Testes de mutação: uma revisão sistemática da literatura

Bruno Fernandes de Castro

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
São Carlos – SP – Brasil

brunofcastroo@gmail.com

Abstract. Context: *Tests are a vital part of a software development life cycle.* ***Objective:*** *To understand how it is possible to improve the quality of unit tests using mutation tests.* ***Method:*** *A systematic review of the literature was conducted to summarize the results of previously published studies.* ***Results:*** *Of the 19 studies identified, 6 were included, 12 were excluded and 1 study was excluded because it was duplicated.* ***Conclusions:*** *After reading the studies and summarizing the data it was possible to observe 3 key points(i) the correlation between the mutation score and the code coverage, (ii) the cost of applying the technique and (iii) how the mutation operators influence the result.*

Resumo. Contexto: *testes são uma parte vital no ciclo de desenvolvimento de um software.* ***Objetivo:*** *compreender como é possível melhorar a qualidade de testes unitários utilizando testes de mutação.* ***Metodologia:*** *uma revisão sistemática da literatura foi conduzida para sumarizar os resultados de estudos publicados anteriormente.* ***Resultados:*** *dos 19 estudos identificados, 6 foram incluídos, 12 foram excluídos e 1 estudo foi excluído por estar duplicado.* ***Conclusões:*** *após a leitura dos estudos e sumarização dos dados foi possível observar três pontos importantes(i) a correlação entre o escore de mutação e a cobertura de código, (ii) o custo de aplicação da técnica e (iii) como os operadores de mutação influenciam o resultado.*

1. Introdução

Segundo Ramler e Kaspar (2012), os testes unitários têm se tornado cada vez mais populares devido à disseminação de metodologias ágeis nos últimos anos.

A prática de testar é essencial no processo de desenvolvimento de um software de acordo com Assylberkov (2013), no entanto é impossível medir diretamente a qualidade de um conjunto de testes, já que é necessário conhecimento prévio a respeito dos defeitos que os próprios testes não conseguiram encontrar.

A cobertura de código é utilizada com frequência para julgar a eficácia do conjunto de testes, porém essa métrica apenas mede qual parte do código foi executada, não levando em consideração se o trecho de código foi testado utilizando asserções apropriadas ou se foi executado sem quaisquer asserções, como apontam Niedermayr; Juergens; Wagner (2016).

Outra forma de se avaliar a qualidade de um conjunto de testes é utilizando os chamados testes de mutação. Durante muito tempo, a análise de mutação foi considerada uma técnica excessivamente demorada e dispendiosa, no entanto, atualmente, há novos algoritmos que melhoraram consideravelmente o desempenho e o poder dos

computadores modernos tornando o processo de mutação viável, como apresenta Ramler e Kaspar (2012).

Segundo Jia e Harman (2010), testes de mutação podem ser utilizados para medir a eficácia de um conjunto de testes em termos de sua capacidade de detectar falhas.

Assim, o presente estudo tem como objetivo apresentar uma investigação utilizando o método de Revisão Sistemática da Literatura (RSL), com base nas diretrizes propostas por Kitchenham (2004), compreendendo como é possível melhorar a qualidade de testes unitários através de testes de mutação.

Este artigo encontra-se organizado no seguinte formato: na Seção 2 serão apresentados conceitos teóricos a respeito de testes unitários, cobertura de código e testes de mutação, na Seção 3 será exposto o método utilizado para conduzir a RSL, já na Seção 4 serão apresentados os resultados da RSL, na Seção 5 haverá uma discussão a partir dos resultados obtidos e, por fim, na Seção 6 serão descritas as conclusões.

2. Fundamentação teórica

Esta seção tem como objetivo introduzir os tópicos abordados por este estudo.

2.1. Teste unitário

O teste unitário é um trecho de código escrito por um desenvolvedor com finalidade de testar uma funcionalidade específica de uma pequena área do código. Geralmente testes unitários são utilizados para testar um método em particular e verificar se a sua funcionalidade está de acordo com os objetivos do desenvolvedor (HUNT; THOMAS, 2003).

2.2. Cobertura de código

Entende-se por cobertura de código à medida que pode ser obtida por alguma ferramenta que irá analisar o código do software quando um conjunto de testes for executado. De acordo com Craig e Jaskiel (2002), essa medida é utilizada para descrever qual porcentagem de linhas de código, ramos ou caminhos do software foram executados pelo conjunto de testes e facilitar ao desenvolvedor a visualização de parte do código que foi ou não foi executada.

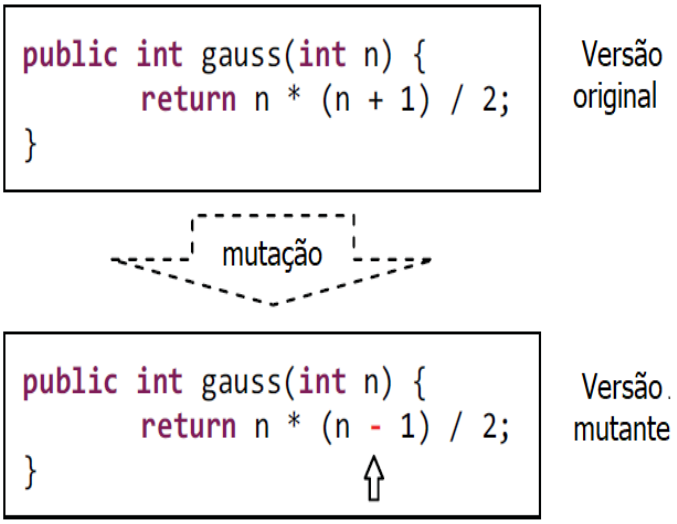
2.3. Teste de mutação

O teste de mutação é uma técnica baseada em defeitos e seu princípio geral é que esses defeitos representam os erros que os programadores cometem com frequência (JIA; HARMAN, 2010).

Um mutante é criado a partir de uma pequena mudança sintática que gera um programa defeituoso. Esse mutante pode ser utilizado para avaliar a qualidade da suíte de testes e, caso o resultado da execução do conjunto de testes seja diferente ao executar o mutante, a falha representada pelo mutante será detectada (JIA; HARMAN, 2010).

Uma consequência do teste de mutação é o escore de mutação, que é a razão entre o número de falhas detectadas e o número total de falhas (JIA; HARMAN, 2010).

Figura 1 - Exemplo de um programa mutante criado a partir da modificação de um operador aritmético (indicado pela seta)

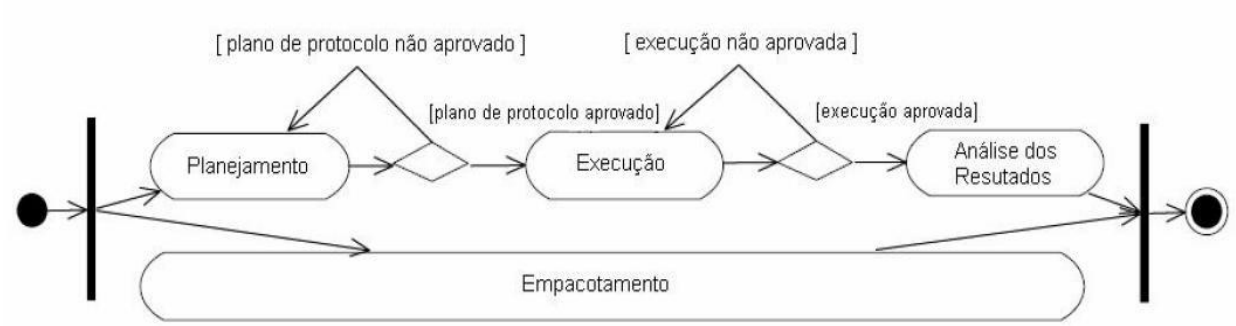


Fonte: Adaptado de Ramler e Kaspar (2012)

3. Método de pesquisa

O protocolo de pesquisa desta RSL foi conduzido utilizando as diretrizes propostas por Kitchenham (2004). A figura 2 exibe as principais etapas seguidas durante a condução da RSL.

Figura 2 - Processo de condução da RSL



Fonte: Adaptado de Biolchini et al. (2007).

3.1. Planejamento

- Questão principal: como é possível melhorar a qualidade de testes unitários utilizando testes de mutação?
- População: estudos primários da área de computação relacionados ao teste de software utilizando a técnica de mutação.
- Intervenção: eficiência dos testes de mutação.
- Resultados: compreensão de como testes de mutação podem ser usados para se avaliar a qualidade de um conjunto de testes unitários
- Contexto: desenvolvedores e testadores de software.

3.2. Execução - identificação dos estudos

Foi utilizado o repositório digital "Scopus" para definir a *string* de busca a partir das palavras-chave identificadas, refinadas e, em seguida, utilizadas para buscar por estudos primários. Os parâmetros de busca utilizados na *string* foram aplicados somente no título, palavras-chave e resumo.

Tabela 1 - Resultado da string de busca

Base	Scopus
String	(TITLE-ABS-KEY ("mutation test*" OR "mutation analysis") AND TITLE-ABS-KEY ("unit test*") AND (TITLE-ABS-KEY ("quality assurance") OR TITLE-ABS-KEY ("mutation score") OR TITLE-ABS-KEY ("code coverage") OR TITLE-ABS-KEY ("test adequacy" OR "adequacy testing") OR TITLE-ABS-KEY ("cost benefit")))
Resultados	19

3.3. Execução - seleção dos estudos

Foram aceitos estudos que satisfazem ao menos um critério de inclusão e rejeitados os estudos que satisfazem ao menos um critério de exclusão, de forma que o critério de exclusão tem prioridade sobre o de inclusão.

Os critérios utilizados para inclusão foram:

- (CI1) Ser um estudo primário sobre teste de software utilizando a técnica de mutação.
- (CI2) O estudo apresenta evidências de como o teste de mutação influencia no processo de garantia de qualidade de testes unitários.

Tabela 2 - Estudos aceitos

	Estudo	**Ano**
E1	An empirical study on the application of mutation testing for a safety-critical industrial software system	2017
E2	Investigating the correlation between mutation score and coverage score	2016
E3	Applicability and benefits of mutation analysis as an aid for unit testing	2013
E4	Should software testers use mutation analysis to augment a test set?	2012
E5	An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage	2009
E6	On guiding the augmentation of an automated test suite via mutation analysis	2009

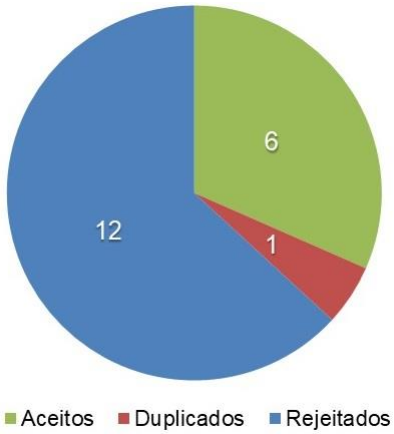
Os critérios utilizados para exclusão foram:

- (CE1) Não estar em inglês.
- (CE2) Introdução de *proceedings.*
- (CE3) O *full paper* não está disponível na web ou no COMUT da UFSCar.
- (CE4) Estudos que abordam apenas o conceito de teste de mutação, como definição de operadores de mutação, desenvolvimento ou comparação de técnicas de mutação e/ou a investigação de maneiras para resolver problemas em aberto relacionados a essa técnica.
- (CE5) O estudo não apresenta evidências de como testes de mutação influenciam nos processos de garantia de qualidade.
- (CE6) Não ser um estudo primário.

Tabela 3 - Estudos rejeitados

Estudo	Critério de exclusão
Memory mutation testing	CE4
Proceedings - 11th International Workshop on Automation of Software Test, AST 2016	CE2
Application research on program mutation technology in software testing	CE3
A survey of software testing practices in Canada	CE5
The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment	CE5
Evaluating automated unit testing in Sulu	CE5
State coverage: A structural test adequacy criterion for behavior checking	CE4, CE5
An approach to test data generation for killing multiple mutants	CE4, CE5
Unit and integration testing strategies for C programs using mutation	CE4, CE5
An experimental study on software structural testing: Deterministic versus random input generation	CE4
A systematic literature review of how mutation testing supports quality assurance processes	CE6
Will my tests tell me if I break this code?	CE5

Figura 3 - Resultado da fase de seleção



3.4. Execução - extração de dados

De cada um dos 6 estudos aprovados pelo processo de seleção, foram extraídos os dados conforme a tabela 2 para responder à questão de pesquisa.

Tabela 4 - Dados extraídos dos estudos

Tópico	Valores
Linguagem	Java, C, não especificado
Framework de mutação	Jumble, MuJava/MuClipse, Simple Jester, Javalanche, PIT, Milu ou test-analyzer
Como a análise de mutação influencia no processo de garantia de qualidade?	
Os operadores de mutação utilizados influenciaram no resultado?	
É viável incluir a análise de mutação no ciclo de desenvolvimento de um software?	

4. Resultados

Nesta seção, serão apresentados os resultados obtidos para responder à questão principal.

4.1. Como é possível melhorar a qualidade de testes unitários utilizando testes de mutação?

O estudo E1 apresenta um software crítico de grande escala (60.000 linhas de código) implementado em C com 100% de cobertura de código e conclui que, ao utilizar-se testes de mutação, foi possível identificar 2 novos defeitos e obter informações de como melhorar a suíte de testes atual, porém os problemas de escalabilidade devem ser resolvidos antes que se possa adotar esta prática a uma rotina diária, pois o tempo de execução excede os recursos computacionais disponíveis e a grande quantidade de

mutantes estaria além do escopo que os engenheiros conseguiriam analisar, também foi constatado que alguns operadores de mutação podem gerar mutantes duplicados.

Ao conduzir uma investigação para determinar se existe uma correlação entre o escore de mutação e a cobertura de código, o estudo E2 constatou um aumento significativo na cobertura de código de até 6% ao realizar a análise de mutação e que o coeficiente de correlação de Pearson e a cobertura de código é > 0.5.

O estudo E3 conclui que as ferramentas de análise de mutação atuais estão evoluindo a um ponto em que informações importantes podem ser extraídas da suíte de teste, como por exemplo, descobrir novas mutações invisíveis em casos onde a cobertura de código já está a 100%. Foi observado que os resultados são altamente dependentes do conjunto de operadores de mutação utilizado pelas ferramentas, portanto, é bem provável que o resultado de ferramentas que utilizam um pequeno conjunto de operadores de mutação seja semelhante à cobertura de código. Em relação ao custo de execução, não foi constatado nenhum custo adicional ao inserir a prática no ciclo de desenvolvimento de um software, pois as ferramentas mostraram tempo de execução rápido o suficiente para serem utilizadas ao mesmo tempo em que o teste está sendo desenvolvido.

Ao conduzir os participantes de uma pesquisa a realizar testes de mutação, o estudo E4 constatou que os participantes foram capazes de aumentar a cobertura de código entre 2% a 9%, além do desenvolvedor ser obrigado a inspecionar manualmente o código fonte para encontrar a localização do mutante ao analisar os resultados. Em relação aos operadores de mutação, a cobertura de código irá aumentar independente da escolha, no entanto, existe, provavelmente, uma quantidade mínima de operadores que serão necessários para se obter um resultado satisfatório. Portanto, ao aplicar testes de mutação, a velocidade e eficiência dos operadores devem ser priorizadas ao invés da quantidade ou tipo, a recomendação é que a técnica seja utilizada ao escrever testes unitários, pois antecipar os possíveis defeitos poderá ser útil para garantir a exatidão do que está sendo desenvolvido.

O estudo E5 compara a técnica de mutação com outros três critérios (*Edge-Pair, All-uses and Prime Path Coverage)* e conclui que utilizar testes de mutação é a forma mais efetiva para se detectar defeitos e, apesar de sua reputação exigir um alto custo, a análise de mutação foi a técnica que exigiu o menor número de testes para satisfazer os critérios.

O estudo E6 também constata o aumento da cobertura de código a partir do aumento do escore de mutação. Foi observado que o conjunto de operadores utilizado irá determinar quantos novos testes deverão ser criados, assim como em qual área do código terá mais atenção, além de não parecer haver um resultado conclusivo sobre quais operadores produzem consistentemente a maioria dos casos de teste.

5. Discussão

Os estudos E2, E4 e E6 possuem resultados similares ao investigar a correlação entre o escore de mutação e a cobertura de código. Foi constatado que quaisquer variações no valor do escore de mutação ocasionarão uma mudança no valor da cobertura de código, além de as duas medidas possuírem uma correlação forte.

Em relação aos operadores de mutação, os estudos E1, E3, E4 e E6 constataram que o resultado obtido ao utilizar testes de mutação é altamente dependente dos

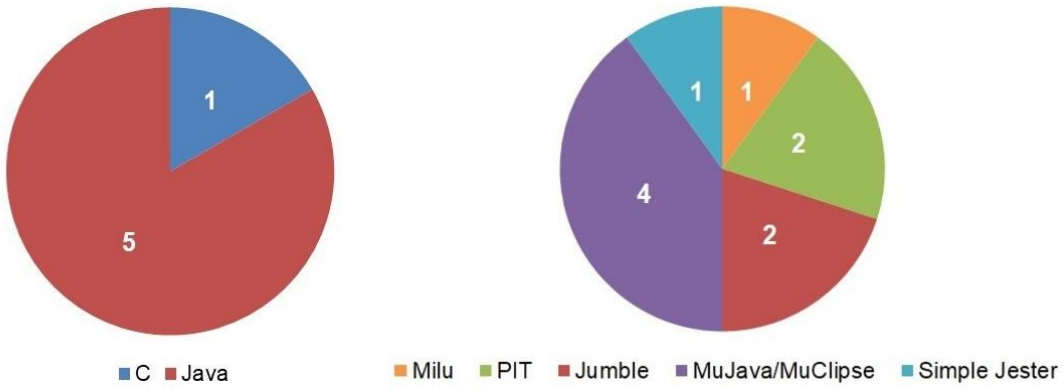
operadores utilizados, sendo que é possível observar as seguintes singularidades: (i) ao utilizar um pequeno conjunto de operadores, o escore de mutação será similar a cobertura de código, (ii) os operadores utilizados influenciam na quantidade de mutantes duplicados e (iii) o conjunto de operadores utilizados irá determinar o número de mutantes gerados. Como não foi observado um resultado conclusivo a respeito de quais operadores irão otimizar o tempo de execução e número de mutantes criados, a recomendação é que sejam priorizadas a velocidade e eficiência dos operadores ao invés da quantidade ou tipo.

Ao analisar os estudos E1, E3, E4, E5, o estudo E1 constata que o custo de se utilizar testes de mutação em um projeto de larga escala é alto e excede os recursos computacionais disponíveis, além de gerar uma enorme quantidade de mutantes. No entanto, o estudo E3 contraria esses resultados, concluindo que é possível inserir tal prática no ciclo de desenvolvimento de um software, pois as ferramentas apresentaram um tempo de execução satisfatório, sendo que o estudo E1 analisa um software implementado em C com cerca de 60.000 linhas de código, diferente do estudo E3 que analisou um projeto feito em Java sem demais especificações, sugerindo que para grandes aplicações seria necessário ter uma maior compreensão dos operadores de mutação utilizados e técnicas para analisar os mutantes a fim de tornar seu uso viável. Comparando com outras técnicas de inserção de falhas, realizar teste de mutação foi a forma mais efetiva para se detectar defeitos e exigiu o menor número de testes para satisfazer os critérios, além de obrigar o desenvolvedor a inspecionar o código fonte manualmente para encontrar a localização do mutante ao analisar os resultados. A recomendação é que testes de mutação sejam utilizados ao escrever testes unitários, pois antecipar os possíveis defeitos poderá ser útil para garantir a exatidão do que está sendo desenvolvido.

5.1. Linguagem e ferramenta utilizadas

De acordo com a figura 4, apenas o estudo E1 utilizou um software feito em C como experimento, todos os outros estudos analisaram experimentos feitos em Java. É possível perceber a variedade de ferramentas disponíveis atualmente para se aplicar testes de mutação, conforme mencionado no estudo E3.

Figura 4 - Linguagens de programação e ferramentas de mutação utilizadas nos estudos



6. Conclusões

A busca inicial retornou 19 estudos que, após serem avaliados de acordo com os critérios de inclusão e exclusão estabelecidos no protocolo de seleção, foram selecionados 6 estudos primários. A extração de dados ocorreu de acordo com a tabela 3 para responder à seguinte questão de pesquisa: como é possível melhorar a qualidade de testes unitários utilizando testes de mutação?

A partir dos resultados obtidos pode-se concluir que introduzir testes de mutação no ciclo de desenvolvimento de um software é uma boa forma de garantir a exatidão do que está sendo desenvolvido, de avaliar tanto a qualidade dos testes unitários quanto a sua eficácia em detectar defeitos e também de aumentar a cobertura de código, pois um alto escore de mutação implica em uma alta cobertura de código devido a uma correlação forte entre as duas medidas. Além disso, atualmente há muitas ferramentas, tanto comerciais quanto *opensource*, em diferentes linguagens de programação para se realizar testes de mutação, podendo-se obter resultados satisfatórios com um tempo de execução rápido o suficiente para ser executado no mesmo momento em que os testes estão sendo escritos.

Como trabalho futuro, devido ao resultado inconclusivo sobre a influência dos operadores de mutação, há a opção de se conduzir uma investigação para analisar quais operadores e técnicas para analisar os mutantes devem ser utilizados para se obter um resultado otimizado, tanto em relação ao número de mutantes quanto ao tempo de execução.

7. Referências

ASSYLBEKOV, Berik et al. Investigating the correlation between mutation score and coverage score. In: **Computer Modelling and Simulation (UKSim), 2013 UKSim 15th International Conference on**. IEEE, 2013. p. 347-352.

CRAIG, Rick David; JASKIEL, Stefan P. **Systematic software testing**. Artech House, 2002.

DE ALMEIDA BIOLCHINI, Jorge Calmon et al. Scientific research ontology to support systematic review in software engineering. **Advanced Engineering Informatics**, v. 21, n. 2, p. 133-151, 2007.

HUNT, Andy; THOMAS, Dave. **Pragmatic unit testing in Java with JUnit**. The Pragmatic Bookshelf, 2003.

JIA, Yue; HARMAN, Mark. An analysis and survey of the development of mutation testing. **IEEE transactions on software engineering**, v. 37, n. 5, p. 649-678, 2011.

KITCHENHAM, Barbara. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1-26, 2004.

NIEDERMAYR, Rainer; JUERGENS, Elmar; WAGNER, Stefan. Will my tests tell me if I break this code?. In: **Proceedings of the International Workshop on Continuous Software Evolution and Delivery**. ACM, 2016. p. 23-29.

RAMLER, Rudolf; KASPAR, Thomas. Applicability and benefits of mutation analysis as an aid for unit testing. In: **Computing and Convergence Technology (ICCCT), 2012 7th International Conference on**. IEEE, 2012. p. 920-925.