ANDREY VASCONCELOS CHAVES

UM ESTUDO SOBRE O CONHECIMENTO DOS PROFISSIONAIS A RESPEITO DE COBERTURA DE TESTE E TESTE DE MUTAÇÃO

(*versão pré-defesa, compilada em 19 de maio de 2022*)

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Andrey Pimentel.

CURITIBA PR

2022

RESUMO

Esta monografia tem como objetivo discutir que algumas técnicas de cobertura de teste não são suficientes para garantir a qualidade do software, especificamente a cobertura por linha e cobertura por ramificação, que são as mais utilizadas. Por isso, precisamos de outra alternativa para garantir que os testes serão altamente capazes de detectar falhas no código fonte do projeto. Neste trabalho iremos abordar o Teste de Mutação como técnica para ajudar neste problema. A discussão foi realizada através da apresentação do tema para um grupo de profissionais da área de engenharia de software. Após a apresentação, os entrevistados responderam um formulário referente à utilização de cobertura de teste e teste de mutação. Notou-se que, 80% dos entrevistados não utilizam cobertura de testes e concordam que este método não apresenta confiabilidade. Por conta disso, 100% dos profissionais concordam que o teste de mutação serve como um complemento para a cobertura, mas 60% não conhecia essa técnica.

Palavras-chave: Cobertura de teste. Teste de Mutação. Cobertura por linhas. Cobertura por ramificação.

ABSTRACT

This work aims to discuss that some test coverage techniques are not enough to guarantee the quality of the software, specifically line coverage and branch coverage, which are the most used. Therefore, we need another alternative to ensure that the tests will be highly capable of detecting flaws in the project's source code. In this work we will approach the Mutation Test as a technique to help in this problem. The discussion took place through presentation of the theme to a group of professionals in the area of software engineering. After the presentation, interviewees answered a form regarding the use of test coverage and mutation testing. It was noted that 80% of respondents do not use test coverage and agree that this method is not reliable. Because of this, 100% of professionals agree that mutation testing serves as a complement to coverage, but 60% did not know this technique.

Keywords: Test coverage. Mutation test. Line coverage. Branch coverage.

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

O teste de software está desempenhando um papel crucial e fundamental no desenvolvimento de software moderno. Embora os testes de software sejam conceitualmente simples — eles são compostos de duas partes principais: entradas que são usadas para executar o programa em teste e um oráculo que é usado para verificar se a execução induzida pelas entradas produz os resultados esperados — eles são muitas vezes difíceis de escrever na prática. A comunidade de pesquisa de engenharia de software forneceu muitas técnicas que podem ajudar os desenvolvedores a determinar se eles escreveram testes eficazes e eficientes, incluindo várias métricas de cobertura que foram amplamente adotadas. Embora tenham se mostrado bem-sucedidos na prática, muitos estudos de acompanhamento mostram que ainda há muito a melhorar nas medições de qualidade dos testes.

1.1 PROBLEMA

Uma dúvida comum no meio da engenharia de software é se a cobertura de teste importa. Embora os pesquisadores ainda não tenham encontrado um número mágico de cobertura que se deva buscar, eles têm encontrado evidências interessantes apontando para os benefícios da realização de testes com maior cobertura.

Em 1994, dentro do domínio limitado de experimentos, conjuntos de teste que atingiram níveis de cobertura acima de 90% geralmente mostraram detecção de falhas significativamente melhor do que conjuntos de teste do mesmo tamanho escolhidos aleatoriamente. Além disso, melhorias significativas na eficácia dos testes baseados em cobertura geralmente ocorreram quando a cobertura aumentou de 90% para 100%. No entanto, os resultados também indicam que 100% de cobertura de código por si só não é um indicador confiável da eficácia de um conjunto de testes (Hutchins et al., 1994).

1.2 OBJETIVO

O objetivo principal deste trabalho consiste em auxiliar os programadores a desenvolver e entregar código com mais qualidade, com foco em reduzir a quantidade de *bugs* aumentando a capacidade dos testes de detectar falhas.

Os objetivos específicos são:

- Recolher o conhecimento de cobertura de teste dos programadores;
- Apresentar o método Teste de Mutação;
- Sugerir o uso de Teste de mutação em conjunto com a cobertura de teste;

- Validar o conhecimento dos desenvolvedores sobre a solução sugerida.

1.3 JUSTIFICATIVA

As empresas e os programadores que dependem totalmente de uma alta cobertura de teste como métrica de qualidade de código, podem ser prejudicadas, pois isso não garante a inexistência de bugs e pode não detectar futuras falhas, fazendo com que a empresa perca recursos, pois precisará alocar um desenvolvedor para resolver essas pendências, e isso poderia ser evitado com o uso correto de cobertura em conjunto com o teste de mutação. Além disso, na procura de uma alta cobertura de teste, a empresa pode perder tempo e recurso novamente, pois os desenvolvedores investiriam mais tempo escrevendo testes apenas para atingir o objetivo, e não garante que os testes vão ser de qualidade.

Os profissionais que utilizam essa abordagem, podem sofrer consequências em seu ambiente de trabalho, pois, o código desenvolvido por ele pode gerar mais falhas e bugs. Em decorrência disso, pode haver situações que façam com que o profissional perca o respeito entre sua equipe de trabalho e não seja promovido. E como consequência mais extrema, a demissão do cargo, dependendo da gravidade e da frequência das falhas.

1.4 ORGANIZAÇÃO

O presente trabalho organiza-se da seguinte maneira: esse capítulo trouxe uma introdução geral ao documento, com uma apresentação do problema, objetivos e justificativa; o próximo capítulo contém uma descrição de conceitos básicos necessários para o entendimento de nossa abordagem; o capítulo 3 apresenta os trabalhos correlatos envolvendo cobertura de teste; o quarto capítulo descreve a abordagem do trabalho; a conclusão e trabalhos futuros se encontram no capítulo 5.

2 CONCEITOS BÁSICOS

Cobertura de teste quer dizer a quantidade ou porcentagem de código de produção que é executada pelos testes. Muitas métricas diferentes podem ser usadas para calcular a cobertura de teste (Wikipedia, 2022). Neste capítulo serão apresentadas algumas das mais básicas técnicas de cobertura, que são a porcentagem de ramos executados e a porcentagem de linhas de programa chamadas durante a execução do conjunto de testes. Depois serão apresentados conceitos sobre teste de mutação.

2.1 COBERTURA DE LINHAS

Se baseia na quantidade de linhas de código que foram executadas até completar os testes. Exemplo:

O programa recebe o número de pontos de dois jogadores de 21. O código deve devolver o número de pontos do vencedor. No 21, quem chegar mais perto de 21 pontos vence. Se um jogador ultrapassar 21 pontos, o jogador perde. Se ambos os jogadores perderem, o programa deve retornar 0.

```
1  public class BlackJack {
2    public int play(int left, int right) {
3  1. int ln = left;
4  2. int rn = right;
5  3. if (ln > 21)
6  4.   ln = 0;
7  5. if (rn > 21)
8  6.   rn = 0;
9  7. if (ln > rn)
10 8.   return ln;
11 9. else
12 10.  return rn;
13   }
14 }
```

Agora os dois casos de teste para esse código:

```
1  public class BlackJackTests {
2    @Test
3    void bothPlayersGoTooHigh() {
4      int result = new BlackJack().play(30, 30);
5      assertThat(result).isEqualTo(0);
6    }
7
8    @Test
9    void leftPlayerWins() {
```

```
10      int result = new BlackJack().play(10, 9);
11      assertThat(result).isEqualTo(10);
12    }
13  }
```

O primeiro teste executa as linhas 1-7, 9 e 10, pois ambos os valores são superiores a 21. Isso significa que, após o teste bothPlayersGoTooHigh, 9 das 10 linhas são cobertas. Assim, a cobertura de linha é 9/10 * 100% = 90% A linha 8 é, portanto, a única linha que o primeiro teste não cobre. O segundo teste, leftPlayerWins, complementa o primeiro teste e executa as linhas 1-3, 5, 7 e 8. Ambos os testes juntos agora atingem uma cobertura de linha de 100%, pois juntos cobrem todas as 10 linhas diferentes do programa.

2.1.1 Problema da cobertura por linhas

Usar linhas de código como forma de determinar a cobertura de linha é uma ideia simples e direta. No entanto, contar as linhas cobertas nem sempre é uma boa forma de calcular a cobertura. O número de linhas em um pedaço de código depende das decisões tomadas pelo programador que escreve o código.

Vejamos novamente o exemplo do 21. O método play também pode ser escrito em 6 linhas, em vez de 10:

```
1  public int play(int left, int right) {
2  1. int ln = left;
3  2. int rn = right;
4  3. if (ln > 21) ln = 0;
5  4. if (rn > 21) rn = 0;
6  5. if (ln > rn) return ln;
7  6. else return rn;
8  }
```

O teste leftPlayerWins cobriu 6/10 linhas na implementação anterior do método play. Nesta nova implementação, abrange as linhas 1-5, ou 5/6 linhas. A cobertura da linha subiu de 60% para 83%, testando o mesmo método com a mesma entrada.

Isso exige uma melhor representação do código-fonte. Um que seja independente dos estilos de código pessoais dos desenvolvedores.

2.2 COBERTURA POR RAMIFICAÇÃO

Programas complexos geralmente dependem de muitas condições complexas (por exemplo, se as instruções são compostas por muitas condições). Ao testar esses programas, almejar 100% de cobertura de linha ou bloco pode não ser suficiente para cobrir todos os casos que desejamos. Precisamos de um critério mais forte.

A cobertura de ramificação (ou cobertura de decisão) funciona de forma semelhante à cobertura de linha, exceto com a cobertura de ramificação contamos (ou visamos cobrir) todos os resultados de decisão possíveis.

Um conjunto de testes alcançará 100% de cobertura de ramificação quando os testes exercitarem todos os resultados possíveis dos blocos de decisão.

Por exemplo, vamos visar 100% de cobertura de ramificação para o código abaixo:

```
1 public String helloOrBye(int a, int b){
2    if ((a > 0) && (b >0))
3       return new String("Hello");
4    else
5       return new String("Bye");
6 }
```

Casos de testes que devemos cobrir:

- a = 1,b = 1: Neste caso, 'a' e 'b' ambos são verdadeiros, então a condição If é executada.
- a = 1,b = 0: Nesse caso, 'a' é verdadeiro e 'b' seria falso, então a parte Else do código é executada.

```
1
2 public class HelloOrByeTests {
3   @Test
4   void hello() {
5
6     String str = helloOrBye(1,1);
7
8     assertEquals("Hello", str);
9   }
10
11  @Test
12  void bye() {
13
14    String str = helloOrBye(1,0);
15
16    assertEquals("Bye", str);
17  }
18 }
```

Como sabemos, o objetivo da Cobertura de ramificação é fazer com que cada ramo seja executado pelo menos uma vez e esse objetivo é alcançado, ou seja, os dois testes juntos atingem uma cobertura de ramificação/decisão de 100%.

2.3 LIMITAÇÕES DA COBERTURA DE TESTE

Técnicas de cobertura de teste por si só podem não ser suficientes para determinar a qualidade dos casos de teste. Na prática, os testes podem cobrir grandes partes do sistema, e não testar nada. Por exemplo:

```
1  public class Division {
2    public static int[] getValues(int a, int b) {
3      if (b == 0) {
4        return null;
5      }
6      int quotient = a / b;
7      int remainder = a % b;
8
9      return new int[] {quotient, remainder};
10   }
11 }
```

Com os seguintes testes:

```
1  @Test
2  public void testGetValues() {
3    int[] values = Division.getValues(1, 1);
4  }
5
6  @Test
7  public void testZero() {
8    int[] values = Division.getValues(1, 0);
9  }
```

Esses testes dão uma cobertura de 100% no código, porém nunca irão falhar porque não estão validando nada (não tem os *asserts*). A função *assert* testa se uma expressão é verdadeira. Ao executar, se a expressão for verdadeira, o programa continua executando normalmente. Entretanto, se a expressão for falsa, o programa será interrompido e uma mensagem de erro será exibida (de Educação Tutorial, 2020).

2.4 CAPACIDADE DE DETECÇÃO DE FALHAS

A capacidade de detecção de falhas indica a capacidade do teste de revelar falhas no sistema em teste. Quanto mais falhas um teste puder detectar ou, em outras palavras, quanto mais falhas um teste falhar, maior será sua capacidade de detecção de falhas. Usando esse critério, podemos indicar a qualidade do nosso conjunto de testes de uma maneira melhor do que apenas com as métricas de cobertura que temos até agora.

A capacidade de detecção de falhas não considera apenas a quantidade de código de produção executado, mas também as assertivas feitas nos casos de teste. Para que um teste seja

adequado de acordo com este critério, ele deve ter um oráculo de teste significativo (ou seja, afirmações significativas).

A capacidade de detecção de falhas, como critério de adequação do teste, é a ideia fundamental por trás do teste de mutação . Nos testes de mutação, alteramos pequenas partes do código e verificamos se os testes conseguem encontrar a falha introduzida.

No exemplo anterior, criamos uma suíte de testes que não era adequada em termos de capacidade de detecção de falhas, pois não havia assertivas e, portanto, os testes nunca encontrariam falhas no código.

Testes com assertivas verificam se o resultado do método é o que esperamos. Isso significa que temos um oráculo de teste agora.

```
@Test
public void testGetValues() {
  int[] values = Division.getValues(1, 1);
  assertEquals(1, values[0]);
  assertEquals(0, values[1]);
}

@Test
public void testZero() {
  int[] values = Division.getValues(1, 0);
  assertNull(values);
}
```

Para ver como os valores em um caso de teste influenciam a capacidade de detecção de falhas, vamos criar dois testes, onde o denominador não é 0.

```
@Test
public void testGetValuesOnes() {
  int[] values = Division.getValues(1, 1);
  assertEquals(1, values[0]);
  assertEquals(0, values[1]);
}

@Test
public void testGetValuesDifferent() {
  int[] values = Division.getValues(3, 2);
  assertEquals(1, values[0]);
  assertEquals(1, values[1]);
}
```

Para a capacidade de detecção de falhas, queremos ver se os testes detectam alguma falha no código. Isso significa que temos que voltar ao código-fonte e introduzir um erro. Substituímos a divisão por uma multiplicação: um bug claro.

```
public class Division {
  public static int[] getValues(int a, int b) {
```

```
3     if (b == 0) {
4       return null;
5     }
6     int quotient = a * b; // the bug was introduced here
7     int remainder = a % b;
8
9     return new int[] {quotient, remainder};
10   }
11 }
```

Se executarmos nossos testes com o código bugado, veremos que o teste com os valores 1 e 1 (o testGetValuesOnes()) ainda passa, mas o outro teste (o testGetValuesDifferent()) falha. Isso indica que o segundo teste tem uma capacidade de detecção de falhas mais alta.

Embora ambos os testes exerçam o método da mesma forma e executem as mesmas linhas, vemos uma diferença na capacidade de detecção de falhas. Isso ocorre devido aos diferentes valores de entrada para o método e aos diferentes oráculos de teste (asserções) nos testes. Nesse caso, os valores de entrada e o oráculo de teste do testGetValuesDifferent() podem detectar melhor o bug.

2.5 TESTE DE MUTAÇÃO

A ideia do teste de mutação é avaliar a qualidade do conjunto de testes . Isso é feito manipulando o código-fonte e executando os testes com esse código-fonte manipulado. Se tivermos um bom conjunto de testes, pelo menos um dos testes falhará neste código alterado (ou seja, com erros). Seguindo este procedimento, temos uma noção da capacidade de erro de falha do nosso conjunto de testes.

No teste de mutação, usamos mutantes. Mutantes são programas modificados que contêm os defeitos ou falhas que introduzimos no código-fonte para serem usados para determinar a qualidade do conjunto de testes.

2.5.1 Operadores

Um operador de mutação é uma regra gramatical que pode ser usada para introduzir uma mudança sintática. Isso significa que, se o gerador vê uma instrução no código que corresponde à regra gramatical do operador (por exemplo, a + b), o operador de mutação específica como alterar essa instrução com uma mudança sintática (por exemplo, transformando em a - b).

Aqui estão exemplos para cada um dos operadores de mutação com primeiro o código original e, em seguida, um mutante que pode ser produzido aplicando o operador de mutação:

- Substituição do operador aritmético

  Original:

```
1 int c = a + b;
```

Exemplo de mutante:

```
int c = a - b;
```

- Substituição de operador relacional

Original:

```
if (c == 0) {
  return -1;
}
```

Exemplo de mutante:

```
if (c > 0) {
  return -1;
}
```

- Substituição de operador condicional

Original:

```
if (a == null || a.length == 0) {
  return new int[0];
}
```

Exemplo de mutante:

```
if (a == null && a.length == 0) {
  return new int[0];
}
```

- Substituição do operador de atribuição

Original:

```
c = a + b;
```

Exemplo de mutante:

```
c -= a + b;
```

2.5.2 Análise do Teste de Mutação

O objetivo é usar o teste de mutação para determinar a qualidade do conjunto de testes. Como vimos antes, os mutantes precisam ser criados primeiro e é melhor fazer isso de forma automatizada com a ajuda de operadores de mutação. Em seguida, executamos o conjunto de testes em cada um dos mutantes com um mecanismo de execução. Se algum teste falhar quando executado contra um mutante, dizemos que o conjunto de testes matou o mutante. Isso é bom,

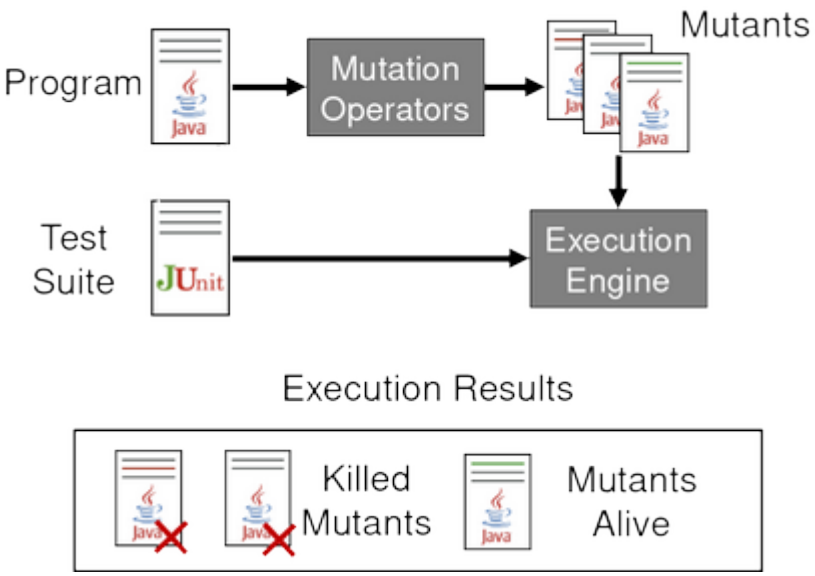

Figura 2.1: Processo teste de mutação.
Fonte: Aniche (2020)

porque mostra que o conjunto de testes tem alguma capacidade de detecção de falhas. Se nenhum dos testes falhar, o mutante permanece vivo. A figura 2.1 mostra como funciona o processo.

Ao realizar o teste de mutação, é contabilizado o número de mutantes que o conjunto de testes matou e o número de mutantes que ainda estão vivos. Ao contar o número de cada um desses grupos mutantes, podemos dar um valor à qualidade do conjunto de testes.

Avaliar a qualidade de um conjunto de testes computando sua pontuação de mutação é chamado de análise de mutação. O teste de mutação significa usar essa análise de mutação para melhorar a qualidade do conjunto de testes adicionando e/ou alterando casos de teste.

Esses conceitos estão altamente interligados. Antes que o teste de mutação seja realizado, deve-se calcular a pontuação da mutação. Isso indica se o conjunto de testes deve ser alterado. Se a pontuação da mutação for baixa, há muitos mutantes que não são mortos pelo conjunto de testes. Então esse conjunto de testes precisa ser melhorado.

2.6 CONCLUSÃO

Esse capítulo apresentou conceitos relevantes para o entendimento do nosso estudo. Foi apresentado duas técnicas de cobertura de teste: cobertura por linhas e cobertura por ramificação. Logo em seguida foi apresentado alguns problemas envolvendo essas técnicas. Por último foi apresentado o conceito de teste de mutação, e como funciona o processo para determinar a qualidade do conjunto de testes utilizando essa técnica.

3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

O processo de mapeamento sistemático seguiu as orientações de Kitchenham e Charters (2007) para a elaboração do estudo. Este plano especifica expressões de busca, estratégias de busca, mecanismos de busca, critérios de inclusão/exclusão de estudos e sistematização, análise e síntese de dados.

3.1 QUESTÕES DE PESQUISA

Este mapeamento sistemático mira em identificar estudos relacionados a métricas de qualidade de testes, e foi feito para prover respostas para as seguintes perguntas de pesquisa:

- Q1:Quais métricas os estudos usam para saber a qualidade de um teste? Resultados esperados: saber métricas são usadas para medir a qualidade de um teste;
- Q2: Quais ferramentas e técnicas de testes são consideradas mais eficientes pelos artigos? Resultados esperados: verificar as técnicas mais eficientes;
- Q3: Quais métricas os artigos usam para saber a cobertura de testes de um software? Resultados esperados: identificar qual tipo de cobertura de teste é mais utilizado;
- Q4: Os estudos consideram a cobertura de testes suficiente para medir a qualidade de um software? Resultados esperados: verificar se a cobertura de teste por si só é confiável como métrica de qualidade dos testes;
- Q5: Como os estudos garantem que um software está bem testado? Resultados esperados: verificar se as métricas de testes influenciam na qualidade do software;

3.2 PROCESSO DE BUSCA

A pesquisa foi planejada para cobrir o maior número de estudos sobre qualidade de teste de software. Assim, as expressões de pesquisa apresentadas a seguir foram selecionadas e calibradas. A biblioteca digital do Google Scholar foi escolhida para realizar a pesquisa, porque aparece como um dos principais fontes de pesquisa e como as bibliotecas digitais mais utilizadas para estudos sistemáticos.

Sobre a string de busca, foram utilizados os termos *software testing, software quality, test quality, test coverage, mutation test, .* As strings de busca foram utilizadas na biblioteca digital Google Scholar, da seguinte forma:

software testing AND ((“software quality” AND "test quality") OR (“test coverage” AND "test quality") OR (“mutation test” AND "test quality") OR ("test quality"AND “test coverage”))

3.3 CRITÉRIOS DE INCLUSÃO E EXCLUSÃO

Na pesquisa, foram encontrados 671 artigos, e depois foram filtrados utilizando os critérios de inclusão e exclusão a seguir.

- **CI1**:O estudo define e/ou apresenta aspectos visuais voltados à qualidade de teste em projetos de software.
- **CI2**: O estudo investiga, compara ou avalia os métodos propostos para métricas de qualidade/cobertura de teste em projetos de software.
- **CI3**: O estudo analisa a aplicação de métodos voltados para a qualidade/cobertura de testes em um projeto de software
- **CE1**: Não aborda qualidade de teste em projetos de desenvolvimento de software.
- **CE2**: Apresenta somente trabalhos futuros
- **CE3**: Não apresente qualquer tipo de investigação, comparação, avaliação ou aplicação de métodos voltados para qualidade/cobertura de testes em projetos de software.
- **CE4**: É duplicado de algum estudo já selecionado
- **CE5**: O artigo não é acessado gratuitamente
- **CE6**: Não está escrito em ingles ou portugues
- **CE7**: Foi publicado antes de 2016
- **CE8**: Mapeamento sistemático da literatura
- **CE9**: Literatura cinzenta, livros e teses

Com base na leitura do título e considerando os filtros de exclusão, foram eliminados 594 artigos, restando 67. Com base na leitura do resumo e considerando os filtros de exclusão, foram eliminados 52 artigos, restando 15 artigos. Por fim, lendo todo o artigo e considerando os filtros de exclusão, foram eliminados 8 artigos, restando apenas 7 artigos: (Buffardi et al., 2019), (Chekam et al., 2017), (Grano et al., 2020), (Vidács et al., 2016) , (Tengeri et al., 2016), (Delgado-Pérez et al., 2018), (Parsai e Demeyer, 2020)

3.4 RESULTADOS

3.4.1 Q1 Métricas qualidade de teste

As métricas utilizadas pelos artigos foram as seguintes: *test accuracy, bug identification, code coverage, fault revelation, readability and understandability of tests, coverage efficiency,*

partitioning efficiency, specialization metric e uniqueness metric, mutation score, test reducibility, statement coverage and branch coverage.

A grande maioria dos artigos (71,4%), utilizaram alguma técnica de cobertura de teste com o intuito de medir a qualidade dos testes. As técnicas de cobertura mais utilizadas foram cobertura por linhas (*statement coverage*) e cobertura por ramificação (*branch coverage*).

Quatro artigos (57,1%) utilizaram teste de mutação (*mutation score*) ou capacidade de detectar falhas(*fault revelation*) como principal métrica de qualidade de teste.

3.4.2 Q2 Métricas mais eficientes

Nenhum dos artigos consideraram cobertura de teste sendo a mais eficiente, a maioria (85,7%) aponta o teste de mutação como mais eficiente, pois tem mais capacidade de detectar falhas.

No estudo de Delgado-Pérez et al. (2018), os resultados mostraram que mesmo depois de atingir uma alta quantidade de cobertura de testes e apesar de atender a um critério-chave de cobertura de teste especificado em um padrão primário de segurança, e apesar de atender às rigorosas expectativas da indústria nuclear. O teste de mutação ainda se faz necessário para complementar a cobertura de teste.

3.4.3 Q3 Técnicas de cobertura de teste

Como esperado, todos os artigos utilizaram cobertura por linhas ou cobertura por ramificação como forma de medir a quantidade de código de produção que é exercida pelos testes. 57% utilizaram cobertura por ramificação e 43% utilizaram cobertura por linhas.

3.4.4 Q4 Eficiência da cobertura de teste

Nenhum estudo considerou a cobertura de teste suficiente para melhorar a qualidade dos testes. Todos tiveram que utilizar uma técnica complementar para garantir a eficiência dos testes.

Mesmo com 100% de cobertura de ramificação, ainda há a possibilidade de falhas passarem despercebidas, pois a cobertura de ramificação apenas verifica se o código de teste executa cada ramificação e não avalia se o estado do programa está infectado ou se a falha é propagada para a saída . Por isso, tais métricas de cobertura não podem revelar nada sobre a qualidade de teste (Parsai e Demeyer, 2020).

3.4.5 Q5 Software bem testado

Quanto a software com testes eficientes, Grano et al. (2020) considera que legibilidade e manutenibilidade dos testes são os principais fatores que influenciam essa métrica. Porém, 57% dos artigos consideram que a capacidade de detectar falhas e a quantidade de mutantes mortos são os principais fatores para garantir a eficiência dos testes e do projeto.

Parsai e Demeyer (2020) demonstraram que a cobertura de mutação revela fraquezas adicionais no conjunto de testes em comparação com a cobertura de ramificações e que é capaz de fazê-lo com uma sobrecarga de desempenho aceitável durante a construção do projeto.

3.5 CONCLUSÃO

Essa seção trouxe dados dos trabalhos sobre cobertura de teste, teste de mutação e qualidade de teste. Os artigos selecionados mostram um consenso na literatura em relação a cobertura de teste, inclusive temos alguns artigos que relacionam as mesmas técnicas, teste de mutação e cobertura de teste, que foram apresentadas na seção 2 Conceitos Básicos. Outro resultado interessante também, foi que a maioria dos artigos consideram o teste de mutação como uma técnica eficiente para garantir a qualidade do conjunto de testes, destaque para Parsai e Demeyer (2020), que compara diretamente as técnicas que irão ser utilizadas nesse trabalho.

4 ESTUDO

Este capítulo tem como objetivo descrever como foi feito o estudo e as entrevistas com os participantes .Dessa forma, permite-se analisar os conhecimentos dos desenvolvedores a respeito da cobertura de teste e teste de mutação.

4.1 METODOLOGIA

Esse estudo foi composto por entrevistas com desenvolvedores da área de tecnologia. Primeiro foi apresentado uma introdução sobre esse trabalho, e depois responderam um questionário envolvendo cobertura de teste e teste de mutação. As entrevistas duraram cerca de 20 minutos, e foram realizadas com 5 desenvolvedores que aceitaram participar do estudo.

4.1.1 Objetivo

O objetivo deste estudo é identificar o conhecimento dos desenvolvedores a respeito da cobertura de teste e teste de mutação. Além disso, saber se já usaram ou utilizaram alguma das métricas e técnicas abordadas nesse trabalho.

4.1.2 Procedimento

O procedimento começou com uma entrevista piloto com um voluntário mais experiente a fim de recolher sugestões de melhoria sobre a apresentação e o questionário. Depois de obter a versão final da apresentação e do formulário, foi convidado alguns companheiros de trabalho e alguns companheiros de faculdade. Aos que aceitaram participar, foi apresentado o conceito de cobertura de teste e suas técnicas, sendo elas a cobertura por linhas e a cobertura por ramificação. Depois foi apresentado o conceito de capacidade de detecção de falhas. E por fim, foi apresentado o conceito de teste de mutação. Depois da apresentação, responderam um formulário com perguntas sobre a apresentação.

4.1.3 Perfil dos entrevistados

As entrevistas foram feitas com 5 desenvolvedores, todos eles com mais de um ano de experiência no mercado de trabalho, e todos estão empregados. Os cargos atuais dos voluntários são os seguintes: Analista de Ciências de Dados, Engenheiro de Software, Desenvolvedor e Atendimento e Testes.

4.2 RESULTADOS

- Cargo atual
  Dos cinco participantes, 2 estão trabalhando como desenvolvedores, 1 está como Analista de Ciência de Dados, 1 está como Engenheiro de Software, e o último está na função de Atendimento e Testes.

- Anos de experiência
  A figura 4.1 ilustra a distribuição de anos de experiência no mercado de trabalho dos participantes. A maioria(60%) tem entre 1 e 3 anos de experiência, e 40% têm entre 3 e 5 anos de experiência. Isso mostra que o estudo foi realizado com participantes iniciantes e intermediários na área de tecnologia.

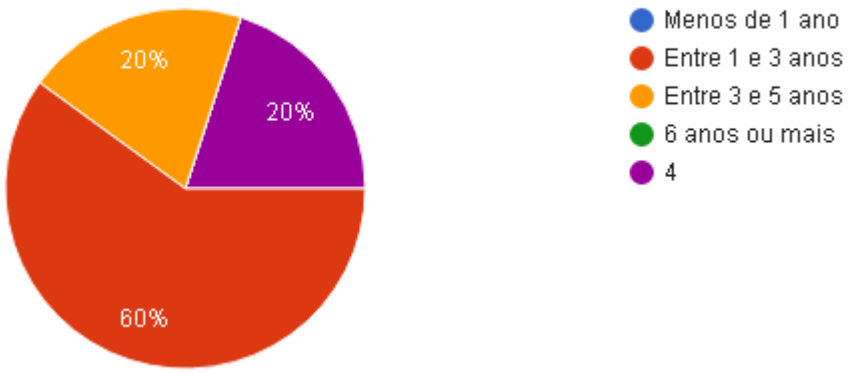

Figura 4.1: Resultados sobre anos de experiência no mercado de trabalho

- Você utiliza cobertura de testes no seu trabalho?
  A figura 4.2 mostra que nenhum dos participantes está utilizando algum método de cobertura de teste no momento. 80% nunca utilizaram essa técnica e 20% já utilizaram em outros projetos. 40% responderam que não utilizam, porém gostariam de utilizar.

Figura 4.2: Resultados sobre utilização de cobertura de teste no trabalho

- O projeto em que você trabalha tem alguma meta de cobertura de testes?
  A figura 4.3 mostra que 80% não utilizam nenhuma porcentagem como meta de cobertura de teste. Dos 4 participantes que não utilizam, apenas 1 tem interesse em utilizá-lo. E apenas 1 (20%) tem uma porcentagem de cobertura para atingir no projeto em que trabalha.

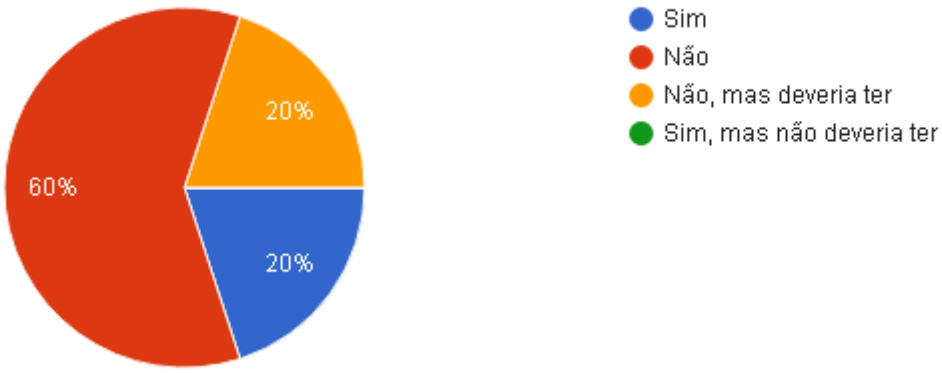

Figura 4.3: Resultados sobre meta de cobertura de teste no trabalho

- 100% de cobertura de teste garante uma boa suite de testes?
  A figura 4.4 mostra que 80% dos participantes não concordam que uma alta porcentagem de cobertura de teste garante um bom conjunto de testes, e os outros 20% concordam. Nenhum dos participantes concordaram parcialmente com a questão.

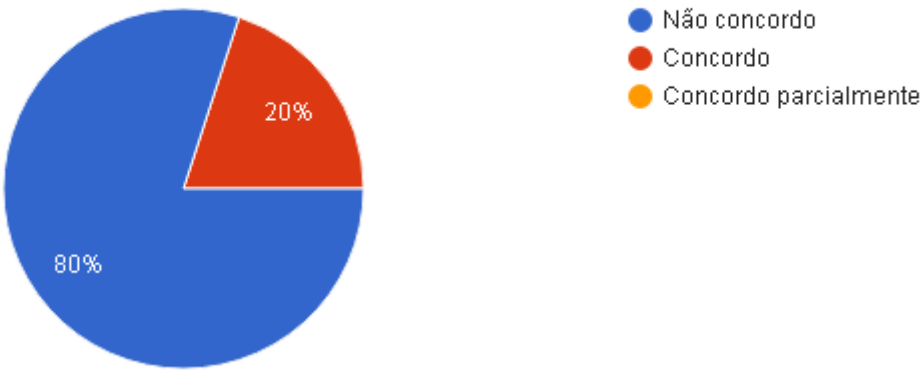

Figura 4.4: Resultados sobre alta porcentagem de cobertura de teste garantir qualidade

- O Teste de Mutação pode ajudar nos problemas de cobertura?
  A figura 4.5 mostra que todos os participantes concordaram que o teste de mutação pode

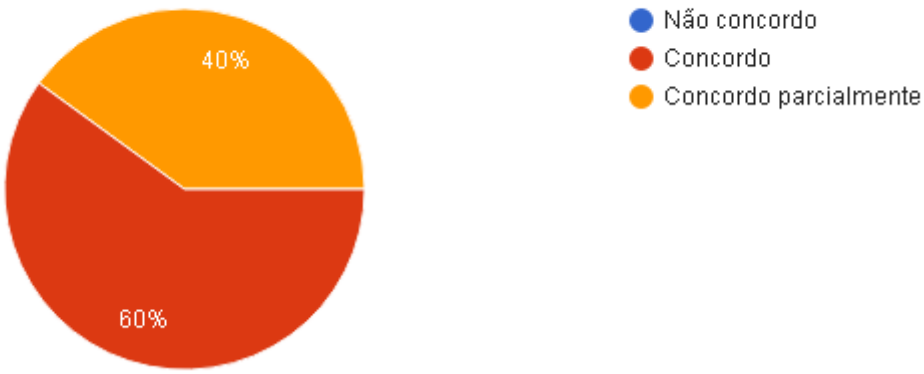

Figura 4.5: Resultados sobre a utilidade do teste de mutação

ajudar nos problemas apresentados sobre cobertura de testes, porém 40% concordaram parcialmente com esta afirmação.

- Conhecia o Teste de Mutação ou já tinha utilizado alguma vez?
  A figura 4.6 mostra que 60% dos participantes não conheciam a técnica teste de mutação. Os outros 40% já conheciam esta técnica.

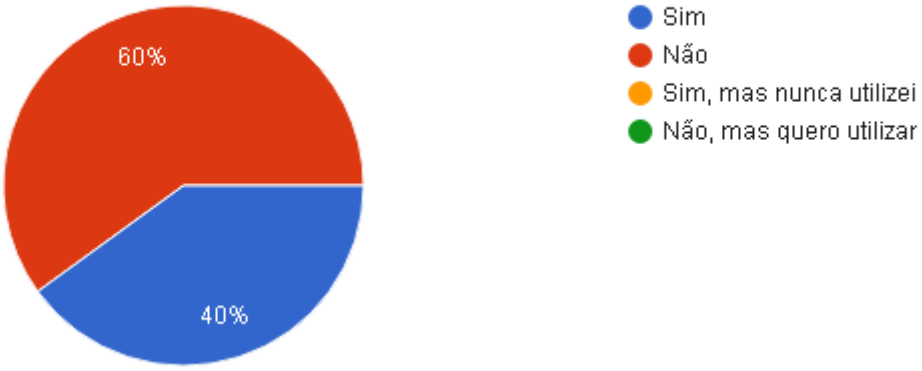

Figura 4.6: Resultados sobre conhecimento do teste de mutação

- O Teste de Mutação pode ser aplicado na sua rotina de trabalho.
  A figura 4.7 mostra que todos os participantes concordam que o teste de mutação pode ser usado no seu trabalho, embora 20% concordam parcialmente com esta questão.

- O Teste de Mutação é muito difícil de aprender e de se utilizar.
  Sobre a dificuldade do teste de mutação, a figura 4.8 mostra que 60% dos participantes não concordam com esta afirmação, e os outros 40% concordam parcialmente.

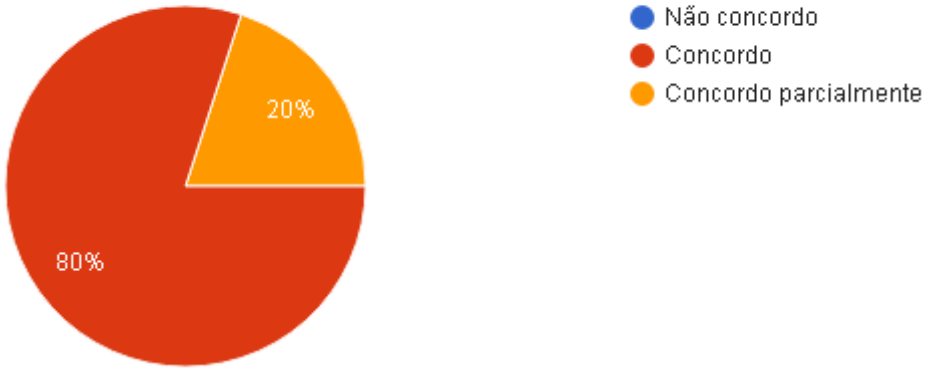

Figura 4.7: Resultados sobre aplicar o teste de mutação no trabalho

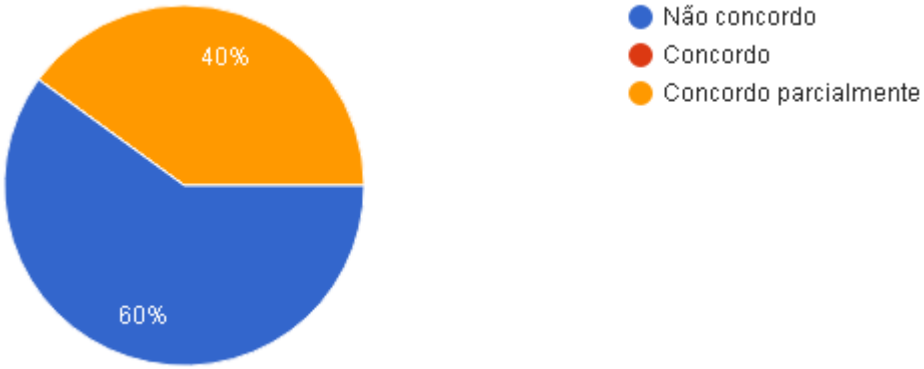

Figura 4.8: Resultados sobre dificuldade de aprender o teste de mutação

4.3 DISCUSSÃO

Nessa seção discute-se os resultados obtidos a partir das entrevistas conduzidas para o estudo. A primeira parte vai ser uma discussão sobre os resultados das questões a respeito de cobertura de teste. Na segunda parte serão discutidos os resultados sobre teste de mutação.

4.3.1 Cobertura de teste

Reunindo os resultados apresentados, percebe-se que a maioria dos participantes não utilizam e nunca utilizaram cobertura de teste no trabalho. Este é um resultado inesperado, pois esperava-se que a maioria, ou pelo menos mais de um participante, utilizasse esta técnica no trabalho ou em projetos que contribui, já que esta técnica é amplamente conhecida no meio da engenharia de software, e até mesmo lecionada em matérias com foco em qualidade de software

No entanto, mesmo sem utilizar essa técnica, ficou claro que grande parte dos participantes concordam que uma alta porcentagem de cobertura de teste não garante um conjunto de testes eficientes. Esse resultado era esperado, devido aos problemas de cobertura de teste que foram apresentados aos participantes.

A respeito de ter como objetivo uma alta porcentagem de cobertura de teste, apenas um participante teve essa experiência no trabalho, e como era esperado, não foi uma experiência positiva. Segue comentário do participante sobre esse ponto:

- "Hoje em dia não utilizo cobertura de testes no meu trabalho, mas já utilizei no trabalho anterior e a experiência não foi muito boa. Acredito que a cobertura de testes deve ser voltada a testar as partes críticas do sistema com rigor de forma a evitar futuros problemas e não correr atrás de uma meta numérica pré estabelecida."

4.3.2 Teste de mutação

Considerando os resultados obtidos na parte de teste de mutação, mais da metade dos participantes não conheciam essa técnica de teste. O que era um resultado esperado, pois não é um método muito conhecido nem muito utilizado nas faculdades e no ambiente profissional. Dos que responderam que já conheciam essa técnica, um deles teve contato através de uma matéria da faculdade, porém só utilizou em um exercício. E o outro, disse que conheceu através de um colega de trabalho que sugeriu o uso dessa ferramenta no projeto, segue comentário:

- "Concordo totalmente, inclusive estamos estudando como aplicar os testes de mutação na aplicação que trabalho hoje dia."

Esse participante foi o mesmo que comentou que não teve uma boa experiência quando utilizou meta de cobertura de teste no trabalho, isso é um indício de que quando os desenvolvedores fazem uso da cobertura de teste e conhecem na prática os seus defeitos, no futuro podem procurar mais uma técnica para usar em conjunto ou substituí-la, que no caso desse participante, foi justamente o teste de mutação. Além disso, todos os participantes concordaram que o teste de mutação pode ser utilizado no trabalho, porém um deles comentou que a escolha dos operadores, que foram apresentados na seção 2.3.1, pode ser muito custosa e muito demorada. Isso realmente é um ponto muito interessante e está sendo amplamente discutido no meio acadêmico, segundo Delgado-Pérez et al. (2019), a escolha dos operadores de mutação é um aspecto fundamental no teste de mutação para orientar o testador para um conjunto de testes eficaz. Projetar um conjunto de operadores de mutação está sujeito a uma troca entre eficácia e custo computacional: uma população de mutação maior pode descobrir mais falhas, mas levará mais tempo para analisar. Porém esse assunto será tratado na seção de trabalhos futuros.

No geral, os participantes concordam que a cobertura de teste é útil para ter uma noção da qualidade do projeto, porém ela apresenta algumas falhas e, por si só, não é confiável. Por isso, todos os participantes concordaram que o teste de mutação pode ser um grande aliado para ajudar nesses problemas, e ajudar a aumentar a qualidade do projeto e a reduzir possíveis erros. No entanto esta técnica pode ser muito custosa, por isso foi recomendado um estudo sobre uma melhor escolha de operadores para diminuir esse problema. Além disso, recomenda-se o uso

dessa técnica em classes selecionadas, e não no projeto todo, pois, dependendo do projeto, pode levar muito tempo.

4.4 CONCLUSÃO

Esse capítulo apresentou como foi realizado o estudo sobre o conhecimento dos desenvolvedores a respeito de cobertura de testes e teste de mutação, que consistiu em apresentações, seguidas de entrevistas com os participantes. Depois de conduzida a metodologia, apresentam-se os resultados obtidos por meio do questionário e em seguida realiza-se a discussão acerca dos resultados.

5 CONCLUSÃO

Este trabalho objetivou auxiliar os desenvolvedores participantes a desenvolver testes com mais qualidade. Para isso, foram apresentados problemas envolvendo a cobertura de teste, com o intuito de mostrar que essa técnica não garante a qualidade do software, e por isso, foi discutido o teste de mutação como possível melhoria nesses cenários problemáticos.

Foi realizado um mapeamento sistemático da literatura sobre esse assunto, e resultou em diversos artigos tratando desse mesmo problema, porém com diferentes abordagens, destaque para Tengeri et al. (2016) e para Parsai e Demeyer (2020), os dois artigos fazem uma relação entre cobertura de teste e teste de mutação em projetos reais, porém nenhum deles realizou entrevistas com desenvolvedores do mercado de trabalho.

Para conduzir o estudo deste trabalho, foram realizadas entrevistas com os participantes, a fim de reunir dados sobre conhecimento desse tema. Com base nesses dados e nos resultados apresentados neste trabalho, grande parte dos participantes concordaram com as questões apresentadas e demonstraram interesse em utilizar o teste de mutação em futuros projetos, portanto pode-se dizer que esse trabalho conseguiu atingir o objetivo proposto.

5.1 TRABALHOS FUTUROS

Neste trabalho foi apresentado o teste de mutação para ser usado em conjunto com a cobertura de teste, e isso pode ser muito custoso, pois o processo de recompilar o código várias vezes para gerar mutações, dependendo do tamanho do projeto, pode demorar muito tempo e consumir muito recurso. Esse é um dos principais motivos que o teste de mutação não é amplamente utilizado em processos de software, por isso a recomendação para trabalhos futuros é o estudo de como escolher operadores mais eficientes para reduzir o custo do teste de mutação.

Além disso, outro ponto interessante para trabalhos futuros, seria o estudo de como utilizar a cobertura de teste de uma maneira eficiente, e que não sobrecarregue os desenvolvedores com altas porcentagens difíceis de serem alcançadas. É claro que a cobertura pode ser útil, porém sabemos que é muito difícil ter uma cobertura de quase 100%. Por isso seria interessante um trabalho abordando essa questão.

REFERÊNCIAS

Aniche, M. (2020). Mutation testing. http://web.archive.org/web/20200805135258/https://sttp.site/chapters/intelligent-testing/mutation-testing.html. Acessado em 18/05/2022.

Buffardi, K., Valdivia, P. e Rogers, D. (2019). Measuring unit test accuracy. Em *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, página 578–584, New York, NY, USA. Association for Computing Machinery.

Chekam, T. T., Papadakis, M., Le Traon, Y. e Harman, M. (2017). An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. Em *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, páginas 597–608.

de Educação Tutorial, P. (2020). Função assert. https://petbcc.ufscar.br/assertfuncoes/. Acessado em 18/05/2022.

Delgado-Pérez, P., Habli, I., Gregory, S., Alexander, R., Clark, J. e Medina-Bulo, I. (2018). Evaluation of mutation testing in a nuclear industry case study. *IEEE Transactions on Reliability*, 67(4):1406–1419.

Delgado-Pérez, P., Rose, L. e Medina-Bulo, I. (2019). Coverage-based quality metric of mutation operators for test suite improvement. *Software Quality Journal*, 27.

Grano, G., De Iaco, C., Palomba, F. e Gall, H. C. (2020). Pizza versus pinsa: On the perception and measurability of unit test code quality. Em *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, páginas 336–347.

Hutchins, M., Foster, H., Goradia, T. e Ostrand, T. (1994). Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. Em *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, página 191–200, Washington, DC, USA. IEEE Computer Society Press.

Kitchenham, B. A. e Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Relatório Técnico EBSE 2007-001, Keele University and Durham University Joint Report.

Parsai, A. e Demeyer, S. (2020). Comparing mutation coverage against branch coverage in an industrial setting. *International Journal on Software Tools for Technology Transfer*, 22(4):365–388.

Tengeri, D., Vidács, L., Beszédes, A., Jász, J., Balogh, G., Vancsics, B. e Gyimóthy, T. (2016). Relating code coverage, mutation score and test suite reducibility to defect density. Em *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, páginas 174–179.

Vidács, L., Horváth, F., Tengeri, D. e Beszédes, A. (2016). Assessing the test suite of a large system based on code coverage, efficiency and uniqueness. Em *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 2, páginas 13–16.

Wikipedia (2022). Code coverage. `https://en.wikipedia.org/wiki/Code_coverage`. Acessado em 18/05/2022.