DiffMutAnalyze: Uma abordagem para auxiliar a identificação de mutantes equivalentes

Juliana Botelho[1], Carlos Henrique Pereira[1], Vinicius H. S. Durelli[2], Rafael S. Durelli [1]

[1]Universidade Federal de Lavras (UFLA)
Lavras – MG – Brasil

[2]Universidade Federal de São João del-Rei (UFSJ)
São João del-Rei – MG – Brasil

juliana.botelho@posgrad.ufla.br, carloshpereira27@gmail.com
durelli@ufsj.edu.br, rafael.durelli@dcc.ufla.br

Abstract. *The mutation test is considered an effective technique for fault localization, but it has the disadvantage of the high cost of its application. Thus, it determines one of the obstacles to its adoption, since it is necessary to manually identify which mutants behave (i.e., produce the same output) as the original code for all possible inputs. These codes are considered equivalent and need to be analyzed manually to confirm this equivalence. In order to assist test analysts in the identification of these mutants, this paper present an approach that supports the generation of mutants, inspects the codes and informs the analysis data. Thus, it is possible for the analyst to perform the mutation test and to analyze the mutants that remained alive in a single tool.*

Resumo. *O teste de mutação é considerado uma técnica eficaz para localização de falhas, porém tem como desvantagem o alto custo de sua aplicação. Com isso, determina um dos obstáculos à sua adoção, pois é necessário identificar manualmente quais mutantes se comportam (i.e., produzem a mesma saída) como o código original para todas as entradas possíveis. Esses códigos são considerados equivalentes e precisam ser analisados manualmente para confirmação dessa equivalência. A fim de auxiliar os analistas de teste na identificação desses mutantes, o artigo propõe uma abordagem que apoia a geração dos mutantes, inspeção dos códigos e informa os dados da análise. Assim, é possível o analista de teste realizar o teste de mutação e analisar os mutantes que permaneceram vivos em uma única ferramenta.*

1. Introdução

No âmbito da engenharia de software, um dos propósitos é garantir a qualidade dos sistemas de software desenvolvidos. Durante o processo de implementação, o sistema de software deve ser avaliado garantindo que o código implementado esteja em conformidade com o que foi especificado. Uma maneira de avaliar o código é utilizando testes de unidade, porém esses testes também precisam atingir um bom nível de qualidade. Assim, para garantir a qualidade dos casos de teste, o critério de teste denominado teste de mutação tem a finalidade de adequar os casos de testes para alcançar uma boa cobertura do sistema de software [Delamaro et al. 2016].

Há uma medida para avaliar a qualidade da atividade de teste de mutação, denominada *"escore de mutação"* (*mutation score*). O escore de mutação é dado pela razão

do número de mutantes mortos pelo total de mutantes gerados subtraído pelo número de mutantes equivalentes [DeMillo 1980]. O escore de mutação varia entre 0 e 1 (100%). Como o cálculo do escore de mutação depende da quantidade de mutantes equivalentes, é importante realizar a análise dos mesmos. Uma vez que os mutantes equivalentes atuam como falsos positivos, pois nenhum teste pode detectá-los [Grün et al. 2009]. Isso significa que a eliminação desses mutantes não afeta na geração dos casos de teste, mas no cálculo do escore de mutação. Assim, a precisão da métrica se torna questionável.

A análise de mutantes equivalentes é uma etapa fundamental para a condução do teste de mutação, pois somente por meio da identificação desses mutantes é que os testadores podem calcular devidamente o escore de mutação. Em outras palavras, tal passo é essencial para possibilitar que os testadores avaliem a qualidade do conjunto de casos de teste existente. Como a distribuição de mutantes tende a ser imprevisível e a identificação da equivalência dos mutantes é um problema indecidível [Budd and Angluin 1982], é difícil julgar a qualidade do conjunto de testes com base no escore de mutação sem que os mutantes equivalentes sejam identificados.

Segundo Grün et al., leva-se aproximadamente 15 minutos para verificar manualmente se um mutante é equivalente ao sistema de software original [Grün et al. 2009]. Considerando esse tempo e acrescentando a quantidade de mutantes gerados pelos operadores de mutação, que geralmente são gerados em grande quantidade, é necessário averiguar maneiras de minimizar o custo dessa análise, uma vez que são gastas horas de um analista de teste para realizar a inspeção dos códigos, o que contribui para o custo humano da análise.

Dentro desse contexto e para auxiliar a inspeção manual da equivalência entre os códigos (i.e., original e mutantes), a abordagem proposta visa auxiliar os analistas de teste na identificação desses mutantes, objetivando minimizar o tempo gasto com a análise desses mutantes. Assim, a ferramenta DiffMutAnalyze foi desenvolvida com o intuito de gerar os mutantes no projeto sendo testado e exibir as duas versões do código (i.e., original e mutante) lado a lado, para verificação guiada e manual da equivalência. Essa visualização do código facilita sua inspeção, permitindo ao analista de teste comparar diretamente o código original e seu mutante, além de evidenciar a alteração realizada.

Durante a realização da análise da equivalência, DiffMutAnalyze contabiliza o tempo gasto com a inspeção dos códigos. Dessa forma, é contabilizado o tempo efetivamente gasto com a análise. Além disso, a ferramenta permite que o analista de teste indique o grau de dificuldade em analisar o dado mutante. Assim, é possível identificar os mutantes mais difíceis de serem analisados e, consequentemente, verificar qual operador de mutação que o gerou, permitindo ao analista conhecer os operadores de mutação que geram a maior quantidade de mutantes equivalentes e quais possuem maior dificuldade em sua análise.

Esse artigo apresenta as seguintes contribuições: *(i)* auxiliar a identificação de mutantes equivalentes; *(ii)* centralizar a geração dos mutantes e a análise dos equivalentes em um só ambiente; e *(iii)* fornecer relatórios sobre: o tempo total gasto com a análise, o tempo gasto em cada mutante e o grau de dificuldade em analisar tais mutantes.

O restante do trabalho está organizado da seguinte maneira: Seção 2 introduz informações sobre teste de mutação e mutantes equivalentes. A ferramenta proposta é

descrita na Seção 3. Os trabalhos relacionados estão referenciados na Seção 4. Por fim, na Seção 5, estão as observações finais e trabalhos futuros.

2. Background

Esta seção fornece informações sobre o teste de mutação e elucida o problema dos mutantes equivalentes. Além disso, fornece informações relacionadas aos operadores de mutação empregados na ferramenta Major.

2.1. Teste de Mutação

O teste de mutação, proposto por DeMillo et al. [DeMillo et al. 1978], é uma proposta eficaz para realizar testes em sistemas de softwares em desenvolvimento. Esse critério de teste introduz mudanças sintáticas no código original e requer que essas alterações sejam descobertas pelos casos de teste construídos. Caso o conjunto de testes definidos inicialmente não identifiquem essas mudanças, novos testes são necessários, ou ainda, os mutantes gerados podem ser equivalentes ao software original. Dessa forma, o teste de mutação contribui com dois fatores [Kintis and Malevris 2014]: *(i)* melhoria da qualidade dos casos de teste; e *(ii)* descoberta de falhas durante o desenvolvimento.

Para avaliar a qualidade de um dado conjunto de testes, os casos de testes existentes são executados nos mutantes gerados, a fim de determinar se a alteração introduzida pode ser detectada, ou seja, se o mutante pode ser morto. Diz-se que um mutante é morto se sua saída for diferente da saída do sistema de software original, quando ambos são executados com o mesmo caso de teste; caso contrário, o mutante permanece vivo. Dessa forma, o ideal é que todos os mutantes sejam mortos pelos casos de teste. Porém, nem todos os mutantes podem ser mortos, esses mutantes são funcionalmente equivalentes ao programa original e, consequentemente, nenhum caso de teste é capaz de detectá-los [Kintins and Malevris 2013]. Estes mutantes são denominados mutantes equivalentes e, como descrito a seguir, eles constituem um custo no teste de mutação.

2.2. Mutantes Equivalentes

A realização de testes de mutação implica em grande quantidade de esforços humanos, contribuindo para o aumento do custo. Um esforço humano bastante significativo está relacionado com a identificação dos mutantes equivalentes. Um mutante é equivalente quando ele possui a mesma semântica do sistema de software original, porém os códigos são sintaticamente diferentes, o que leva a uma possível mudança não observável no comportamento [Jia and Harman 2011]. Uma vez que tem como resultado a mesma saída que o software original. Dessa forma, não existe um caso de teste capaz de distinguir esse mutante do código original [Papadakis et al. 2014].

Com a geração desse tipo de mutante, ele se tornou um dos principais obstáculos da adoção do teste de mutação. O problema do mutante equivalente foi mostrado como sendo indecidível em sua forma geral [Budd and Angluin 1982]. Isso implica que uma solução totalmente automatizada não pode ser atingida [Kintis and Malevris 2014]. Portanto, esses mutantes precisam ser analisados manualmente, para verificar se há, de fato, a equivalência entres os códigos original e mutante. Assim, esses mutantes constituem um custo importante no teste de mutação.

2.3. Ferramenta de Mutação Major

O teste de mutação faz alterações no programa a ser testado e essas alterações são baseadas em um conjunto de regras. Essas regras são implementadas nos operadores de

mutação, que por sua vez são projetados para realizar a transformação do código e gerar os mutantes [Ammann and Offutt 2008]. Geralmente esses operadores estão empregados em ferramentas de mutação e, para a abordagem proposta neste artigo, a ferramenta Major[1] foi definida.

Major é uma ferramenta de mutação integrada ao compilador Java e não requer uma estrutura de análise de mutação específica. Por isso, ela pode ser usada em qualquer ambiente baseado em Java. Major manipula a árvore sintática abstrata (*AST - Abstract Syntax Tree*) analisando o código-fonte do programa em teste. Como a Major realiza as alterações utilizando a AST, que representa o código desenvolvido, evita gerar mutantes que não podem ser mapeados para uma localização específica no código original [Just 2014].

3. DiffMutAnalyze

A ferramenta de apoio computacional proposta, denominada DiffMutAnalyze [2], objetiva auxiliar os analistas de teste na identificação dos mutantes equivalentes. A principal motivação para o desenvolvimento dessa ferramenta é tentar minimizar o alto custo da análise desses mutantes. Além de fornecer evidências do grau de dificuldade em analisar tais mutantes e o tempo efetivamente gasto com a análise. A ferramenta possui o propósito de localizar a alteração realizada e apresentar ao analista de teste a comparação, lado a lado, da diferença entre o código original e o mutante. Desse modo, os objetivos da ferramenta incluem: *(i)* acessar o projeto diretamente pelo GitHub ou importando uma pasta *.zip* do projeto; *(ii)* gerar os mutantes através da ferramenta Major; *(iii)* disponibilizar os mutantes vivos para análise da equivalência; *(iv)* associar o grau de dificuldade na análise com o mutante gerado; *(v)* contabilizar o tempo da análise; e *(vi)* disponibilizar relatórios com os dados da análise.

3.1. Proposta da Abordagem

DiffMutAnalyze é uma ferramenta que gerencia e controla os mutantes analisados. Um gráfico (ou quadro) é gerado com todos os mutantes que foram destinados para análise. Assim, o analista de teste consegue gerenciar e controlar a análise desses mutantes. Após a inspeção dos códigos – original e mutante – o analista de teste determina a equivalência de ambos e avalia a análise de acordo com o grau de dificuldade. Finalizada a avaliação, o analista de teste procede com a análise do próximo mutante.

Caso haja dúvida da equivalência de um mutante com seu código original correspondente, DiffMutAnalyze permite que o analista de teste prossiga para a próxima análise e volte posteriormente para esse mutante. Dessa forma, o gráfico dos mutantes possibilita ao analista de teste identificar os mutantes que foram analisados e determinada sua equivalência, assim como, identificar os mutantes que foram analisados, porém sua equivalência não foi determinada. Além disso, os mutantes que não foram analisados são marcados, para que o analista de teste obtenha a visualização desses mutantes e possa escolher qual será o próximo a ser inspecionado.

O tempo gasto pelo analista de teste em analisar os mutantes é contabilizado em cada análise e, ao final, é possível obter o tempo total gasto. Para que o tempo seja contabilizado de maneira real, um *timer* fica disponível na tela para que o analista possa pausar a contagem do tempo, caso seja necessário. Um relatório é exibido ao analista

[1]Major. Disponível em: http://mutation-testing.org/

[2]Disponível em: https://github.com/PqES/DiffMutAnalyze

de teste contendo todos os dados da análise, como quantidade de mutantes equivalentes, avaliação do grau de dificuldade e tempo gasto com a análise.

3.2. Visão geral da DiffMutAnalyze

Conforme ilustrado na Figura 1, a ferramenta DiffMutAnalyze permite que o analista de teste inclua projetos diretamente pelo GitHub, inserindo a *URL* do projeto, ou submeta uma pasta (*.zip*) do projeto que será analisado (Figura 1 (a)), os projetos inseridos precisam conter casos de teste criados previamente. Após inserir o projeto, a ferramenta gera os mutantes utilizando os operadores da ferramenta de mutação Major (Figura 1 (b)). Os mutantes gerados são armazenados em um diretório do projeto, contendo todas as classes Java alteradas. Com os mutantes gerados, os casos de teste são executados nos mutantes e são verificados quais mutantes foram mortos e quais sobreviveram (Figura 1 (c, d)).

Os mutantes que não foram detectados pelos casos de teste (ou permaneceram vivos) são agrupados e destinados para análise entre eles e o código original (Figura 1 (e)). DiffMutAnalyze exibe para o analista de teste os códigos original e mutante que permaneceu vivo lado a lado para serem analisados manualmente (Figura 1 (f)). A cada mutante analisado, o analista de teste indica se há ou não a equivalência entre os códigos e a ferramenta contabiliza o tempo gasto da análise do mutante em questão (Figura 1 (g)). Além disso, o analista de teste deve indicar o grau de dificuldade da análise (Figura 1 (h)). Assim, a análise é realizada para todos os mutantes que sobreviveram e relatórios (Figura 1 (i)) são gerados com os dados da análise.

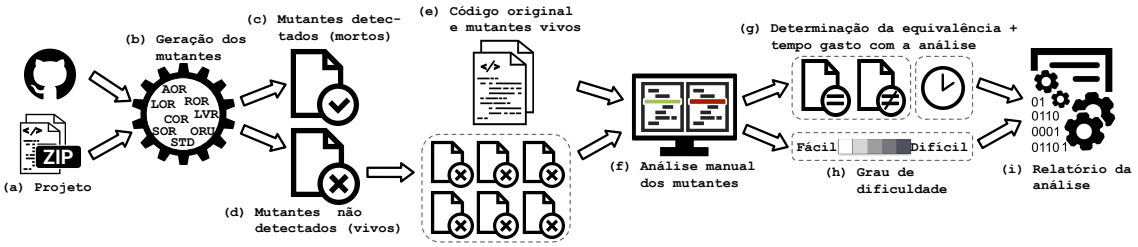

Figura 1. Passo a passo da ferramenta DiffMutAnalyze.

A ferramenta DiffMutAnalyze foi desenvolvida em Java, utilizando tecnologias como o *framework* Spring Boot[3], JavaScript e HTML e os dados são armazenados no banco de dados MySQL. DiffMutAnalyze possui um *design web*, porém seu código será disponibilizado para que seja utilizada no servidor local. O principal motivo de utilizar a ferramenta localmente, é devido à segurança do código do projeto. Uma vez que o mesmo será submetido à ferramenta e seu código inserido no banco de dados, para que todo o processo da mutação seja realizado.

Como pode ser observado na Figura 2, a ferramenta DiffMutAnalyze disponibiliza ao analista de teste a visualização dos códigos lado a lado e permite que o usuário informe se os códigos são equivalentes e o grau de dificuldade em analisá-los. Mais detalhes sobre os procedimentos para realização da análise estão descrito na seção a seguir.

3.3. Procedimentos para realização da análise

1) Seleção do projeto a ser analisado: Para facilitar a geração e análise dos mutantes sobreviventes, DiffMutAnalyze permite que o analista de teste insira o projeto e, no mesmo

[3]https://projects.spring.io/spring-boot/

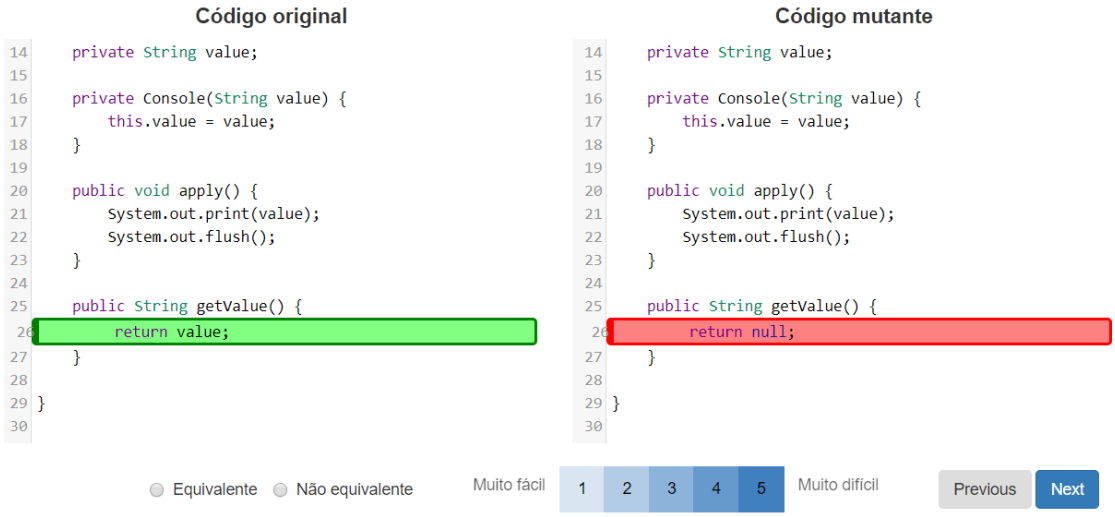

Figura 2. Analisador DiffMutAnalyze.

ambiente, a ferramenta gera os mutantes, executa os casos de teste e mantém os mutantes que serão analisados. Uma vez que os mutantes vivos são selecionados, DiffMutAnalyze busca pelo primeiro mutante e o apresenta para análise.

2) Projeto inserido: DiffMutAnalyze gera uma árvore contendo a estrutura do projeto inserido, essa funcionalidade permite ao analista de teste navegar pelo projeto e escolher a classe que deseja analisar.

3) Quadro dos mutantes: Os mutantes que sobreviveram são inseridos em um quadro. Assim, é possível identificar quais mutantes serão analisados em cada classe do sistema de software. Cada mutante é representado por um retângulo e possui três cores distintas: *(i)* azul; *(ii)* verde; e *(iii)* vermelha, no qual cada cor representa um *status* do mutante. Os mutantes representados pela cor azul são os mutantes que não foram analisados. A cor verde referencia mutantes que foram analisados e avaliados, determinada sua equivalência e o grau de dificuldade da análise. A cor vermelha representa os mutantes que foram inspecionados, porém, sua equivalência não foi definida. O quadro contendo todos os mutantes e seus status possibilita ao analista de teste obter uma visão geral da situação da análise, além de permitir selecionar um mutante para análise clicando sobre o retângulo correspondente.

4) Buscar mutante não analisado: Durante a análise dos mutantes podem surgir dúvidas quanto a sua real equivalência. Dessa forma, o analista de teste pode seguir adiante e inspecionar outro mutante, retornando à esse mutante posteriormente. Assim, a ferramenta busca pelo primeiro mutante não analisado ou "pulado" por meio do botão *"Buscar mutante não analisado"*, colocando-o em evidência no campo de análise dos códigos.

5) *Diff* entre os códigos original e mutante: DiffMutAnalyze exibe os códigos original e mutante lado a lado e coloca em evidência a mutação realizada. Isso é um benefício para tornar a análise mais rápida. Suponha uma classe com muitas linhas de código, assim, o analista de teste não necessita procurar onde foi efetuada a mutação.

6) Determinação da equivalência entre os códigos: Após a inspeção dos códigos, o analista de teste indica se há equivalência ou não do mutante com o código original.

7) Avaliação do grau de dificuldade: Além de determinar se o mutante é equivalente a seu original correspondente, o analista de teste indica o grau de dificuldade da análise. Essa funcionalidade foi determinada para auxiliar a identificar, ao final da análise, os mutantes mais difíceis de serem analisados e, consequentemente, determinar o operador de mutação que o gerou, considerando assim a complexidade da análise.

8) Geração do relatório da análise: O relatório da análise é emitido através do botão *"Gerar relatório"*. Os dados são exibidos em três estágios: *(i)* resultados da análise total dos mutantes; *(ii)* resultados da análise por classe do sistema de software sendo testado; e *(iii)* resultados da análise por mutante. Além disso, o cálculo do escore mutação é informado. O primeiro estágio informa: o tempo total gasto com a análise de todos os mutantes, o tempo médio por mutante, o grau de dificuldade médio da análise, a quantidade total de mutantes que foram determinados como equivalente e a quantidade de mutantes não equivalentes. O segundo estágio informa os resultados por meio das mesmas variáveis do estágio anterior, a diferença é que os resultados são exibidos por cada classe do projeto. Por fim, o terceiro estágio do relatório, exibe os resultados por mutante, informando o tempo gasto com a análise de determinado mutante, o grau de dificuldade atribuído a ele e a equivalência definida.

4. Trabalho relacionado

Uma abordagem de gamificação para identificar os mutantes equivalentes de maneira interativa foi proposta por Houshmand e Paydar, denominada EMVille [Houshmand and Paydar 2017]. Essa abordagem propõe uma estrutura que procura reduzir as limitações das técnicas automatizadas existentes. EMVille é na verdade um sistema baseado na *web* onde os analistas de teste analisam os mutantes sobreviventes através de um jogo.

Os resultados do experimento mostram que o envolvimento dos participantes é visivelmente aumentado pela abordagem de gamificação. Além disso, EMVille possui o *Diff Visualizer*, um visualizador que mostra as diferenças do código original e do mutante de uma forma que o jogador pode compará-los, assim, os participantes do experimento informaram que os componentes que contemplam essa abordagem auxiliam na análise das instâncias dos mutantes. Dessa forma, é possível perceber que a existência de uma ferramenta para auxiliar a análise dos mutantes sobreviventes é útil durante o processo do teste de mutação. Em relação à ferramenta proposta neste artigo, além de possuir a visualização dos códigos lado a lado, há as informações do grau de dificuldade em analisar os mutantes e a contabilização real do tempo gasto com a análise.

5. Conclusão

No contexto do teste de mutação, os mutantes formam os objetivos do processo de teste. Assim, casos de teste que são capazes de distinguir os comportamentos dos mutantes e do código original são indicados a identificar possíveis falhas no sistema de software [Papadakis et al. 2017]. Na prática, alguns desses mutantes são equivalentes, ou seja, eles formam versões funcionalmente equivalentes ao sistema de software original, dessa forma, eles não contribuem para melhorar o conjunto de casos de teste e precisam ser analisados para verificar se são realmente equivalentes ao código original ou se mais casos de teste são necessários para identificar os mutantes sobreviventes.

Com base nesse contexto, este artigo propôs uma abordagem que auxilia na identificação dos mutantes equivalentes. Como resultados, espera-se que a ferramenta Diff-

MutAnalyze contribua de forma eficiente com a realização da análise dos mutantes, sendo possível inspecionar os mutantes sobreviventes comparando-os com sua versão original e obter informações para verificar o custo real da análise dos mutantes. Uma vez que a ferramenta permite que o analista de teste determine a equivalência dos mutantes analisados e indique o grau de dificuldade de cada análise, além de contabilizar o tempo gasto total e de cada análise dos mutantes.

Em suma, as informações extraídas podem auxiliar a outros pesquisadores a utilizar os operadores de mutação pontualmente. Uma vez que serão analisados quais os operadores que mais contribuem com o custo, gerando maior quantidade de mutantes equivalentes e quais os operadores mais complexos de serem analisados, consequentemente, o analista de teste pode decidir quais operadores serão utilizados para outros sistemas de software. Além disso, a ferramenta será útil para estudos empíricos em projetos de pesquisa. Como trabalho futuro, pretende-se realizar um experimento para avaliar a funcionalidade da ferramenta e obter informações reais do custo em analisar os mutantes sobreviventes.

AGRADECIMENTOS

Este projeto é apoiado pela FAPEMIG.

Referências

Ammann, P. and Offutt, J. (2008). *Introduction to Software Testing*. Cambridge University Press.

Budd, T. A. and Angluin, D. (1982). Two notions of correctness and their relation to testing. *Acta Informatica*, 18(1):31–45.

Delamaro, M., Maldonado, J., and Jino, M. (2016). *Introdução ao teste de software*. Elsevier, 2nd edition.

DeMillo, R. A. (1980). Mutation analysis as a tool for software quality assurance. Technical report, Georgia Int of Tech Atlanta School of Information and Computer Science.

DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.

Grün, B. J., Schuler, D., and Zeller, A. (2009). The impact of equivalent mutants. In *2nd International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 192–199. IEEE.

Houshmand, M. and Paydar, S. (2017). Emville: A gamification-based approach to address the equivalent mutant problem. In *7th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 326–332.

Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE transactions on Software Engineering*, 37(5):649–678.

Just, R. (2014). The major mutation framework: Efficient and scalable mutation analysis for java. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 433–436. ACM.

Kintins, M. and Malevris, N. (2013). Identifying more equivalent mutants via code similarity. In *20th Asia-Pacific Software Engineering Conference (APSEC)*, pages 180–188.

Kintis, M. and Malevris, N. (2014). Using data flow patterns for equivalent mutant detection. In *7th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 196–205.

Papadakis, M., Delamaro, M., and Le Traon, Y. (2014). Mitigating the effects of equivalent mutants with mutant classification strategies. *Science of Computer Programming*, 95:298–319.

Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Le Traon, Y., and Harman, M. (2017). Mutation testing advances: An analysis and survey. *Advances in Computers*.