

Teste de mutação

E automação com *Striker Mutator*

Apresentação Individual - 13/11/2020 - Lucas Barioni Toma
Professores: Adilson Marques da Cunha e Luiz Alberto Vieira Dias

Lucas Barioni Toma

 : ELE-21 (4º ano graduação)

 TERRAMAGNA : Desenvolvedor Backend

TS#2 : Médicos

Cursando CE-237 como disciplina eletiva



lbtoma@gmail.com

Objetivos

- Introduzir a técnica de teste de mutação (ou de mutantes) no contexto do *Test Driven Development* (TDD); e
- Apresentar uma ferramenta para automatizar testes de mutação.

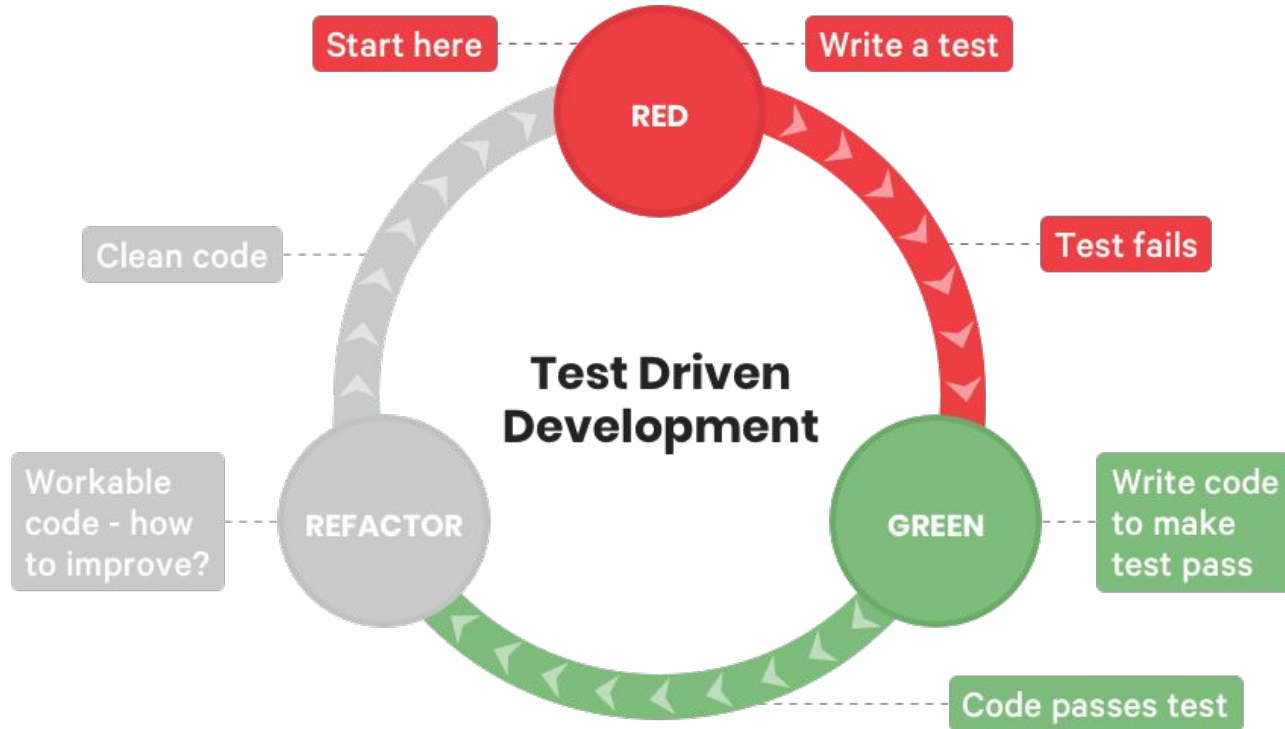
Roteiro

1. Contextualização: breve revisão sobre *Test Driven Development* (TDD);
2. A partir do contexto do TDD, introduzir a técnica de teste de mutação;
3. Mostrar exemplos, aplicações e seus principais benefícios;
4. Apresentar o *Stryker Mutator* como ferramenta para automatizar a execução do teste de mutação;
5. Apresentar o uso dessa ferramenta no contexto da disciplina e do projeto **STEPES-TR**; e
6. Levantar possíveis pontos negativos.

Considerações

- Contexto principal: Test Driven Development (TDD)
- O usual é utilizar o teste de mutação para analisar a eficácia dos testes de unidade
- Normalmente não se utiliza a teste de mutação juntamente com testes de integração

Falando um pouco sobre TDD



Suponhamos que desejamos criar a função `fibonacci()`

$$\begin{cases} f(n) = f(n-1) + f(n-2); \\ f(1) = f(2) = 1; \end{cases}$$

Criando os testes de unidade (*test cases*)

...

```
test("Must return 0 for 0", () => {  
  const result = fibonacci(1);  
  
  expect(result).toBe(0);  
});
```

```
test("Must return 1 for 1", () => {  
  const result = fibonacci(1);  
  
  expect(result).toBe(1);  
});
```

...

Criando os testes de unidade (*test cases*)

```
./fibonacci.test.js
```

```
Must return 0 for 0
```

```
Must return 1 for 1
```

```
Must return 1 for 2
```

```
Must return 0 for any negative number
```

```
f(n) must be f(n+2) - f(n+1) (3 ms)
```

Implementação ingênua

```
function fibonacci(n) {  
  if (n == 1) {  
    return 1;  
  } else if (n == 2) {  
    return 1;  
  } else {  
    return fibonacci(n - 1) + fibonacci(n - 2);  
  }  
}
```

$$\begin{cases} f(n) = f(n-1) + f(n-2); \\ f(1) = f(2) = 1; \end{cases}$$

Executando os testes

FAIL ./fibonacci.test.js

✗ Must return 0 for 0 (6 ms)

✓ Must return 1 for 1 (2 ms)

✓ Must return 1 for 2

✗ Must return 0 for any negative number (3 ms)

✗ $f(n)$ must be $f(n+2) - f(n+1)$ (3 ms)

Refatorando a função fibonacci()

```
function fibonacci(n) {  
  if (n == 1) {  
    return 1;  
  } else if (n == 2) {  
    return 1;  
  } else {  
    return fibonacci(n - 1) + fibonacci(n - 2);  
  }  
}
```

Para aceitar o caso em que $n == 0$

```
function fibonacci(n) {  
  if (n == 0) {  
    return 0;  
  } else if (n == 1) {  
    return 1;  
  } else if (n == 2) {  
    return 1;  
  } else {  
    return fibonacci(n - 1) + fibonacci(n - 2);  
  }  
}
```

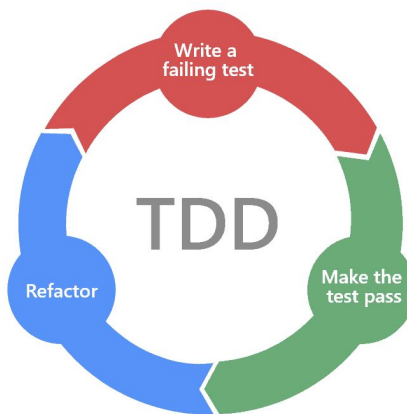
Executando os testes novamente

```
PASS ./fibonacci.test.js
✓ Must return 0 for 0 (3 ms)
✓ Must return 1 for 1 (1 ms)
✓ Must return 1 for 2
✓ Must return 0 for any negative number
✓ f(n) must be f(n+2) - f(n+1) (1 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.457 s
```

Observação:

Todos os testes passaram neste momento, mas normalmente o ciclo se repete várias vezes



Aparentemente está tudo bem,

Você criou seus testes de unidade,

os testes falharam,

you refactored the code until the tests pass,

and you feel **satisfied**

PASS `./fibonacci.test.js`

- ✓ Must return 0 for 0 (3 ms)
- ✓ Must return 1 for 1 (1 ms)
- ✓ Must return 1 for 2
- ✓ Must return 0 for any negative number
- ✓ $f(n)$ must be $f(n+2) - f(n+1)$ (1 ms)



Agora, vamos ver o que ocorre quando introduzimos algumas anomalias no código

```
3 function fibonacci(n) {
4   if (n == 0) {
5     return 0;
6   } else if (n == 1) {
7     return 1;
8   } else if (n == 2) {
9     return 1;
10  } else {
11-   return fibonacci(n - 1) + fibonacci(n - 2);
12  }
13 }
14
15
16 module.exports = fibonacci;
```

```
3 function fibonacci(n) {
4   if (n == 0) {
5     return 0;
6   } else if (n == 1) {
7     return 1;
8   } else if (n == 2) {
9     return 1;
10  } else {
11+   return fibonacci(n - 1) * fibonacci(n - 2);
12  }
13 }
14
15
16 module.exports = fibonacci;
```

Todos os testes passaram!

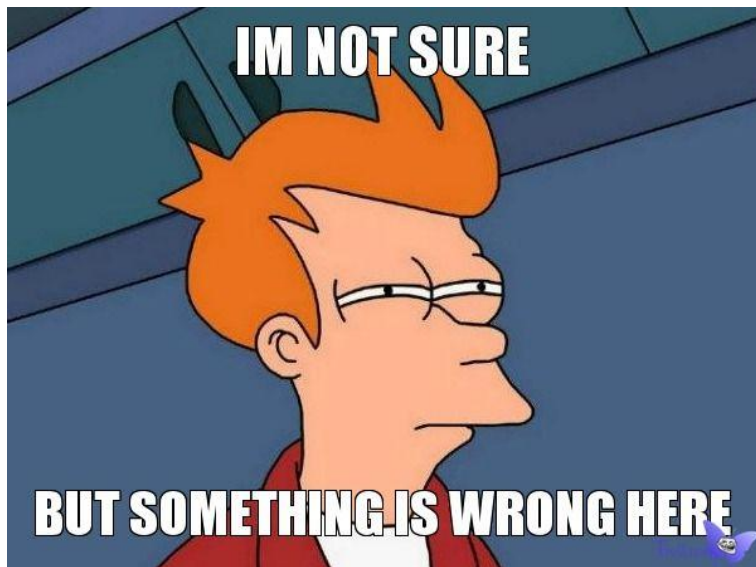
```
PASS ./fibonacci.test.js
  ✓ Must return 0 for 0 (3 ms)
  ✓ Must return 1 for 1
  ✓ Must return 1 for 2 (1 ms)
  ✓ Must return 0 for any negative number
  ✓ f(n) must be f(n+2) - f(n+1)
> tdd-example-1
Test Suites: 1 passed, 1 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 1.292 s
Ran all test suites.
Done in 3.21s.
```

Todos os testes passaram!?

Você criou uma anomalia no seu código

e os teste continuaram passando,

provavelmente há algo **errado**



O propósito dos testes é garantir o correto funcionamento do *software*, mas como garantir que os testes estão cumprindo essa premissa?



Uma forma de **testar** os **testes** é por meio do

Teste de **mutação**

Tipos de empresa que utilizam do teste de mutação

- Finaças
- Seguros
- Biotecnologia
- Laboratórios de pesquisa como o CERN

Ou seja, empresas que prestam serviços com **alto custo sobre falhas**.

Como funciona?

Executar a suite de testes original para garantir que todos os testes estão passando e, então introduzir as mutações:

- Trocar operadores:
 - Lógicos: `&&`, `||`;
 - Comparativos: `>`, `>=`, `<`, `==`, `!=`
 - Matemáticos: `+`, `-`, `*`
 - *Bitwise*: `<<`, `>>`, `^`, `&`, `|`
- Trocar *true* por *false*
- Omitir alguma chamada de função
- Trocar alguma chamada de função
- Etc



Gera uma infinidade de combinações

Retornando à função `fibonacci()`

Mesmo trocando o operador `+` por `*`, os testes passaram

```
function fibonacci(n) {  
  if (n == 0) {  
    return 0;  
  } else if (n == 1) {  
    return 1;  
  } else if (n == 2) {  
    return 1;  
  } else {  
    return fibonacci(n - 1) * fibonacci(n - 2);  
  }  
}
```

PASS `./fibonacci.test.js`

- ✓ Must return 0 for 0 (3 ms)
- ✓ Must return 1 for 1 (1 ms)
- ✓ Must return 1 for 2
- ✓ Must return 0 for any negative number
- ✓ $f(n)$ must be $f(n+2) - f(n+1)$ (1 ms)

Examinando melhor o script de testes

```
const fibonacci = require("./fibonacci");

function randomNumber() {
  return Math.floor(Math.random());
}

...

test("f(n) must be f(n+2) - f(n+1)", () => {
  const n = randomNumber();
  const result = fibonacci(n);

  expect(result).toBe(fibonacci(n + 2) - fibonacci(n + 1));
});

...
```



```
const fibonacci = require("./fibonacci");
```

```
function randomNumber() {
```

```
  return Math.floor(Math.random() * 1000000);
```

```
}
```

```
...
```

```
test("f(n) must be f(n+2) - f(n+1)", () => {
```

```
  const n = randomNumber();
```

```
  const result = fibonacci(n);
```

```
  expect(result).toBe(fibonacci(n + 2) - fibonacci(n + 1));
```

```
});
```

```
...
```

O método `Math.random()` retorna apenas números entre 0 e 1

```
> Math.random()
```

```
0.8710025305127436
```

```
> Math.random()
```

```
0.9837449568983341
```

```
> Math.floor(Math.random())
```

```
0
```

```
> Math.floor(Math.random())
```

```
0
```

Conclusão: havia um defeito dentro do próprio teste

Alterando a função `randomNumber()` para gerar corretamente números de zero a 10...

<pre>3 function randomNumber() { 4- return Math.floor(Math.random()); 5 }</pre>	<pre>3 function randomNumber() { 4+ return Math.floor(Math.random()*10); 5 }</pre>
---	--

```
FAIL ./fibonacci.test.js  
✓ Must return 0 for 0 (3 ms)  
✓ Must return 1 for 1 (4 ms)  
✓ Must return 1 for 2  
✗ Must return 0 for any negative number (5 ms)  
✗ f(n) must be f(n+2) - f(n+1) (3 ms)
```

...alguns testes passam a falhar

Podemos encontrar outros defeitos agora

```
3 function fibonacci(n) {
4-   if (n == 0) {
5       return 0;
6   } else if (n == 1) {
7       return 1;
8   } else if (n == 2) {
9       return 1;
10  } else {
11      return fibonacci(n - 1) + fibonacci(n - 2);
12  }
13 }
```

FAIL ./fibonacci.test.js

- ✓ Must return 0 for 0 (3 ms)
- ✓ Must return 1 for 1 (4 ms)
- ✓ Must return 1 for 2
- ✗ Must return 0 for any negative number (5 ms)
- ✗ $f(n)$ must be $f(n+2) - f(n+1)$ (3 ms)

● **Must return 0 for any negative number**

RangeError: Maximum call stack size exceeded

```
3 function fibonacci(n) {
4+   if (n <= 0) {
5       return 0;
6   } else if (n == 1) {
7       return 1;
8   } else if (n == 2) {
9       return 1;
10  } else {
11      return fibonacci(n - 1) + fibonacci(n - 2);
12  }
13 }
```

PASS ./fibonacci.test.js

- ✓ Must return 0 for 0 (3 ms)
- ✓ Must return 1 for 1 (1 ms)
- ✓ Must return 1 for 2
- ✓ Must return 0 for any negative number
- ✓ $f(n)$ must be $f(n+2) - f(n+1)$ (1 ms)

Test Suites: 1 passed, 1 total

Tests: 5 passed, 5 total

Snapshots: 0 total

Time: 1.423 s

Ran all test suites.

Done in 3.36s.

Outro importante benefício

O teste de mutação também aponta partes do código que podem ser refatoradas

Trocando `==` por `<` ...

```
function fibonacci(n) {  
  if (n <= 0) {  
    return 0;  
  } else if (n == 1) {  
    return 1;  
  } else if (n < 2) {  
    return 1;  
  } else {  
    return fibonacci(n - 1) + fibonacci(n - 2);  
  }  
}
```

Oportunidade de
refact

```
PASS ./fibonacci.test.js  
✓ Must return 0 for 0 (3 ms)  
✓ Must return 1 for 1  
✓ Must return 1 for 2  
✓ Must return 0 for any negative number (1 ms)  
✓ f(n) must be f(n+2) - f(n+1)  
  
Test Suites: 1 passed, 1 total  
Tests: 5 passed, 5 total  
Snapshots: 0 total  
Time: 1.457 s  
Ran all test suites.  
Done in 3.40s.
```

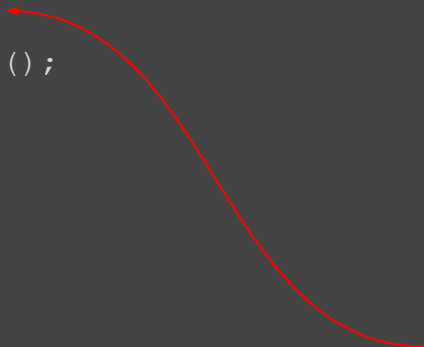
... notamos que o há um trecho de código **desnecessário**

A codebase agora está mais limpa

```
function fibonacci(n) {  
  if (n <= 0) {  
    return 0;  
  } else if (n == 1) {  
    return 1;  
  } else {  
    return fibonacci(n - 1) + fibonacci(n - 2);  
  }  
}
```

Outro exemplo

```
function someLogic (num: number): void {  
  if (num <= 10) {  
    throw new Error("num can not be lower or equal to 10" );  
  }  
  if (num >= 10) {  
    doSomethingElse ();  
  }  
}  
  
...  
if (num > 10) {  
  doSomethingElse ();  
}  
...
```



Pode-se detectar

- Código morto ou não utilizado
- Código que afeta apenas o estado interno
- Código que afeta apenas a performance

Criar as mutações manualmente pode ser muito trabalhoso, para isso existem *frameworks* que fazem esse trabalho na *codebase* de maneira automatizada

Stryker Mutator

Test your tests with mutation testing.





JavaScript and friends



C#



Scala

Stryker Mutator

Framework para automatizar a execução dos testes de mutação e gerar relatórios e atribuir notas aos arquivos do código. Disponível para:

- Javascript (inclui Typescript, ReactJS, etc)
- C#
- Scala

Backend de Médicos - STEPES-BD

PASS src/app.test.js

Test physicians routes

- ✓ Creating a new physician (84ms)
- ✓ Get the created physician (117ms)
- ✓ Update the physician (118ms)

Test specialties routes

- ✓ Creating a new specialty (7ms)
- ✓ Get the created specialty (108ms)
- ✓ Update the specialty (113ms)

Test physician specialty routes

- ✓ Creating a new physician specialty (9ms)
- ✓ Get the created physician specialty (108ms)
- ✓ Get the physician specialties (9ms)
- ✓ Delete the physician specialty (105ms)

Test addresses routes

- ✓ Get the addresses (3ms)

Test contact routes

- ✓ Creating a new contact (5ms)
- ✓ Get the created contact (108ms)
- ✓ Get the contacts (6ms)
- ✓ Update the contact (114ms)
- ✓ Delete the contact (108ms)

- ✓ Creating a new physician specialty (9ms)
- ✓ Get the created physician specialty (108ms)
- ✓ Get the physician specialties (9ms)
- ✓ Delete the physician specialty (105ms)

Test addresses routes

- ✓ Get the addresses (3ms)

Test contact routes

- ✓ Creating a new contact (5ms)
- ✓ Get the created contact (108ms)
- ✓ Get the contacts (6ms)
- ✓ Update the contact (114ms)
- ✓ Delete the contact (108ms)

Test Suites: 1 passed, 1 total

Tests: 16 passed, 16 total

Snapshots: 0 total

Time: 3.215s

Ran all test suites.

Relatórios

```
$ stryker run
```

Ran 10.41 tests per mutant on average.

File	% score	# killed	# timeout	# survived	# no cov	# error
All files	32.33	61	36	203	0	6
controllers	39.30	49	30	122	0	6
AddressController.js	8.82	2	1	31	0	1
ContactController.js	73.53	14	11	9	0	1
PhysicianController.js	50.00	12	5	17	0	1
PhysicianSpecialtyController.js	50.00	9	8	17	0	1
ProfileController.js	0.00	0	0	31	0	1
SpecialtyController.js	50.00	12	5	17	0	1
database	0.00	0	0	63	0	0
migrations	0.00	0	0	63	0	0
20200410210056_create_physicians.js	0.00	0	0	11	0	0
20200410210541_create_contacts.js	0.00	0	0	12	0	0
20200410210640_create_addresses.js	0.00	0	0	17	0	0
20200410210742_create_specialties.js	0.00	0	0	9	0	0
20200410210844_create_physician_specialties.js	0.00	0	0	14	0	0
routes	56.25	12	6	14	0	0
index.js	56.25	12	6	14	0	0
app.js	0.00	0	0	2	0	0

4 Tipos de mutantes

Morto:



Sobrevivente:



Tempo limite excedido:



Erro de *runtime*:





File / Directory	Mutation score	# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
controllers	39.30	49	122	30	0	0	6	0	79	122	207
JS AddressController.js	8.82	2	31	1	0	0	1	0	3	31	35
JS ContactController.js	73.53	14	9	11	0	0	1	0	25	9	35
JS PhysicianController.js	50.00	12	17	5	0	0	1	0	17	17	35
JS PhysicianSpecialtyController.js	50.00	9	17	8	0	0	1	0	17	17	35
JS ProfileController.js	0.00	0	31	0	0	0	1	0	0	31	32
JS SpecialtyController.js	50.00	12	17	5	0	0	1	0	17	17	35

```

32     const result = await connection('contacts')
33         .where(53 'id', id)
34         .update(54 {
35             physicianId,
36             type,
37             contact
38         })
39
40     if (56 55 true result === 0) 58 {} {
41         return response.status(406).json({ error: 60 'contact not updated!'
42     })
43
44     return response.status(200).json({ id, physicianId, type, contact
45     }),
46     async destroy(request, response) {
47         const { id } = request.params
48
49         const result = await connection('contacts').where('id', id).delete()
50
51         if (67 result === 0) 69 {
52             return response.status(406).json(70 { error: 71 'contact not found!'

```

BlockStatement

👽 Survived

```

25 },
26   async update(request, response) 121 {
27     const { id } = request.params
28
29     const { physicianId, specialtiesId } = request.body;
30
31     const physicianSpecialty = await connection(122 ""'physician_specialti
32       .where(123 ""'id', id)
33       .update(124 {
34         physicianId,
35         specialtiesId
36       })
37
38     if (125 126 127 physician_specialty === 0) 128 {
39       return response.status(406).json(129 { error: 130 'physician_specia
40     }
41
42     return response.status(200).json(131 { msg: 132 { id, physicianId, spec
43   },
44   async destroy(request, response) 133 {
45     const { id } = request.params

```

StringLiteral

👽 Survived

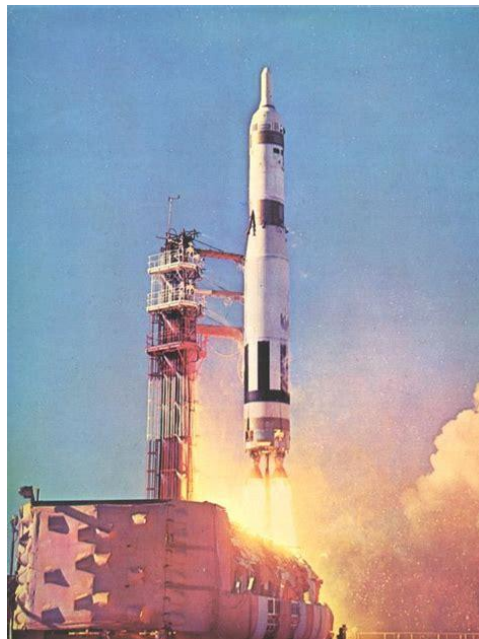
Pontos negativos e exigências

- Dificuldade em automatizar a aplicação da técnica em testes a nível de integração
- Com o aumento da *codebase*, surge o problema do teste demandar muito tempo de execução
- Recomenda-se que a análise de teste de mutação seja incluída em algum mecanismo de CD/CI
- E mais importante...

Cuidado

- Não execute o Stryker em códigos que façam coisas do tipo:

```
If (!isSafeToLaunch) {  
    launchTheRocket();  
}
```



Referências

MUTATION TESTING: CASE STUDIES ON SPRING BOOT APIS. Spring Developer, out. 2019. Disponível em: <https://www.youtube.com/watch?v=88fDcPurp-Y>. Acesso em: 9 novembro 2020.

STRYKER: GETTING STARTED. Stryker mutator. Disponível em: <https://stryker-mutator.io/docs/stryker/getting-started>. Acesso em: 9 novembro 2020.

Obrigado

Repositório com os exemplos da apresentação:

<https://github.com/lbtoma/ce-237-mutation-test-demo>