To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

Zheng Gao⁺, Christian Bird*, Earl Barr⁺

⁺University College London, *Microsoft Research
Static Typing  vs.  Dynamic Typing
Static Typing  vs.  Dynamic Typing

- Performance
- Modularity
- Documentation
- Early Detection

- Conciseness
- Flexibility
- Expressiveness
- Metaprogramming
Static Typing vs. Dynamic Typing

Early Detection
Engine of the Web

is dynamically typed;

has a large set of long-running projects.
Engine of the Web

is dynamically typed;

has a large set of long-running projects.

3,599,113 JavaScript repos on GitHub
Static Typing for JavaScript
Static Typing for JavaScript

Microsoft → TypeScript

facebook → flow
Had Static Typing been Used ...
Had Static Typing been Used …
Had Static Typing been Used …
Had Static Typing been Used ...
Central Finding
Central Finding

15%
Central Finding
Bug Life Cycle
Bug Life Cycle
Bug Life Cycle

Saved bugs

Edit-time bugs
Bug Life Cycle

1. git commit
2. git push

team bugs
saved bugs
edit-time bugs
Bug Life Cycle

1. git commit
2. git push
Bug Life Cycle

field bugs

team bugs
Bug Life Cycle

field bugs

public bugs

team bugs
Type System Detectable

**Definition** (ts-detectable): Given a static type system $ts$, a bug is $ts$-detectable when

1. adding or changing type annotations causes the program containing the bug to fail to type check on a line a fix changes.
Problem

When the type of \( b \) is nullable \texttt{number}, annotating

\begin{verbatim}
    var a = b + 1;
\end{verbatim}

to

\begin{verbatim}
    var a:boolean = b + 1;
\end{verbatim}

“trivially” triggers a type error.
Consistency

**Definition** (Consistency): The added or changed type annotations are consistent with a fixed version of the program containing the bug $f$, if they carried to $f$ type check, and the type of every annotated term is a supertype of that term’s type when an oracle precisely annotates it in $f$. 
Type System Detectable

**Definition** (*ts*-detectable): Given a static type system *ts*, a bug is *ts*-detectable when

1. adding or changing type annotations causes the program containing the bug to fail to type check on a line a fix changes;

2. the new annotations are *consistent* with a fixed version of the program containing the bug.
Example of Detection

JavaScript:
```
1  // addNumbers in JavaScript
2  function addNumbers(x, y) {
3      return x + y;
4  }
5  console.log(addNumbers(3, "0"));
```

TypeScript:
```
1  // addNumbers in TypeScript
2  function addNumbers(x:number, y:number) {
3      return x + y;
4  }
5  console.log(addNumbers(3, "0"));
```

Error-free in JavaScript, and unexpectedly displays an string, 30.

TypeScript throws the following error:
```
t.ts(5,27): error TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.
```
Research Question

What percentage of public bugs are detectable under Flow or TypeScript?
Experiment Overview
Corpus Collection
Corpus Collection

- What is the sample size?
- How to identify public bugs?
Corpus Collection

- What is the sample size?
- How to identify public bugs?
Sample Size Calculation

\[ s = \frac{X^2NP(1 - P)}{d^2(N - 1) + X^2P(1 - P)} \]

s: sample size

\( X^2 \): a constant for the confidence level of 95%

N: population size, 3910969

P: population proportion, 0.5

d: degree of accuracy, 0.05

3,910,969 closed bug reports

384 bugs
Corpus Collection

- What is the sample size?
- How to identify public bugs?
Bug Identification
Bug Identification
Bug Identification

Parents are buggy

Bug Identification → Fix Identification

GitHub

Projects

Link Identification

Issues

Commit Logs

Candidates

Filter

Buggy Programs

Filter

Annotated Programs

Annotate

Type Checking

No

Detectable Bugs
Fix Identification

- Issue Pages
  - Extract Commit Hashes
  - Linked Fix Candidates

- Commit Logs
  - Extract Issue Numbers
  - Linked Fix Candidates
These candidates may include commits that add features or refactor.
Subjects

- general bugs
- ts-detectable bugs
- public bugs
Subjects

general bugs

ts-detectable bugs

public bugs
Subjects

general bugs

ts-detectable bugs

public bugs

fixed public bugs
Size Statistics of the Corpus

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>1144440</td>
<td>32</td>
<td>18117.9</td>
<td>1736</td>
</tr>
<tr>
<td>Fix</td>
<td>270</td>
<td>1</td>
<td>16.2</td>
<td>6</td>
</tr>
</tbody>
</table>

The sizes are in lines of code.
Methodology

Project Version History

Researchers
Methodology

Project Version History

Researcher
Methodology

Project Version History

$p_0$  ...  buggy version $p_{i-1}$  $p_i$  fixed version  ...  $p_n$

travel back in time
Methodology

Project Version History

Check out $p_{i-1}$
Methodology

Gradually annotate $p_{i-1}$
Methodology

a is the annotation function

Project Version History

Type check $a(p_{i-1})$
Annotation
Annotation

- Gradually add annotation
- Type check
- Detected?
  - Yes
    - Explain annotations
  - No
    - Possibly detectable?
      - Yes
        - Justify undetectable
      - No
        - Undetectable

GitHub

Link

Identification

Projects → Issues → Commit Logs

Candidates

Filter

Buggy Programs → Annotated Programs

Type Checking

Detectable Bugs
We do not fully annotate the program; we rely on gradual typing to locally, minimally annotate the patched region.
Annotation Sources

- bug fixes
- bug reports
- project documentation
Expert Source
Problem

```javascript
var t = {x:0, z:1};
t.x = t.y; // the error is y does not exist on t
t.x = t.z;
```
```javascript
var t = {x:0, z:1};
t.x = t.y; // the error is y does not exist on t
t.x = t.z;
```

What is the type of variable t?
Problem

What is the type of variable t?

```
var t = {x: 0, z: 1};
t.x = t.y; // the error is y does not exist on t
```

Seems to be {x: number, z: number}?
```javascript
var t = {x: 0, z: 1};
t.x = t.y; // the error is y does not exist on t
t.x = t.z;
```

**Problem**

What is the type of variable t?

Seems to be `{x: number, z: number}`?

Not necessarily!
Problem

What is the type of variable t?

```javascript
var t = {x: 0, z: 1};
t.x = t.y; // the error is y does not exist on t
t.x = t.z;
...
...
...
t.x = "a";
```

Seems to be {x: number, z: number}?

Not necessarily!
Problem

What is the type of variable t?

```javascript
var t = {x:0, z:1};
t.x = t.y; // the error is y does not exist on t
t.x = t.z;

...  
...  
...  
t.x = "a";
```

Seems to be `{x: number, z: number}`?

Not necessarily!

Now becomes `{x: number | string, z: number}`.
Type Shims

A set of type bindings for the free identifiers that

1). is *consistent* with but

2). may not exist in

a fixed version of the program containing the bug.
Shim Example

```javascript
var t = {x:0, z:1};
t.x = t.y; // y does not exist on t
t.x = t.z;

interface T {
  x: any;
  z: any;
}
var t:T = {x:0, z:1};
t.x = t.y;
t.x = t.z;
```

This shim is consistent, as T must be the supertype.
Annotation Quality

84% of the annotated fixed versions type check.

we add the same annotations to $p_i$
Results

Both Flow and TypeScript detect 15% of the collected bugs; the confidence range is [11.5%, 18.5%], at a 95% confidence level.
Implications

“That’s shocking. If you could make a change to the way we do development that would reduce the number of bugs being checked in by 10% or more overnight, that’s a no-brainer. Unless it doubles development time or something, we’d do it.”

- An engineering manager at Microsoft
Experimental Artefacts

To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

OVERVIEW

Javascript being a dynamically typed language and these could have a huge impact on production code outside the real world. Javascript is thus traditionally typed language for both static type systems, namely Facebook’s F# and Microsoft’s TypeScript. Juxtapose, what benefits do these static type systems provide?

Literally, by using static typing, we define what a bug will look like and what the end user will see. For example, the above code containing the type error will return the output “Number is not defined”. This allows for more declarative code and reduces bugs in the real world. However, this is at the cost of runtime flexibility and increased development time. For example, a developer writes a function that returns a number. Without static typing, the function may return a string, which can cause issues at runtime. Static typing, on the other hand, allows for more flexibility and reduces bugs in the real world.

METHODOLGY

The factor that long running in abstract production code classes comes with the experience of static type systems that improve speed and clarity in abstract production code. With abstract production code, it becomes clear what the function does and what the function returns. This is especially important for large applications where the function may perform many operations and return different types of data.

RESULTS

What we have discovered

- More bugs detected with TypeScript
- Fewer bugs detected with Flow
- Developers say

"That's odd. I would have thought there would be a lot of bugs."

An Engineering Manager, Microsoft

CASE STUDY

Some bugs are easier to spot than others. In our case study, we observed three types of bugs: type errors, flow errors, and logic errors. Type errors occur when the function returns a value of the wrong type, such as returning a number when expected a string. Flow errors occur when the function performs operations that are not defined in the type system, such as returning a string when expected a number. Logic errors occur when the function performs operations that are undefined in the type system, such as performing a mathematical operation on a string.

http://ttendency.cs.ucl.ac.uk/projects/type_study/index.html
Research Question

What is the percentage of public bugs that are detectable under Flow or TypeScript?

Experiment Overview

Methodology

\[ a \text{ is the annotation function} \]

Results

\[
\begin{align*}
\text{collected bugs} & \quad 400 \\
\text{Flow-detectable} & \quad 57 \\
\text{TypeScript-detectable} & \quad 3 \\
\end{align*}
\]

The confidence range for both Flow and TypeScript is \([11.5\%, 18.5\%]\), at a 95% confidence level.

http://ttendency.cs.ucl.ac.uk/projects/type_study/index.html