EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

Geração Automática de Casos de Testes de *Software:* Uma Avaliação Empírica na Utilização das Ferramentas Evosuite e Randoop

Ana Caroline Vitória de L. Bastos[1]
ana.bastos@fatec.sp.gov.br

Edson Saraiva de Almeida[1]
edson.saraiva@fatec.sp.gov.br

Automatic Generation of Software Test Cases: An Empirical Evaluation in the Use of Evosuite and Randoop Tools

Generación automática de casos de prueba de software: Una evaluación empírica en el uso de las herramientas Evosuite y Randoop

Palavras-chave:
Teste de software.
Automação de testes.
Evosuite.
Randoop.
Qualidade de software.

Keywords:
Software Testing.
Test Automation.
Evosuite.
Randoop.
Software Quality.

Palabras clave:
Pruebas de Software.
Automatización de Pruebas.
Evosuite.
Randoop.
Calidad de Software.

Enviado em:
20 novembro, 2023

Apresentado em:
05 dezembro, 2023

Publicado em:
04 outubro, 2024

Evento:
6º EnGeTec

Local do evento:
Fatec Zona Leste

Avaliadores:
Edson Company Colalto Júnior
Rodrigo Campos

Resumo:
O Teste de Software se constitui em uma das principais atividades no ciclo de vida do processo de desenvolvimento de software para controlar a qualidade. Geralmente a atividade de teste é realizada de forma manual e tem um alto custo no processo de desenvolvimento de software. Ferramentas para automação das atividades de teste de software podem reduzir o esforço e os custos associados, melhorando o processo de desenvolvimento e consequentemente a qualidade do produto. O principal objetivo deste trabalho é investigar o uso de ferramentas para geração automática de scripts de teste. Foi realizado um estudo empírico para avaliar a eficácia em revelar falhas, cobertura de código e qualidade das condições de teste comparando duas ferramentas para geração de casos de teste automatizados a *EvoSuite* e *Randoop*. As ferramentas foram submetidas ao mesmo projeto e a cobertura de código bem como a qualidade dos casos de teste foram analisados. Os resultados mostram que ainda existe a necessidade de mais pesquisas e melhorias para que ferramentas automatizadas na geração de casos de testes sejam amplamente adotadas na comunidade de desenvolvimento de software.

Abstract:
Software Testing is one of the main activities in the software development life cycle to control quality. Usually, the testing activity is performed manually and has a high cost in the software development process. Tools for automating software testing activities can reduce the effort and costs associated with improving the development process and consequently the product quality. The main objective of this work is to investigate the use of tools for automatic generation of test scripts. An empirical study was conducted to evaluate the effectiveness in revealing faults, code coverage and quality of test conditions comparing two tools for generating automated test cases: EvoSuite and Randoop. The tools were submitted to the same project and the code coverage as well as the quality of the test cases were analyzed. The results show that there is still a need for more research and improvements for automated tools in the generation of test cases to be widely adopted in the software development community.

Resumen:
Las pruebas de software son una de las principales actividades en el ciclo de vida del proceso de desarrollo de software para controlar la calidad. Generalmente, la actividad de prueba se realiza manualmente y tiene un alto costo en el proceso de desarrollo de software. Las herramientas para automatizar las actividades de pruebas de software pueden reducir el esfuerzo y los costos asociados, mejorando el proceso de desarrollo y, en consecuencia, la calidad del producto. El objetivo principal de este trabajo es investigar el uso de herramientas para la generación automática de scripts de prueba. Se realizó un estudio empírico para evaluar la efectividad en la revelación de fallas, la cobertura del código y la calidad de las condiciones de prueba mediante la comparación de dos herramientas para generar casos de prueba automatizados, EvoSuite y Randoop. Las herramientas se sometieron al mismo diseño y se analizó la cobertura de código, así como la calidad de los casos de prueba. Los resultados muestran que todavía hay necesidad de más investigación y mejoras para que las herramientas automatizadas en la generación de casos de prueba sean ampliamente adoptadas en la comunidad de desarrollo de software.

[1] FATEC Zona Leste

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

1. Introdução

O processo de desenvolvimento de software é fator decisivo na qualidade do produto entregue ao cliente. No entanto, muitas organizações de desenvolvimento de software enfrentam dificuldades na implementação de modelos de qualidade que melhorem os seus processos, especialmente a média e pequena empresa. Para se manterem competitivas, essas organizações precisam investir em melhorias contínuas em seus processos com base em evidências de que as práticas adotadas atendem às suas necessidades de negócio. Uma das práticas fundamentais para garantir a qualidade do software são os testes, que incluem a verificação e validação do comportamento do software em relação aos requisitos. Teste de software é uma atividade que exige muito esforço e custo durante o processo de desenvolvimento, geralmente é realizada manualmente, cerca de 50% do tempo de desenvolvimento é gasto em testes (CHOUDHARY; KUMAR, 2011).

É por isso que a automação de testes é crucial para qualquer empresa que segue padrões de qualidade para criar produtos, seja para uso comercial ou pessoal. Além de melhorar o processo de desenvolvimento de software e a qualidade do produto, automatizar as atividades de testes de software também pode reduzir o esforço e o envolvimento de custos (SERNA et al, 2019).

No entanto, nem todas as ferramentas automatizadas de geração de casos de teste são adequadas para todos os cenários. Portanto, é importante determinar a melhor ferramenta para cada situação, levando em consideração os padrões de qualidade e confiabilidade dos testes do software em teste.

Neste contexto surge a seguinte questão de pesquisa: Qual ferramenta entre EvoSuite e Randoop tem melhor desempenho na geração de casos de testes automatizados em termos de cobertura de código e testes de mutação?

O objetivo principal deste trabalho é comparar e avaliar o desempenho das ferramentas EvoSuite e Randoop na geração de casos de teste automatizados, usando como critérios a detecção de falhas, a cobertura de código e a qualidade das condições de teste. Para isso, os testes foram gerados automaticamente pelas duas ferramentas e submetidos à mesma aplicação. Em seguida, uma análise de cobertura de código foi realizada com a ferramenta JaCoCo e uma avaliação da qualidade das condições de teste foi feita com a ferramenta Pitest. Os resultados indicam que as ferramentas para geração automática de casos de teste ainda requerem mais pesquisas e melhorias antes que possam ser amplamente adotadas pela comunidade de desenvolvimento de software.

2. Fundamentação Teórica

2.1. Qualidade De Software

A melhoria do processo de desenvolvimento de software pode ser considerada uma vantagem competitiva para entrega de produtos com qualidade. Apesar da existência de modelos de qualidade para melhoria de processos, para maioria das organizações de desenvolvimento de software a melhoria do processo ainda é um desafio especialmente para pequenas e médias organizações. Para se manterem competitivas estas organizações devem investir na melhoria de processos obtendo evidências de que o processo de melhoria realmente atende suas necessidades de negócios (DELLAMARO et al, 2013).

Para Sommerville (2011), "o teste de software é uma parte essencial do processo de desenvolvimento de software, que deve ser realizado em todas as fases do ciclo de vida do software", desde as fases de análise e projeto até a implantação e manutenção do sistema.

Porém, o processo de teste de software é uma tarefa bastante complexa, que demanda muitos recursos humanos e materiais, além de ser propensa a erros e inconsistências. De acordo com Pressman (1995), através do processo de testes, busca-se avaliar se as funcionalidades do software estão aparentemente trabalhando de acordo com as especificações e requisitos do projeto, garantindo que o software atinja o nível de qualidade esperada pelos interessados no produto.

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

Entretanto, o teste de software também é uma atividade complexa e custosa, segundo Bastos et al (2007, p. 2) "o teste de software representa de 30% a 40% do valor total do projeto, dependendo das técnicas de teste utilizadas e da tolerância a falhas exigidas pelo projeto". Geralmente a atividade de teste é realizada de forma manual e tem um alto custo no processo de desenvolvimento, que envolve diversas etapas, desde o planejamento, o projeto, a implementação, a execução, a análise e o relatório dos testes, até a gestão e o controle da qualidade do processo e do produto. Além disso, o teste de software enfrenta vários desafios, como a cobertura de todos os cenários possíveis de uso, a dependência da qualidade dos requisitos e da especificação, a dificuldade de reproduzir e rastrear os defeitos, e a impossibilidade de garantir a ausência total de erros. (BASTOS et al, 2007)

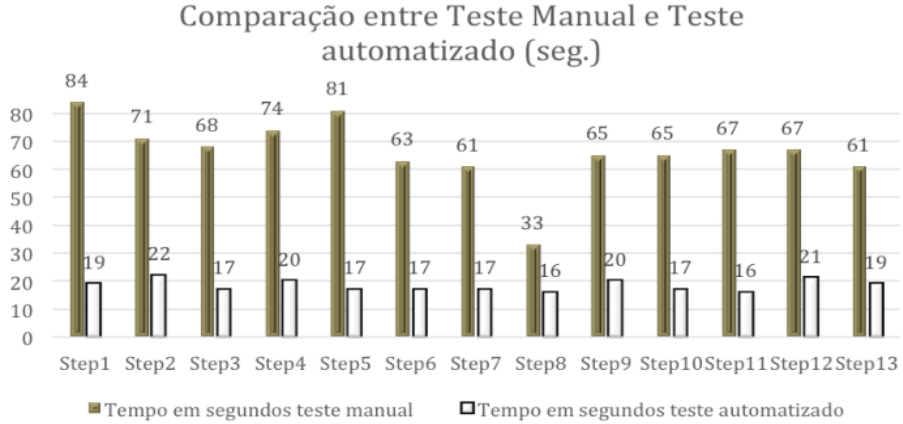
Com o objetivo de melhorar a qualidade da análise e o tempo de execução dos testes, foram criados os testes automatizados, que proporcionam a execução dos testes mais rapidamente, e com maior cobertura do software. A geração automática de casos de teste pode trazer algumas vantagens para o processo de teste de software, como a redução do esforço e do tempo necessários para criar e manter os casos de teste, a melhoria da cobertura e da qualidade dos casos de teste, a diminuição da dependência de testadores experientes e a facilitação da rastreabilidade entre os requisitos e os casos de teste (SOMMERVILLE, 2007).

De acordo com Sommerville (2011), "uma ferramenta de teste reduz a intervenção humana nos resultados obtidos, aumentando a qualidade de teste, influencia diretamente a confiabilidade do software testado", pois elimina ou minimiza os erros e as inconsistências que podem ocorrer quando o teste é realizado manualmente. Além disso, uma ferramenta de teste influencia diretamente a confiabilidade do software testado, pois permite ampliar a cobertura dos testes, detectar e corrigir mais defeitos, verificar e validar mais requisitos, e garantir que o software atenda aos padrões de qualidade estabelecidos.

Chicanelli et al, (2019), afirmam que o teste manual tem um custo muito superior ao teste automatizado, pois exige a realização de casos de teste em todos os ciclos de entrega desde a primeira fase de desenvolvimento de software. Isso implica que a automação de testes não só diminui o custo dos testes, mas também reduz o tempo de teste, amplia a cobertura de teste e aprimora a consistência dos testes.

A figura abaixo mostra o comparativo de tempo entre o teste manual e o automatizado.

Figura 1 – Comparação entre o teste manual e automatizado



FONTE: CHICANELLI et al (2019)

Neste contexto, percebe-se a importância de encontrar métodos e ferramentas automatizadas para geração de casos de teste que possam otimizar o processo de teste de software, o que pode ter um impacto positivo na produtividade no processo de teste de software, reduzindo assim o tempo normalmente investido manualmente e os recursos.

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

3. Materiais e Métodos

O estudo apresentado neste artigo utiliza a pesquisa experimental com abordagem quantitativa, este último tem como objetivo governar os dados através do uso de ferramentas e técnicas objetivas, utilizando a análise matemática da informação para produzir resultados generalizados para pesquisa (FREITAS e PRODANOV, 2013). Gil (2008) explica que o método experimental envolve submeter os objetos de estudo a variáveis particulares em condições controladas e conhecidas pelo pesquisador, com a finalidade de observar os efeitos que a variável tem sobre o objeto.

Outra metodologia realizada neste artigo é uma pesquisa bibliográfica baseada em artigos científicos, publicações e livros, com o propósito de reunir e analisar materiais já escritos sobre o assunto da pesquisa (FREITAS e PRODANOV, 2013, p. 54).

3.1. Engenharia de Software Baseada em Evidências

O estudo apresentado neste artigo é uma pesquisa que utiliza a metodologia de Engenharia de Software Baseada em Evidências (ESBE). Esta metodologia inspira-se na Medicina Baseada em Evidências, que busca combinar expertise clínica com a melhor comprovação científica possível. O seu principal objetivo é atualizar os processos de tomada de decisão relativos à aceitação de novas tecnologias no desenvolvimento e manutenção de software. Isto é adquirido através da fusão de evidências baseadas em pesquisas com experiências do mundo real (KITCHENHAM, et al., 2004).

ESBE envolve cinco etapas:

1. Transformar um problema ou informação relevante em uma pergunta.
2. Pesquisar na literatura as melhores evidências disponíveis que respondam à pergunta.
3. Avaliar criticamente a validade, o impacto e a aplicabilidade das evidências.
4. Integrar as evidências das avaliações com a experiência prática, os valores do cliente e o contexto de tomada de decisão.
5. Avaliar o desempenho e procure formas de melhorar.

Para a condução dos experimentos foram utilizadas as ferramentas EvoSuite e Randoop aplicadas ao projeto Apache Commons Net para gerar casos de testes automatizados, com o objetivo de avaliar a cobertura de código e testes de mutação utilizando a ferramenta PITest como forma de verificação de qualidade dos testes gerados.

3.2. Análise De Mutantes

A análise de mutantes tem como objetivo avaliar a qualidade de uma suíte de testes avaliando sua capacidade de detectar falhas. Os testes mutantes introduzem falhas artificiais baseadas no que se pensa serem erros reais comumente cometidos pelos programadores. Os casos de teste existente são executados em uma versão do programa (mutante), a fim de ver se algum dos casos de teste pode detectar que há uma falha. Um mutante que é detectado como tal é considerado morto, e de nenhuma utilidade. Um mutante vivo, no entanto, mostra um caso em que a suíte de testes potencialmente falha em detectar um erro e, portanto, precisa de melhoria. (ZELLER; FRASER, 2010).

O teste de mutação é considerado o padrão para avaliar a qualidade do teste porque é capaz de verificar se cada instrução do código é testada de forma significativa, ao contrário de outras métricas de cobertura que medem apenas quais partes do código são executadas pelos testes (Jia; Harman, 2011).

PITest é uma ferramenta de teste de mutação para Java. De acordo com COLES 2016, o PITest funciona gerando código de mutação que altera o bytecode do código-fonte, consequentemente alterando a lógica, remove ou até mesmo altera os valores de retorno do método. As principais vantagens do PIT são que ele é fácil de usar e bem integrado às ferramentas de desenvolvimento. Dessa forma, o código pode ser avaliado de forma mais completa, os defeitos encontrados e tornados confiáveis. (COLES et al, 2016).

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

3.3. EVOSUITE

EvoSuite é uma ferramenta de código aberto que gera conjuntos de testes unitários de classes Java a partir de bytecode e suas dependências, projetados para maximizar a cobertura do código. O EvoSuite emprega uma abordagem baseada em pesquisa na qual um algoritmo genético (GA) otimiza todo o conjunto de testes com base em critérios alvo selecionado. A vantagem desta abordagem é que ela não é afetada negativamente por objetivos inviáveis. (FRASER; ARCURI, 2011)

Para melhorar a usabilidade e a compreensão dos casos de teste resultantes, o EvoSuite implementa uma série de medidas de pós-processamento que levam à criação de suítes de testes unitários sucintos e compactos. A etapa final dessas medidas envolve a integração das asserções do teste, ou declarações que confirmam o resultado do teste. Como o EvoSuite depende apenas do bytecode como entrada, e as especificações formais estão frequentemente ausentes deste bytecode, as asserções geradas são baseadas no comportamento observado e não no comportamento pretendido (GALLEOTTI et al, 2013).

O EvoSuite é capaz de automatizar tarefas de teste de software ao produzir sistematicamente conjuntos de testes que alcançam alta cobertura de código. A ferramenta implementa diversas técnicas para alcançar seus objetivos de forma efetiva, incluindo a geração de conjuntos de testes inteiros, a otimização com relação a vários objetivos de cobertura e a geração de afirmações baseadas em mutação. (FRASER; ARCURI, 2011). A figura 2 mostra um script de caso de teste gerados automaticamente pelo Evosuite, no caso a aplicação testada foi um algoritmo de fatorial.

FIGURA 2 – Caso de teste gerado pelo Evosuite.

```
@Test(timeout = 4000)
public void test0()  throws Throwable  {
    String[] stringArray0 = new String[2];
    Fatorial.main(stringArray0);
    assertEquals(2, stringArray0.length);
}

 shilisnk
@Test(timeout = 4000)
public void test1()  throws Throwable  {
    Fatorial fatorial0 = new Fatorial();
}
}
```

Fonte: AUTORES (2023)

3.4. RANDOOP

O Randoop é uma ferramenta que usa uma técnica inspirada no teste aleatório que usa feedback de execução coletado a partir da execução de entradas de teste à medida que elas são criadas, para evitar gerar entradas redundantes e ilegais (PACHECO et al, 2007). O Randoop cria sequências de método incrementalmente, selecionando aleatoriamente uma chamada de método para aplicar e selecionando argumentos de sequências previamente construídas. O Randoop recebe como entrada um conjunto de classes sob teste, um limite de tempo e, opcionalmente, um conjunto de "verificadores de contrato" que estendem aqueles usados pelo Randoop como padrão e produz duas suítes de testes. Uma contém testes que violam contratos, que detectam bugs no código atual. A outra contém testes de regressão, que verificam se o comportamento do código não mudou após alterações (PACHECO et al, 2007).

Isso significa que o Randoop é capaz de criar sequências de chamadas de método com valores de entrada adequados para testar diferentes partes do código, bem como verificar se os valores de saída ou de estado são os esperados. O Randoop usa o mecanismo de reflexão do Java para obter informações sobre a estrutura e a interface das classes sob teste, e gera casos de teste JUnit que podem ser executados e analisados posteriormente. O Randoop geralmente gera dois tipos de testes: testes que revelam erros, que detectam bugs no código atual, e testes de regressão, que verificam se

DOI: https://doi.org/10.5281/zenodo.13367669
Daroncho e Esquisito (2024)
EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

o comportamento do código não mudou após alterações. (SMEETS; SIMONS, 2011). A figura abaixo apresenta um exemplo de teste que violam contrato e outro código de teste de regressão.

FIGURA 3 – Exemplos de testes gerados pelo Randoop.

Example error-revealing test

```
// Fails on Sun 1.5, 1.6.
public static void test1() {
   LinkedList l1 = new LinkedList();
   Object o1 = new Object();
   l1.addFirst(o1);
   TreeSet t1 = new TreeSet(l1);
   Set s1 = Collections.unmodifiableSet(t1);
   Assert.assertTrue(s1.equals(s1));
}
```

Example regression test

```
// Passes on Sun 1.5, fails on Sun 1.6 Beta 2.
public static void test2() {
   BitSet b = new BitSet();
   Assert.assertEquals(64, b.size());
   b.clone();
   Assert.assertEquals(64, b.size());
}
```

Fonte: COAGRE; LIRA (2018)

4. Resultados e Discussões

Neste artigo, foi utilizado as ferramentas EvoSuite e Randoop para gerar casos de teste automatizados para o projeto Apache Commons Net, que é um projeto disponível no GitHub, escrito em Java, que fornece componentes de rede para implementar o lado cliente de diversos protocolos básicos da Internet, tais como FTP, NNTP, SMTP, Telnet, entre outros. O projeto foi escolhido por atender aos requisitos das ferramentas de teste, foi construido com o Maven como ferramenta de gerenciamento de dependências, possui 144 classes, sendo 101 classes de código-fonte e 43 classes de teste, com um total de 24.522 linhas de código-fonte e 13.663 linhas de código de teste. A Tabela 1 mostra as características das classes de software usadas na pesquisa.

Tabela 1: Características das classes de software utilizadas na pesquisa

Classes	Número de linhas	Número de métodos
org.apache.commons.net	25364	761
org.apache.commons.net.ftp	10667	282
org.apache.commons.net.nntp	2133	69
org.apache.commons.net.smtp	1919	64
org.apache.commons.net.pop3	1338	46
org.apache.commons.net.imap	1311	44
org.apache.commons.net.telnet	1001	21
org.apache.commons.net.tftp	916	26
org.apache.commons.net.finger	318	8
org.apache.commons.net.whois	308	8
org.apache.commons.net.rexec	292	9
org.apache.commons.net.time	271	8
org.apache.commons.net.echo	160	4
org.apache.commons.net.discard	159	4
org.apache.commons.net.ntp	153	5
org.apache.commons.net.util	149	7
org.apache.commons.net.bsd	147	6
org.apache.commons.net.chargen	146	4
org.apache.commons.net.daytime	145	4
org.apache.commons.net.ftp.parser	144	5
org.apache.commons.net.io	143	5
org.apache.commons.net.smtp.util	142	4

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

Para gerar os casos de teste com as ferramentas Evosuite e Randoop, os seguintes parâmetros foram configurados:

- Evosuite: Tempo de busca de 60 segundos por classe, um tempo limite de asserção de 30 segundos, e sem ampliação de testes.
- Randoop: Tempo de geração de 60 segundos por classe, tamanho máximo de sequência de 100.

Para executar os casos de teste gerados e medir a cobertura de linha, foi utilizada a ferramenta JaCoCo uma ferramenta de análise de cobertura de código para Java, e o PITest para gerar e executar os testes mutantes.

A Tabela 2 mostra os resultados da geração de casos de teste com as ferramentas Evosuite e Randoop, incluindo o número de testes gerados, o número de linhas de código de teste, o tempo total de geração e a cobertura de linha obtida.

Tabela 2: Resultados da geração de casos de teste com Evosuite e Randoop

Ferramenta	Scripts de Testes gerados	Linhas de código de teste	Tempo de geração (min)	Cobertura de linha (%)
Evosuite	1.223	35.678	101	94,2
Randoop	1.512	19.843	96	65.4

Fonte: AUTORES (2023)

A Tabela 3 mostra os resultados da geração e execução de mutantes com a ferramenta Pitest, incluindo o número de mutantes gerados, o número de mutantes mortos, o número de mutantes vivos e a taxa de mutação obtida.

Tabela 3: Resultados da geração e execução de mutantes com Pitest

Ferramenta	Mutantes gerados	Mutantes mortos	Mutantes vivos	Taxa de mutação (%)
Evosuite	4.032	1.471	2.561	36,5
Randoop	4.032	1.701	2.331	42,2

Fonte: AUTORES (2023)

A partir dos resultados, podemos observar que a ferramenta Evosuite gerou mais casos de teste e mais linhas de código de teste comparado a ferramenta Randoop, além de obter uma maior cobertura de linha de 94,2%, isso indica que a ferramenta Evosuite obteve maior cobertura de caso de teste.

Por outro lado, o Randoop consumiu menos tempo e gerou menos código de teste, mostrando que a ferramenta foi mais simples e rápida na geração de casos de teste, porém menos abrangente e diversificada.

Em relação à geração e execução de mutantes, podemos observar que a ferramenta Pitest gerou um número de 36,5% de mutantes nos casos de testes gerados pelo Evosuite e 42,2% no Randoop. Estes números indicam que ainda há espaço para melhoria em ambas as ferramentas de geração de caso de testes, pois mesmo que ambas as ferramentas tenham uma boa cobertura de código, a falta de capacidade de detectar alterações, mostra uma baixa qualidade nos casos de testes gerados.

5. Conclusão

Neste artigo, foram comparadas as ferramentas de geração automática de casos de teste de software Evosuite e Randoop, a fim de, avaliar se desempenham em termos de cobertura de código e detecção de mutantes. Os resultados mostraram que o Evosuite obteve uma maior cobertura de código, mas uma menor capacidade de identificar mutações, sugerindo que os casos de teste gerados podem ser fracos e irrelevantes. Por outro lado, o Randoop apresentou uma menor cobertura de código, mas uma maior taxa de eliminação de mutantes, indicando que os casos de teste gerados exploraram melhor as funcionalidades e cenários do código-fonte e comparação com o Evosuite.

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

No entanto, mesmo que o Randoop tenha um desempenho melhor em matar mutantes, ambas as ferramentas apresentaram um baixo índice de mutantes detectados, o que revela que a qualidade dos casos de teste gerados pode ser insuficiente, comprometendo a manutenibilidade dos casos de teste e a qualidade do software.

Portanto, conclui-se que as ferramentas ainda precisam ser aprimoradas para gerar casos de teste mais efetivos e robustos, que possam cobrir um maior número de cenários e detectar mais alterações no código-fonte.

Referências

AMMANN, P.; OFFUTT, J. **Introduction to software testing**. 2. ed. Cambridge: Cambridge University Press, 2017.ASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. **Base de conhecimento em teste de software.** 2. ed. São Paulo: Martins, 2007.

AMMANN, P.; OFFUTT, J. **Introduction to software testing.** 2. ed. Cambridge: Cambridge University Press, 2017.

CAMPOS, J.; PANICHELLA, A.; FRASER, G. **EvoSuite at the SBST 2019 tool competition.** In: 2019 I CHICANELLI, Rachel et al. **Aspectos sociais, humanos e econômicos da utilização de testes automatizados no desenvolvimento de sistemas.** In: Décima Oitava Conferência Ibero Americana de Sistemas Cibernética e Informática. 2019.EEE/ACM 12th International Workshop on Search-Based Software Testing (SBST). IEEE, 2019. p. 29-32.

CHOUDHARY, D.; KUMAR, V. **Software testing. Journal of Computational Simulation and Modeling**, v. 1, n. 1, p. 1, 2011.

COLES, H. et al. **Pit: a practical mutation testing tool for java**. In: Proceedings of the 25th international symposium on software testing and analysis. 2016. p. 449-452.

DELAMARO, Marcio; JINO, Mario; MALDONADO, Jose**. Introdução ao teste de software**. Elsevier Brasil, 2013.

ELBERZHAGER, F., MÜNCH, J., & NHA, V. T. (2012). **A systematic mapping study on the combination of static and dynamic quality assurance techniques. Information and Software Technology.** 54(1), 1-15.

FRASER, G.; ARCURI, A. **Evosuite: automatic test suite generation for object-oriented software.** In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011. p. 416-419.

FRASER, Gordon; ARCURI, Andrea**. Evosuite at the second unit testing tool competition.** In: Future Internet Testing: First International Workshop, FITTEST 2013, Istanbul, Turkey, November 12, 2013, Revised Selected Papers 1. Springer International Publishing, 2014. p. 95-100.

FRASER, Gordon; ARCURI, Andrea. **Whole test suite generation.** IEEE Transactions on Software Engineering, v. 39, n. 2, p. 276-291, 2012.

FREITAS, E. C.; PRODANOV, C.C. **Metodologia do trabalho de pesquisa: métodos e técnicas de pesquisa e do trabalho científico**. 2 ed. Novo

GAROUSI, Vahid; MÄNTYLÄ, Mika V. **A systematic literature review of literature reviews in software testing.** Information and Software Technology, v. 80, p. 195-216, 2016.

GIL, A. C. **Métodos e técnicas do conhecimento científico**. São Paulo: Atlas, 2000.

JIA, Y.; HARMAN, M. **An analysis and survey of the development of mutation testing.** IEEE Transactions on Software Engineering, v. 37, n. 5, p. 649–678, 2011.

KITCHENHAM, B.; MENDES, E. **Software productivity measurement using multiple size measures.** IEEE Transactions on Software Engineering, v. 30, n. 12, p. 1023-1035, 2004.

EnGeTec em Revista | 09
ISSN 2965-9302
v. 1 | n. 9 | e373 | outubro 2024

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing.** 3. ed. Hoboken, NJ: John Wiley & Sons, 2011.

PACHECO, C.; LAHIRI, S. K.; ERNST, M. D.; BALL, T. **Feedback-directed random test generation**. In: Proceedings of the 29th international conference on Software Engineering. [S.l.]: IEEE Computer Society, 2007. p. 75–84.

SERNA, E. M.; MARTÍNEZ, R. M.; TAMAYO, P. O. **A review of reality of software test automation | Una revisión a la realidad de la automatización de las pruebas del software.** Computacion y Sistemas, [s. l.], v. 23, n. 1, p. 169–183, 2019.

SILVA, André et al. Flacoco: **Fault localization for java based on industry-grade coverage.** arXiv preprint arXiv:2111.12513, 2021.

SILVA, I. P. S. C.; ALVES, E. L. G.; ANDRADE, W. L**. Analyzing automatic test generation tools for refactoring validation.** In: 2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST). IEEE, 2017. p. 38-44.

SMEETS, N.; SIMONS, A. J. **Automated unit testing with Randoop, JWalk and µJava versus manual JUnit testing.** Research report, Department of Computer Science, University of Sheffield/University of Antwerp, Sheffield, Antwerp, 2011.

SOMMERVILLE, I. **Engenharia de Software.** 8ª ed. São Paulo: Pearson Education Brasil, 2007.

TORRES, D. G. Specnl**: Uma ferramenta para gerar descrições em linguagem natural a partir de especificações de casos de teste.** 2006. Dissertação de Mestrado. Universidade Federal de Pernambuco.

VERA-PÉREZ, O. L.; MONPERRUS, M.; BAUDRY, B. **Descartes: A pitest engine to detect pseudo-tested methods: Tool demonstration**. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018. p. 908-911.