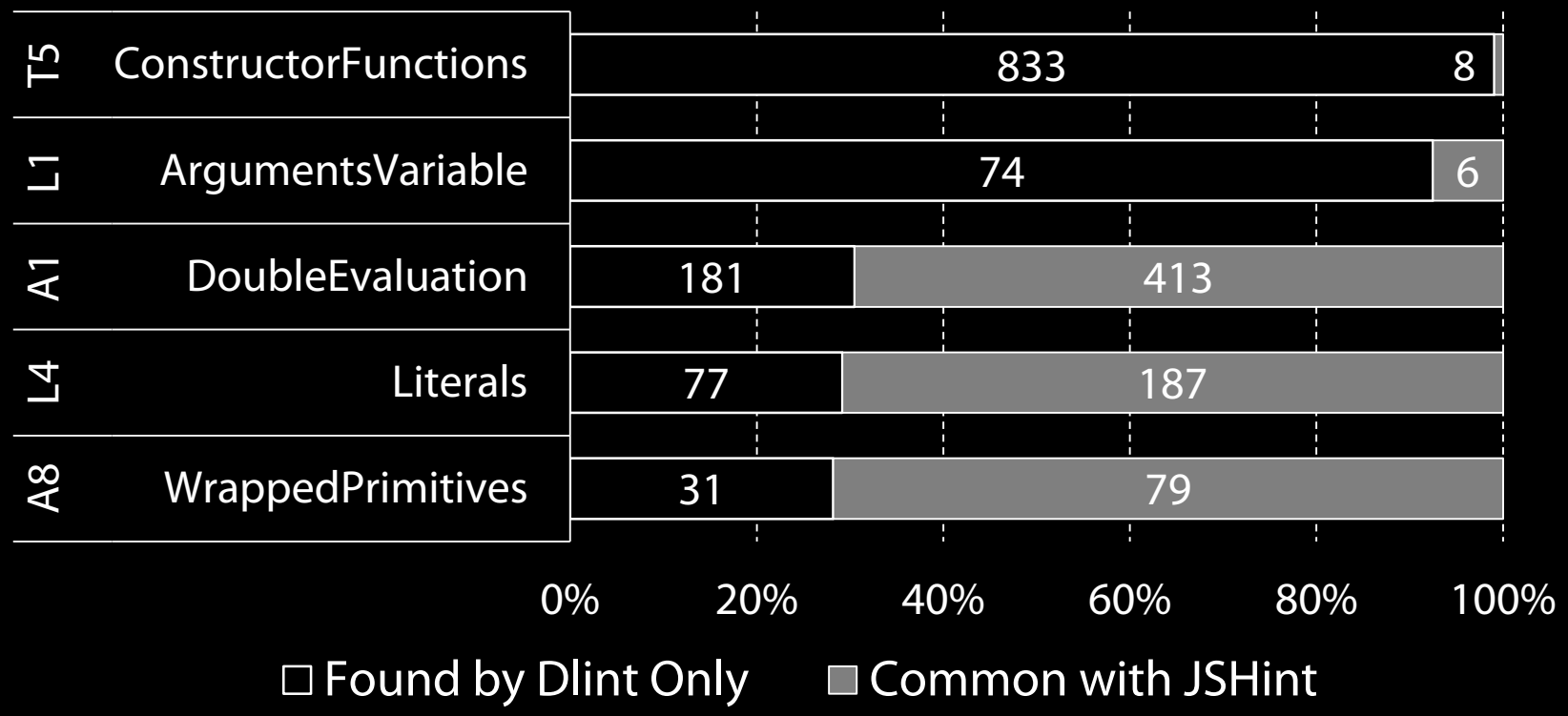
1. Predicates over runtime events

- $propWrite(base,\ name,\ val)$
- $propRead(base,\ name,\ val)$
- $cond(val)$
- $unOp(op,\ val,\ res)$
- $binOp(op,\ left,\ right,\ res)$
- $call(base,\ f,\ args,\ ret,\ isConstr)$

Example:

$propWrite(*,\ \text{"}myObject\text{"},\ val)$
$|\ isPrim(val)$


Missing 'new' prefix when invoking a constructor.
eval can be harmful.
Implied eval (string instead of function as argument).
Do not override built-in variables.
document.write can be a form of eval.
The array literal notation [] is preferable.
The object literal notation {} is preferable.
The Function constructor is a form of eval.
Do not use Number, Boolean, String as a constructor.
4933
8
2335
26
202
15
62
6
1401
167
416
125
191
62
359
205
74
79
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Fount by JSHint Only
Common with DLint




E.g., 181 calls of eval(), Function(), etc. missed by JSHint

Have an Account? Log In

TACKLE WAREHOUSE

GUARANTEED LOWEST PRICES
PRICE MATCH DETAILS - CLICK HERE
800.300.4916

My Shopping Cart:
0 Items $0.00 Checkout

ORDER TRACKING | CUSTOMER SERVICE | SHIPPING | INTERNATIONAL | MESSAGE BOARD | DEMO PROGRAM | GIFT CARDS

SHOP BY BRAND
SALE ITEMS
CASTING REELS
SPINNING REELS
CASTING RODS
SPINNING RODS
FISHING LINE
SOFT BAITS
HARD BAITS
SWIMBAITS
UMBRELLA RIGS
MARINE
APPAREL

FREE GROUND SHIPPING $5 TWO-DAY SHIPPING (1-10 LBS.) $10 NEXT DAY SHIPPING (1-4 LBS.)
Applies To All Orders Over $50. Some Restrictions Apply. Click Here For Details

Home // Fat Sack Tackle Company // Fat Sack Tackle Company Jigs //

g+1 Like 8

Twenty Five Days OF SAVINGS
A New Deal Revealed Everyday at 12pm PST!
Limited To Stock On Hand | Sale Ends 1/09 5pm PST
SHOP NOW

- $

PRICE ALERT

Color:

Description | Customer Reviews

Infinity% – –Infinity% Off
MSRP: $Infinity - $-Infinity
You Save: $Infinity - $-Infinity

Built using only top-of-the-line components, the **Fat Sack Tackle Company Fizzle Jig** delivers a high level of attraction and weedless performance. It goes where other vibrating jigs can't ... to catch bass that other jigs won't.

Fitted with a matching hex blade, the Fat Sack Tackle Company Fizzle Jig produces a strong vibration and an eye-catching flash. The premium hex blade also pulls double duty as a weedguard to keep the hook point from snagging.

As durable as it is detailed, the Fat Sack Tackle

0 Available Colors

Description | Customer Reviews

Infinity% – –Infinity% Off
MSRP: $Infinity - $-Infinity
You Save: $Infinity - $-Infinity

comScore

ENGLISH LOGIN

PRODUCTS INDUSTRIES INSIGHTS TOTAL VIDEO ACADEMY ABOUT US CAREERS

Analytics for a Digital

Click to see what the Digital World is doing

ERE: CMSC.RE/DABTA UNDEFINED

HE COMSCORE DATA MINE

Mobile App Hours per User

@COMSCORE: CHECK OUT THE TOP 10 YOUTUBE PARTNER CHANNELS BY #UNIQUEVIEWERS HERE: CMSC.RE/DABTA UNDEFINED

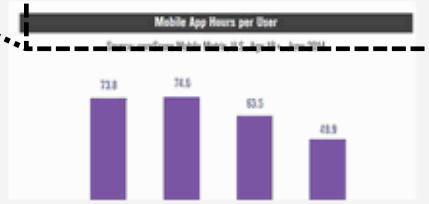
VIDEOS


15 May - From TV to Total Video: Moving from Newfronts to Allfronts with Cross-Screen Measurement.

PRESENTATIONS & WHITEPAPERS

The U.S. Total Video Report
Download now

THE COMSCORE DATA MINE




25-34 Year-Olds are the Heaviest Mobile App Users in U.S.

EVENTS & WEBINARS

12 Nov - The Cross-Platform Measurement Mandate: Tackle It Head-On

comScore Chairman Emeritus and Co-Founder, Gain Fulgoni, will present with President and CEO of the ARF Gayle Fuguitt on Laying the Foundations for Growth in Cross-Platform Measurement.
Read More

- Nato Watch Straps
- Expansion Bands
- Str
- DA
- Str
- Straps for Omega
- Straps for Bell & Ross
- Watch Parts
- Bands for Rolex
- Bands for Tissot
- Other

ebay

## sizes

| | | |
|---|---|---|
| 10 mm | 12 mm | 14 mm |
| 15 mm | 16 mm | 18 mm |
| 19 mm | 20 mm | 21 mm |
| 22 mm | 23 mm | 24 mm |
| 26 mm | 28 mm | 30 mm |

## colors

## hot sellers

StrapsCo Genuine Patent Leather Watch Strap Womens Band in Black

$NaN buy now

18mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling

$NaN buy now

20mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling

$NaN buy now

22mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling

$NaN buy now

Shark Mesh Watch Band Strap Breitling Navitimer Superocean 18mm

$NaN buy now

Matte Black PVD Shark Mesh Watch Band Strap fits Seiko 18mm 20mm

$NaN buy now

Yellow Gold PVD Shark Mesh Watch Band Strap fits Breitling 18mm 20mm

$NaN buy now

StrapsCo Expansion Watch Band Stainless Steel Strap sz 12mm

$NaN buy now

features | description | dimensions of strap | shipping | returns | policies

- Nato Watch Straps
- Expansion Bands
- Str
- DA
- Str
- Straps for Omega
- Straps for Bell & Ross
- Watch Parts
- Bands for Rolex
- Bands for Tissot
- Other

ebay

## sizes

| | | |
|---|---|---|
| 10 mm | 12 mm | 14 mm |
| 15 mm | 16 mm | 18 mm |
| 19 mm | 20 mm | 21 mm |
| 22 mm | 23 mm | 24 mm |
| 26 mm | 28 mm | 30 mm |

## colors

## hot sellers

StrapsCo Genuine Patent Leather Watch Strap Womens Band in Black
$NaN buy now

18mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling
$NaN buy now

20mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling
$NaN buy now

22mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling
$NaN buy now

Shark Mesh Watch Band Strap Breitling Navitimer Superocean 18mm
$NaN buy now

Matte Black PV Mesh Watch Ba fits Seiko 18m
$NaN buy now

$NaN buy now

Co Expansion and Stainless rap sz 12mm
$NaN buy now

18mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling
$NaN buy now

features | description | dimensions of strap | shipping | returns | policies

# NaN case study for IKEA

```
<prices>
  <normal>
    <priceNormal unformatted="9.9">$9.90</priceNormal>
    <pricePrevious />
    <priceNormalPerUnit />
    <pricePreviousPerUnit />
  </normal>
</prices>
```

XML: Empty Element

JS: undefined

```
previousPrice =
        getElementValue("pricePrevious" + suffix, normal);
```

```
parseFloat(previousPrice).toFixed(2).replace(...)
.replace('.00', ''));
```

JS: NaN

# Type Related Checker

Avoid accessing the undefined property.

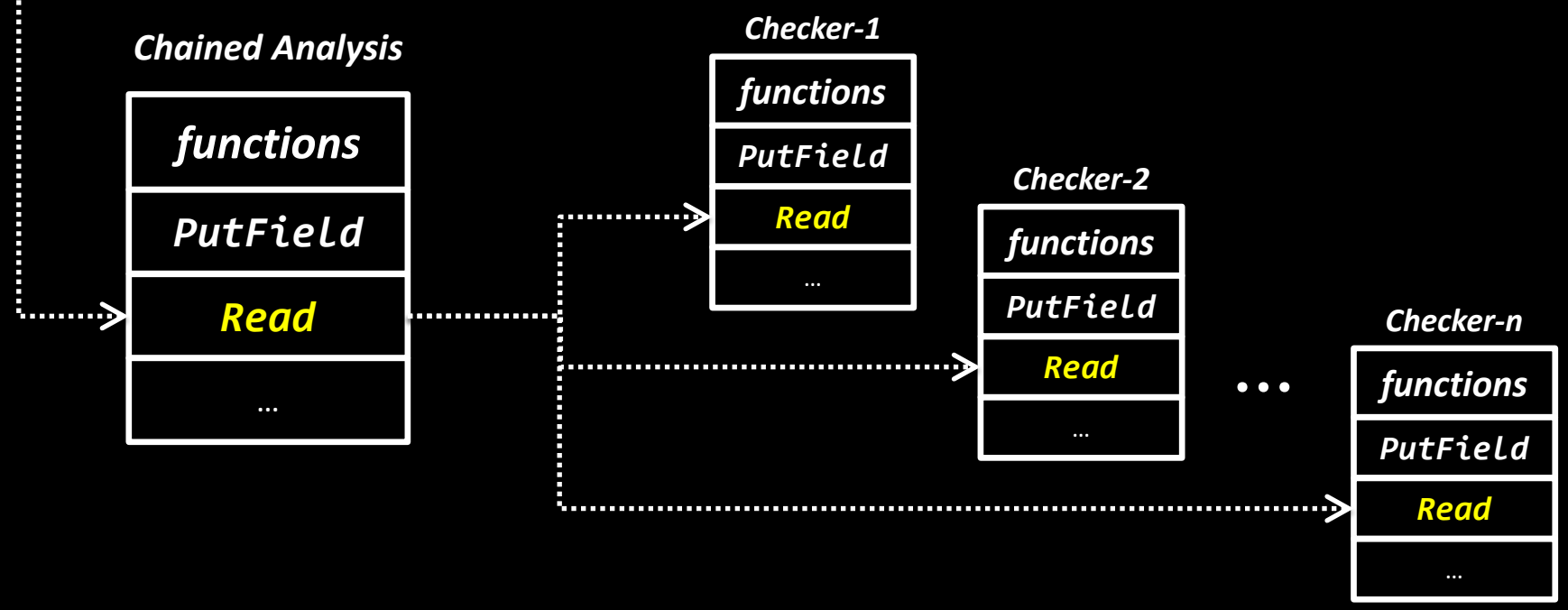
```
var x; // undefined
var y = {};
y[x] = 23; // { undefined: 23 }
```

$propWrite(*, "undefined", *)$
$propRead(*, "undefined", *)$

# Chained Analysis

```
a.f = b.g
```

```
PutField(Read("a", a), "f", GetField(Read("b", b), "g"))
```



# Rule: avoid using *for..in* on arrays

www.google.com/chrome,
included code from Modernizr:

```
for (i in props) {  // props is an array
  prop = props[i];
  before = mStyle.style[prop];
  ...
}
```

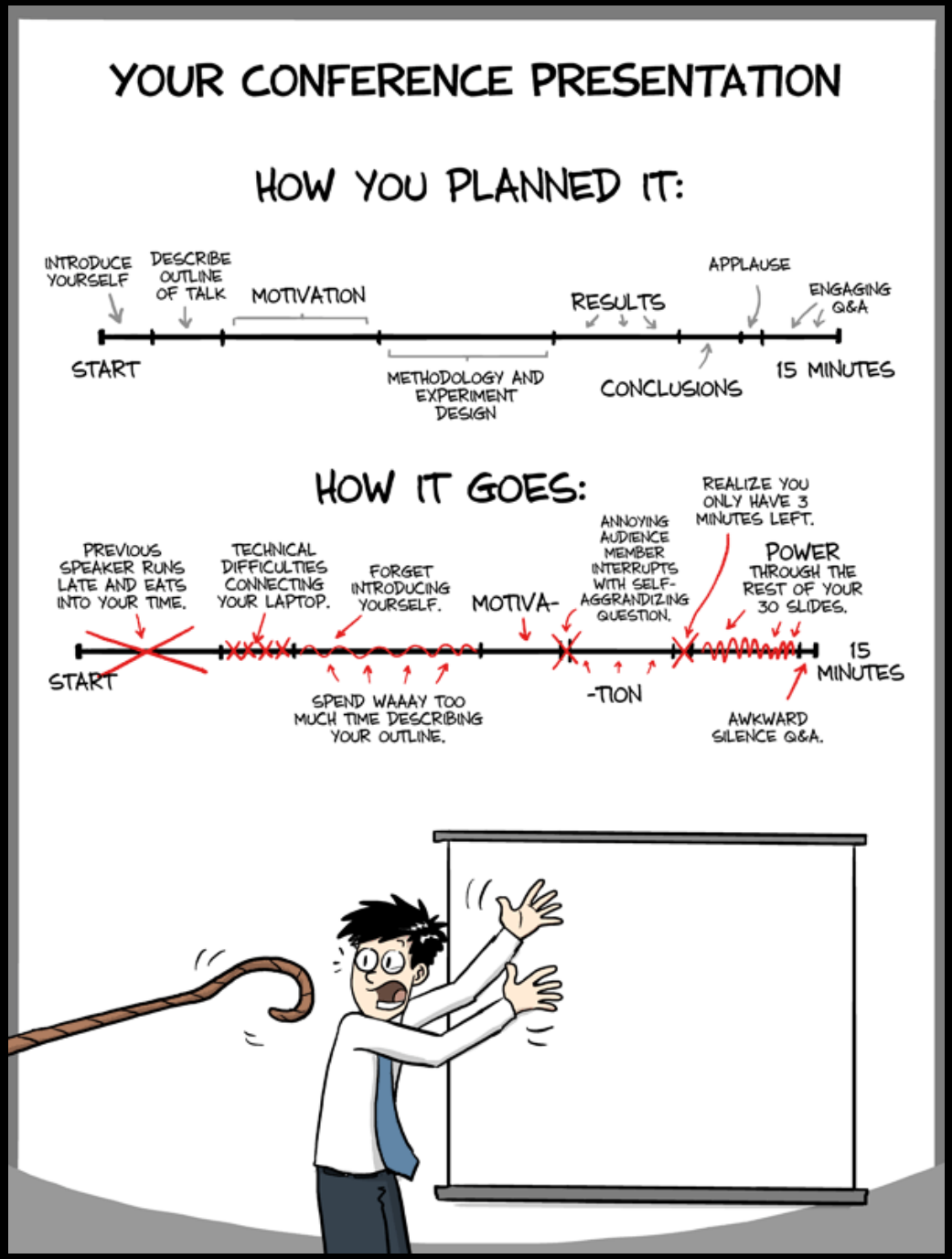
Modernizr
https://github.com/Modernizr/Modernizr/pull/1419

# API Misuse

***eval*** is evil, do not use ***eval***.

```
var fun = eval;
...
⚠️ fun("var a = 1;");
```

```
call(builtin, eval, *, *, *)
call(builtin, Function, *, *, *)
call(builtin, setTimeout, args, *, *)
    | isString(args[0])
call(builtin, setInterval, args, *, *)
    | isString(args[0])
call(document, write, *, *, *)
```


YOUR CONFERENCE PRESENTATION
HOW YOU PLANNED IT:
INTRODUCE YOURSELF
DESCRIBE OUTLINE OF TALK
MOTIVATION
RESULTS
APPLAUSE
ENGAGING Q&A
START
METHODOLOGY AND EXPERIMENT DESIGN
CONCLUSIONS
15 MINUTES
HOW IT GOES:
PREVIOUS SPEAKER RUNS LATE AND EATS INTO YOUR TIME.
TECHNICAL DIFFICULTIES CONNECTING YOUR LAPTOP.
FORGET INTRODUCING YOURSELF.
MOTIVA-
ANNOYING AUDIENCE MEMBER INTERRUPTS WITH SELF-AGGRANDIZING QUESTION.
REALIZE YOU ONLY HAVE 3 MINUTES LEFT.
POWER THROUGH THE REST OF YOUR 30 SLIDES.
START
SPEND WAAAY TOO MUCH TIME DESCRIBING YOUR OUTLINE.
-TION
AWKWARD SILENCE Q&A.
15 MINUTES