JAVASCRIPT MUTATION TESTING

Improving Confidence in Your Tests

Brian Batronis

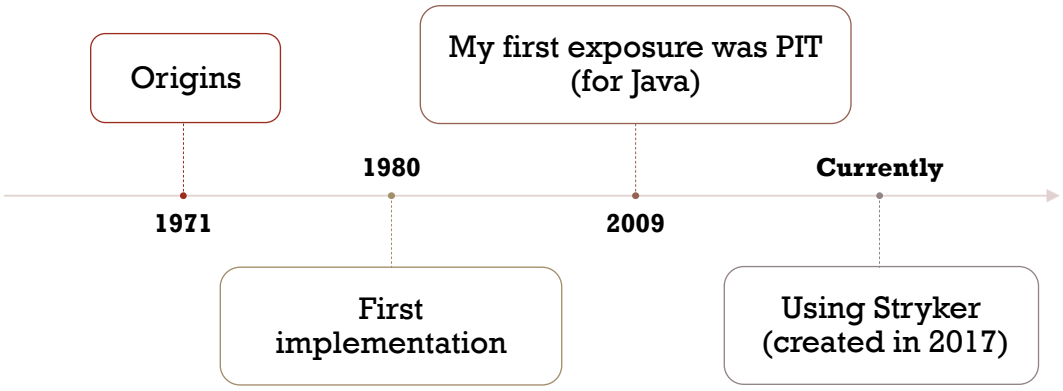
Confidence

WHY DO WE WRITE UNIT TESTS?

HOW DOES MUTATION TESTING WORK?

Introduce a single fault (mutant) → Test mutated code → Check if mutant was dispatched

HISTORY



SIMPLE EXAMPLE

Code:

```
const isPositive = (num) => {
    return num > 0;
}
```

Test:

```
expect(isPositive(5)).toBe(true);
```

SIMPLE EXAMPLE MUTATIONS


1. `const isPositive = (num) => {}`


2. `return false;`


3. `return num <= 0;`


4. `return true;`


5. `return num >= 0;`

```
expect(isPositive(5)).toBe(true);
```

SIMPLE EXAMPLE MUTATIONS

1. `const isPositive = (num) => {}`
2. `return false;`
3. `return num <= 0;`
4. `return true;`
5. `return num >= 0;`

```
expect(isPositive(5)).toBe(true);
expect(isPositive(-3)).toBe(false);
expect(isPositive(0)).toBe(false);
```

MUTATION TEST ADVANTAGES

More reliable metrics than coverage

Ensures test adequacy

MUTATION TESTING DISADVANTAGES

- Many mutants
- All tests are run for every mutant

STRYKER MUTATORS

Arithmetic Operator	Array Declaration	Block Statement	Boolean Literal	Conditional Expression
Equality Operator	Logical Operator	Method Expression	Object literal	Optional chaining
Regex	String Literal	Unary Operator	Update Operator	

THEORY

Clean code produces fewer mutants

IF-ELSE

```
if (operationName === 'A') {●●●●●
    someObject.item1 = true;●
    someObject.item2 = false;●
    someObject.item3 = false;●
} else if (operationName === 'B') {●●●●●
    someObject.item1 = false;●
    someObject.item2 = true;●
    someObject.item3 = false;●
} else if (operationName === 'C') {●●●●●
    someObject.item1 = false;●
    someObject.item2 = false;●
    someObject.item3 = true;●
}
```

24 Mutants

IF-ELSE REFACTORED

```
someObject.item1 = false;●
someObject.item2 = false;●
someObject.item3 = false;●
const operations = {●
    A: () => { someObject.item1 = true },●●
    B: () => { someObject.item2 = true },●●
    C: () => { someObject.item3 = true } ●●
}
operations[operationName]();
```

10 Mutants

COMPLEX EXAMPLE

32 Mutants

```
const doStuff = (result) => {●
    let resultData = [];●
    if (result.aRecords !== undefined && result.aRecords.length > 0) {●●●●●●●●●
        for (let i = 0; i < result.aRecords.length; i++) {●●●●●
            resultData.push({ "createdByName": result.aRecords[i].Name })●
        }
    }
    if (result.bRecords !== undefined && result.bRecords.length > 0) {●●●●●●●●●
        for (let i = 0; i < result.bRecords.length; i++) {●●●●●
            resultData.push({ "createdByName": result.bRecords[i].Name })●
        }
    }
    return resultData;
}
```

COMPLEX EXAMPLE REFACTOR

10 Mutants

```
const getRecordData = (recordData) => {●
    return recordData?.map((record) => {●●
        return { createdByName: record.Name };●
    });
}

const doStuff = (result) => {●
    const aRecordData = getRecordData(result.aRecords);
    const bRecordData = getRecordData(result.bRecords);
    return [...(aRecordData ?? []), ...(bRecordData ?? [])];●●●●●
}
```

DEMO

MORE INFORMATION

ARITHMETIC OPERATOR

Original	Mutated
a + b	a - b
a - b	a + b
a * b	a / b
a / b	a * b
a % b	a * b

ARRAY DECLARATION

Original	Mutated
new Array(1, 2, 3, 4)	new Array()
[1, 2, 3, 4]	[ ]

BLOCK STATEMENT

Removes the content of every block statement.

The following:

```
const isPositive = (num) => {
    return num > 0;
}
```

becomes:

```
const isPositive = (num) => {}
```

BOOLEAN LITERAL

Original	Mutated
true	false
false	true
!(a == b)	a == b

CONDITIONAL EXPRESSION

Original	Mutated
for (var i = 0; i < 10; i++) { }	for (var i = 0; false; i++) { }
while (a > b) { }	while (false) { }
do { } while (a > b);	do { } while (false);
if (a > b) { }	if (true) { }
if (a > b) { }	if (false) { }
var x = a > b ? 1 : 2;	var x = true ? 1 : 2;
var x = a > b ? 1 : 2;	var x = false ? 1 : 2;

EQUALITY OPERATOR

Original	Mutated
a < b	a <= b
a < b	a >= b
a <= b	a < b
a <= b	a > b
a > b	a >= b
a > b	a <= b

Original	Mutated
a >= b	a > b
a >= b	a < b
a == b	a != b
a != b	a == b
a === b	a !== b
a !== b	a === b

LOGICAL OPERATOR

Original	Mutated
a && b	a || b
a || b	a && b
a ?? b	a && b

METHOD EXPRESSION

Original	Mutated
endsWith()	startsWith()
startsWith()	endsWith()
trim()	
trimEnd()	trimStart()
trimStart()	trimEnd()
substr()	
substring()	

Original	Mutated
toUpperCase()	toLowerCase()
toLowerCase()	toUpperCase()
toLocalLowerCase()	toLocalUpperCase()
toLocalUpperCase()	toLocalLowerCase()
sort()	
some()	every()
every()	some()

METHOD EXPRESSION CONTINUED

Original	Mutated
reverse()	
filter()	
slice()	
charAt()	
min()	max()
max()	min()

OBJECT LITERAL

Original	Mutated
{ foo: 'bar' }	{ }

OPTIONAL CHAINING

Original	Mutated
foo?.bar	foo.bar
foo?.[1]	foo[1]
foo?.()	foo()

REGEX

Original	Mutated
^abc	abc
abc$	abc
[abc]	[^abc]
[^abc]	[abc]
\d	\D
\D	\d
\s	\S

Original	Mutated
\S	\s
\w	\W
\W	\w
a?	a
a*	a
a+	a
a{1,3}	a

REGEX CONTINUED

Original	Mutated
a*?	a
a+?	a
a{1,3}?	a
a?+	a
a*+	a
a++	a
a{1,3}+	a

Original	Mutated
(?=abc)	(?!abc)
(?!abc)	(?=abc)
(?<=abc)	(?<!abc)
(?<!abc)	(?<=abc)

STRING LITERAL

Original	Mutated
"foo" (non-empty string)	"" (empty string)
"" (empty string)	"Stryker was here!"
s"foo ${bar}" (string interpolation)	s""

UNARY OPERATOR

Original	Mutated
+a	-a
-a	+a

UPDATE OPERATOR

Original	Mutated
a++	a--
a--	a++
++a	--a
--a	++a