DiffMutAnalyze: uma ferramenta para auxiliar a análise dos mutantes equivalentes no ensino do teste de mutação

Juliana Botelho[1], Maurício Souza[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
julianabotelhoc@gmail.com, mauricio.ronny@ufla.br
durelli@ufsj.edu.br, rafael.durelli@ufla.br

Abstract. *Mutation testing is a technique that aims to evaluate a test cases and, consequently, to help improve their quality. Mutation testing is a promising technique to assist students in developing skills in software testing. Thus, this article presents the use of the DiffMutAnalyze tool in the academic field. It provides an environment that: (i) promotes the generation of mutants, (ii) the execution of test cases in the mutants generated, (iii) the identification of the detected mutants, and (iv) the availability of the analysis of possible equivalent mutants. DiffMutAnalyze was evaluated with 11 Graduate Students in Computer Science, to support the teaching of mutant testing and to show the practical application of mutation and analysis of mutants. A study was performed with the students to verify the cost of the equivalent mutant analysis, being compared in two ways: (i) by using DiffMutAnalyze; and (ii) through manual analysis, without using DiffMutAnalyze. The results showed a significant time reduction in comparison to the manual analysis.*

Resumo. *O teste de mutação é uma técnica que visa avaliar os casos de teste e, consequentemente, ajudar a melhorar sua qualidade. O teste de mutação é uma técnica promissora para ajudar os alunos a desenvolverem habilidades em testes de software. Assim, este artigo apresenta a utilização da ferramenta DiffMutAnalyze no âmbito acadêmico. Ela fornece um ambiente que: (i) promove a geração de mutantes, (ii) promove a execução de casos de teste nos mutantes gerados, (iii) permite a identificação dos mutantes detectados e (iv) disponibiliza para análise dos possíveis mutantes equivalentes. A DiffMutAnalyze foi avaliada com 11 estudantes de Pós-Graduação em Ciência da Computação, para apoiar o ensino de testes de mutantes e para mostrar a aplicação prática de mutação e análise de mutantes. Foi realizado um estudo, junto aos estudantes, para verificar o custo da análise dos mutantes equivalentes, sendo comparado de duas maneiras: (i) pelo uso da DiffMutAnalyze; e (ii) por meio da análise manual, sem o uso da DiffMutAnalyze. Os resultados mostraram uma redução significativa de tempo em comparação com a análise manual.*

1. Introdução

No mundo real, a verificação da existência de *bugs* em bases de código grandes e complexas não é uma tarefa trivial, é necessário que os analistas possuam conhecimentos de

diferentes práticas para inspecionar esses softwares. Tais práticas variam de simples testes exploratórios manuais, buscando encontrar *bugs* manualmente, até técnicas avançadas de teste, como testes automatizados [Aniche et al. 2019]. Vinton G. Cerf discutiu sobre programação responsável na indústria [Cerf 2014], esse termo enfatiza o senso de responsabilidade dos programadores no que diz respeito à operação confiável e resistência ao comprometimento e erro. Vinton G. Cerf acredita que são necessárias melhores ferramentas interativas que possam verificar o comportamento do sistema sendo desenvolvido.

Incentivar os alunos sobre a importância dos testes é um tópico que deve ser trabalhado em sala de aula, é preciso encontrar maneiras de incentivar a prática da utilização de técnicas de testes em projetos de programação. Somente exigir que os alunos utilizem essas técnicas durante as aulas pode não ser muito efetivo, o intuito é fazer com que os alunos entendam a necessidade dos testes; é fazer com que eles apreciem o valor que um caso de teste possui [Zheng et al. 2018, Spacco and Pugh 2006] e incentivar a produção de seus próprios casos de teste, almejando desenvolver softwares com qualidade.

Para garantir a qualidade dos casos de teste, há o critério de teste chamado teste de mutação, que possui a finalidade de adequar os casos de teste para alcançar uma boa cobertura do sistema de software [Delamaro et al. 2016]. A ideia do teste de mutação é, dado um sistema de software, modificações são aplicadas e versões diferentes do código são geradas, chamadas de mutantes, como se falhas estivessem sendo "semeadas" no código original. A técnica faz com que o desenvolvedor construa casos de teste que explorem comportamentos diferentes do comumente explorado [Papadakis et al. 2019]. Em geral, enfatiza a criação de casos de teste mais eficazes por meio do refinamento do conjunto de teste inicial.

Ao verificar a importância de desenvolver novas habilidades e gerir novos conhecimentos para os alunos de engenharia de software, o critério de teste de mutação é promissor para ser empregado nessas disciplinas. Uma ferramenta de apoio computacional ao ensino do teste de mutação foi desenvolvida e sua abordagem foi apresentada em um trabalho anterior [Botelho et al. 2018]. A DiffMutAnalyze possui o intuito de disponibilizar um ambiente amplo de aprendizagem sobre teste de mutação, contemplando desde a geração dos mutantes até os resultados das análises desses mutantes. Assim, o aluno conta com um ambiente que integra a ferramenta Major [Just 2014] para geração da mutação, não sendo necessária a realização da mutação fora do contexto da ferramenta.

Um dos fatores que motivou a criação da DiffMutAnalyze para a análise dos mutantes equivalentes, está relacionado com o escore de mutação. Para que essa medida seja calculada de maneira exata, é necessário identificar a quantidade de mutantes equivalentes gerados. Mesmo que os mutantes equivalentes não ajudem a melhorar os casos de teste, pois nenhum teste pode detectá-los [Grün et al. 2009], é necessário identificá-los para não afetar o cálculo do escore de mutação e, assim, manter a precisão da métrica.

No trabalho anterior [Botelho et al. 2018], a DiffMutAnalyze foi apresentada, porém não haviam avaliações sobre seu uso por estudantes. Dessa forma, este trabalho possui o objetivo de avaliar o uso da ferramenta para análise dos mutantes, essa avaliação foi realizada comprando a análise dos mutantes de duas maneiras: utilizando a ferramenta e sem o uso da mesma, identificando assim, o custo da análise manual. As principais contribuições deste trabalho são: (*i*) Avaliação da ferramenta DiffMutAnalyze em ambiente

acadêmico; (*ii*) Proposta do uso de teste de mutação para apoiar o aprendizado na criação de testes de unidade; e (*iii*) Investigação do custo da análise manual dos mutantes equivalentes. O restante deste artigo está organizado da seguinte forma: Seção 2 apresenta os trabalhos relacionados. Na Seção 3, está descrita uma investigação do custo em analisar os mutantes equivalentes manualmente. As ameças à validade são apresentadas na Seção 4. Por fim, na Seção 5, estão as observações finais.

2. Trabalhos Relacionados

XSOFT [Radhakrishnan et al. 2015]: Foi desenvolvido um modelo de ensino e aprendizado de software genérico chamado *eXtreme Software Teaching* (XSOFT), integrando os principais elementos do método *Extreme Programming*. O objetivo da ferramenta é fazer com que os alunos realizem tarefas de projetos em tempo real. Testes e integração são realizados após cada tarefa concluída. O esforço do aluno em realizar a tarefa é medido por meio de uma estimativa quantitativa, ou seja, a quantidade de esforço despendido pelo aluno para realizar cada tarefa. As tarefas oferecidas aos alunos são de projeto em tempo real com o intuito de fazer com que o desenvolvimento e a correção estejam no mesmo nível de uma indústria de software.

STCAES [Zheng et al. 2018]: *Software Testing Computer Assistant Education System* (STCAES) é uma ferramenta para testes em sala de aula e aprendizagem em testes de software. Por meio da utilização dessa ferramenta, os alunos acreditam que a utilização de uma ferramenta para aprendizagem do teste de software se torna mais útil e interessante na vida real, sendo mais propenso a estimular o aprendizado. Os autores acreditam que o conteúdo da sala de aula construído pelo STCAES é mais atraente do que o conteúdo tradicional e mais eficiente em termos de tempo, o que pode aumentar muito o entusiasmo dos alunos pelo aprendizado.

3. Identificação do custo em analisar os mutantes manualmente

Conforme mencionado, no trabalho anterior [Botelho et al. 2018], foi apresentada a ferramenta DiffMutAnalyze. Uma de suas propostas é auxiliar os analistas durante a identificação dos mutantes equivalentes. Assim, diante do contexto elucidado, a utilização dessa ferramenta auxilia no aprendizado dos alunos com as seguintes contribuições: (*i*) Realizar todo o processo da mutação em um ambiente único; (*ii*) Diminuir o tempo gasto em todo o processo; (*iii*) Utilizar projetos diretamente do GitHub ou um arquivo .zip de um projeto; (*iv*) Realizar a mutação e aplicar os casos de teste nos mutantes gerados; (*v*) Analisar a possível equivalência dos mutantes sobreviventes; (*vi*) Disponibilizar a visualização do local onde a mutação foi realizada; (*vii*) Definir a equivalência entre os códigos original e mutante; (*viii*) Indicar o grau de dificuldade em analisar cada mutante individualmente.

Para verificar o uso da ferramenta durante a análise dos mutantes, foi realizado um experimento com onze alunos de pós-graduação do curso de Ciência da Computação de uma Universidade Federal. Os alunos analisaram os códigos de duas maneiras distintas: *(i)* utilizando a DiffMutAnalyze; e *(ii)* manualmente, sem a utilização da ferramenta. Os alunos tiveram uma aula que pudesse retratar o conhecimento básico sobre o teste de mutação. Os alunos puderam conhecer a teoria sobre esse critério de teste, conhecer o funcionamento da geração da mutação e verificar exemplos de mutações realizadas. Além disso, os alunos foram instruídos sobre o problema dos mutantes equivalentes e o que

eles causam no escore de mutação, caso não sejam identificados. Exemplos de mutantes equivalentes foram exibidos aos alunos para que eles pudessem entender a maneira que eles se comportam no código mutante. Passado o conhecimento necessário para que os alunos compreendessem o que é o teste de mutação e como ele é realizado, foi conduzido o experimento para a análise dos códigos original e mutante. Os alunos foram instruídos como deveriam analisar a possível equivalência entre os códigos por meio de exemplos e demostrações de análises.

3.1. Procedimentos do Experimento

Os onze alunos foram divididos em dois grupos, sendo um grupo realizando primeiramente a análise dos mutantes utilizando a DiffMutAnalyze e posteriormente fazendo a análise manualmente, sem o auxilio da ferramenta. Ou seja, utilizando apenas uma ferramenta de texto para visualização dos códigos original e mutante, registrando em um local a parte, se os mutantes são equivalentes ao código original. O segundo grupo fez o processo inverso, iniciando pela análise manual e depois fazendo o uso da ferramenta.

O primeiro grupo (sujeitos S5, S6, S9, S10 e S11) iniciou a análise pela DiffMutAnalyze, eles analisaram os mutantes gerados do algoritmo *InsertionSort*. O segundo grupo, composto pelos sujeitos S1, S2, S3, S4, S7 e S8, iniciou a análise manual com os mutantes gerados do algoritmo *SelectionSort*. Ao final da análise dos mutantes de cada algoritmo, os grupos inverteram a forma de análise. Dessa forma, o algoritmo *InsertionSort* foi utilizado somente na DiffMutAnalyze e os dois grupos analisaram todos os mutantes desse algoritmo. O algoritmo *SelectionSort* foi utilizado na análise manual e todos os seus mutantes foram analisados por ambos os grupos.

Um dos fatores para medir o custo humano para analisar os mutantes é por meio do tempo gasto com a análise. Dessa forma, quando os sujeitos analisaram os mutantes utilizando a DiffMutAnalyze, a própria ferramenta fazia a contagem do tempo. Ao analisar os mutantes sem a utilização da DiffMutAnalyze, o tempo foi cronometrado e anotado. Foram gerados 26 mutantes do algoritmo *SelectionSort* e 38 mutantes do algoritmo *InsertionSort*. Os sujeitos analisaram os códigos e indicaram se possuía equivalência com o código original e o grau de dificuldade em analisar esses mutantes.

3.2. Resultados

Os onze sujeitos analisaram 38 mutantes do algoritmo *InsertionSort* na DiffMutAnalyze. Como pode ser observado na Tabela 1, os sujeitos gastaram em média 14 minutos para analisar todos os mutantes e, considerando o tempo médio para analisar cada mutante, os sujeitos analisaram em 21 segundos cada mutante. O maior tempo de análise gasto foi do sujeito S11, gastando 19 minutos para analisar todos os mutantes, obtendo o tempo médio de 30 segundos por mutante. Considerando o menor de tempo de análise, o sujeito S3 gastou 10 minutos e 1 segundo para analisar os 38 mutantes, atingindo o tempo médio de 16 segundos por mutante.

Em relação ao tempo gasto com a análise manual os sujeitos analisaram 26 mutantes gerados do algoritmo *SelectionSort*, com exceção do sujeito S7 que analisou 20 mutantes, sendo por escolha do próprio participante não analisar todos os mutantes (Tabela 2). Para medir o tempo dessa análise manual, foi considerado o tempo inspecionando cada mutante individualmente e o tempo total gasto com a análise, sendo: o tempo de abertura

Tabela 1. Tempo gasto com a análise utilizando a DiffMutAnalyze.

Sujeito	Quant. Mutantes	Tempo total gasto	Tempo médio por mutante
S1	38	00:12:07	00:00:19
S2	38	00:12:00	00:00:19
S3	38	00:10:01	00:00:16
S4	38	00:15:05	00:00:24
S5	38	00:16:05	00:00:25
S6	38	00:11:08	00:00:17
S7	38	00:10:06	00:00:16
S8	38	00:14:08	00:00:22
S9	38	00:14:03	00:00:22
S10	38	00:16:07	00:00:25
S11	38	00:19:00	00:00:30
Tempo médio		00:14:00	00:00:21

da classe mutante, uma vez que cada mutante é gerado em uma classe diferente e o tempo para identificar o local da mutação. O motivo pelo qual esses tempos foram contabilizados a parte nessa análise, é para comparar com o processo efetuado pela DiffMutAnalyze, uma vez que ela busca os mutantes automaticamente na pasta do projeto e os disponibiliza para o usuário evidenciando o local da alteração realizada, sendo necessário apenas clicar no botão "Próximo". Dessa forma, na Tabela 2 há duas colunas de tempo, a coluna "Tempo de análise" considera somente o tempo em que o mutante foi inspecionado, ou seja, o tempo de verificação se são equivalentes. A coluna "Tempo total gasto" considera todo o tempo utilizado para o processo da análise, conforme mencionado anteriormente sobre esse processo, e incluindo o tempo de análise.

Tabela 2. Tempo gasto com a análise sem a utilização da DiffMutAnalyze.

Sujeito	Quantidade de mutantes analisados	Tempo de análise	Tempo total gasto	Tempo total médio por mutante
S1	26	00:28:40	00:50:00	00:01:58
S2	26	00:09:36	00:40:00	00:01:31
S3	26	00:11:14	00:26:00	00:01:00
S4	26	00:26:27	00:53:00	00:02:03
S5	26	00:10:20	00:21:00	00:00:48
S6	26	00:04:48	00:22:00	00:00:51
S7	20	00:16:52	00:25:00	00:01:25
S8	26	00:14:32	00:26:00	00:01:00
S9	26	00:10:52	00:29:00	00:01:06
S10	26	00:06:22	00:21:00	00:00:48
S11	26	00:11:15	00:26:00	00:01:00
Tempo médio		00:12:50	00:31:30	00:01:06

Os sujeitos gastaram em média aproximadamente 13 minutos para comparar os códigos dos mutantes com o código original e 31 minutos para realizar todo o procedimento da análise. O tempo médio gasto com a análise manual de cada mutante, considerando o tempo total, é de 1 minuto e 6 segundos. Para analisar todos os mutantes e considerando o tempo total gasto, o sujeito S4 gastou 53 minutos para efetuar todo o processo de análise, obtendo o tempo médio por mutante de aproximadamente 2 minutos. O menor tempo total de análise é dos sujeitos S5 e S10, contabilizando 21 minutos para analisar os mutantes, com o tempo médio de 48 segundos por mutante.

Como o objetivo da análise era definir se os mutantes são equivalentes ao código

original, a seguir estão descritas as quantidades de mutantes que foram definidos como equivalentes e não-equivalentes ao código original, para as duas formas de análises do experimento. Na Tabela 3, estão contidas as duas análises nas colunas "DiffMutAnalyze" e "Análise Manual". Os mutantes definidos como equivalentes estão na coluna denominada **M_EQ** e os mutantes não-equivalentes na coluna **M_N-EQ**.

Tabela 3. Quantidade de mutantes definidos como equivalentes.

Sujeito	*DiffMutAnalyze*			*Análise Manual*		
	M_EQ	**M_N-EQ**	**% Acerto**	**M_EQ**	**M_N-EQ**	**% Acerto**
S1	0	38	84,2	2	24	73,1
S2	2	36	78,9	6	20	88,5
S3	5	33	71,1	7	19	84,6
S4	5	33	76,3	5	21	53,8
S5	3	35	76,3	4	22	73,1
S6	2	36	78,9	6	20	88,5
S7	3	35	73,7	2	18	55,0
S8	1	37	86,8	0	26	73,1
S9	4	34	73,7	3	23	84,6
S10	7	31	65,8	5	21	69,2
S11	19	19	44,7	13	13	69,2

Como descrito na Tabela 3, dez sujeitos definiram algum mutante como equivalente ao código original do algoritmo *InsertionSort*. Considerando o percentual de acerto dos participantes, o sujeito S8 obteve o maior percentual de acerto, com 86,8%. Em seguida, o sujeito S1 obteve 84,2% de acerto. O sujeito S11 obteve o menor percentual de acerto, acertando 44,7% dos mutantes. Os demais sujeitos acertaram entre 60% e 80% dos mutantes. Na análise manual, sem a utilização da ferramenta DiffMutAnalyze, os sujeitos definiram alguns mutantes como equivalentes (Tabela 3). Os sujeitos com o maior percentual de acertos durante a análise foram o sujeito S2 e o S6, atingindo 88,5% das asserções cada um deles. Os sujeitos S3 e S9 obtiveram 84,6% de acertos cada. O menor percentual de acerto foi do sujeito S4, com 53,8% de mutantes definidos corretamente.

Para o grau de dificuldade da análise dos mutantes, na Tabela 4 está descrita a quantidade de cada nível, definido para o grau de dificuldade, em que cada sujeito informou para as análises realizadas na DiffMutAnalyze. A maioria dos sujeitos definiu o grau de dificuldade em analisar os mutantes do algoritmo *IsertionSort* como fáceis de serem analisados no geral.

Tabela 4. Grau de dificuldade da análise utilizando a DiffMutAnalyze.

Sujeito	Muito Fácil	Fácil	Médio	Difícil	Muito Difícil
S1	0	1	36	1	0
S2	30	7	1	0	0
S3	28	7	3	0	0
S4	0	19	15	4	0
S5	11	14	7	6	0
S6	26	8	3	1	0
S7	17	13	8	0	0
S8	13	22	2	1	0
S9	30	5	1	2	0
S10	0	34	4	0	0
S11	5	9	14	7	3

Os sujeitos S2, S3, S5, S6, S7, S8, S9 e S10 definiram maior quantidade de mu-

tantes como sendo muito fácil ou fácil se comparado com os outros níveis de dificuldade médio, difícil ou muito difícil. O sujeito S1 definiu a maioria dos mutantes como dificuldade média, sendo 36 mutantes considerados nesse nível de dificuldade. O sujeito S4 definiu metade dos mutantes como sendo fácil de analisar e 15 mutantes como médio e quatro como sendo difícil de serem analisados. O sujeito S11 definiu maior parte dos mutantes como sendo médio, difícil ou muito difícil de serem analisados, somando 24 mutantes definidos nesses níveis. Os outros 14 mutantes foram definidos como muito fácil ou fácil de serem analisados.

Na Tabela 5 estão descritas as quantidades de mutantes analisados manualmente, ou seja, sem o uso da DiffMutAnalyze, distribuídos em cada nível de dificuldade. O mesmo cenário se repete nessa análise. A maioria dos sujeitos consideraram os mutantes do algoritmo *SelectionSort* fáceis de serem analisados. Dessa forma, os mesmos sujeitos que definiram maior quantidade de mutantes como sendo fáceis de serem analisados na ferramenta DiffMutAnalyze, também definiram na análise manual, ou seja, são os sujeitos S2, S3, S5, S6, S7, S8, S9 e S10. O sujeito S1 definiu o grau de dificuldade fácil para cinco mutantes e os outros 21 mutantes como sendo médio, difícil ou muito difícil de serem analisados. O sujeito S4 definiu 12 mutantes como muito fácil ou fácil e os outros 14 mutantes como médio ou difícil de serem analisados. Por fim, o sujeito S11 definiu 8 mutantes como fáceis e os demais mutantes como médio, difícil ou muito difícil de verificar sua equivalência com o código original, somando 18 mutantes.

Tabela 5. Grau de dificuldade da análise sem a utilização da DiffMutAnalyze.

Sujeito	Muito Fácil	Fácil	Médio	Difícil	Muito Difícil
S1	0	5	16	3	2
S2	17	7	1	1	0
S3	15	6	5	0	0
S4	2	10	10	4	0
S5	9	11	3	2	1
S6	20	4	1	1	0
S7	6	13	1	0	0
S8	10	16	0	0	0
S9	18	5	3	0	0
S10	0	34	4	0	0
S11	0	8	12	5	1

4. Ameças à Validade

Validade interna: *Nível de Experiência dos Participantes:* o conhecimento dos participantes em relação ao projeto utilizado para realizar a mutação pode afetar os dados coletados durante a análise. Para mitigar essa ameaça, foram utilizados algoritmos de ordenação para tentar abranger o conhecimento prévio dos participantes para a análise dos códigos original e mutante, uma vez que os participantes são alunos de pós-graduação do curso de Ciência da Computação.

Validade da Construção: *Marcação do tempo da análise*: existe a possibilidade da marcação do tempo da análise não ser real, pois os participantes podem realizar alguma ação que não seja relacionada com a análise dos códigos, fazendo com que o tempo da análise daquele determinado mutante seja corrente até que o participante prossiga para a próxima análise. Para mitigar essa ameaça, foi colocado na ferramenta um botão de ”pause” para

que os participantes pudessem parar a contagem do tempo em caso de realizar outra ação que não esteja relacionada com a análise dos mutantes.

5. Conclusão

Foi possível observar que os alunos tiveram interesse em conhecer sobre o teste de mutação, e também se manifestaram entusiasmados em poder ver como o teste de mutação funciona na prática, o que torna o aprendizado mais dinâmico. Como foi demonstrado nos resultados das análises, a escolha da utilização dos algoritmos de ordenação para aplicar a mutação, facilitou o processo da análise, uma vez que esses algoritmos são conhecidos por grande parte dos participantes.

Por meio do resultado da análise, é possível perceber que há influência quando o sujeito define um mutante como não-equivalente, sendo que o mesmo possui equivalência com o código original, pois o analista gastará tempo tentando criar um caso de teste para tentar matar o determinado mutante, uma vez que não há teste de unidade que consiga identificar a alteração de um mutante equivalente. Em contrapartida, se o contrário ocorre, o analista não criará um novo caso de teste para tentar matar esse mutante, assim, estará comprometendo a qualidade do conjunto de testes. Dessa forma, quando o sujeito comete erros durante a análise dos mutantes, tem como consequência o tempo e a qualidade, uma vez que podem ser gastos esforços sem a garantia de sucesso e a qualidade pode ser comprometida quando não se produz novos casos de testes que seriam necessários.

Referências

Aniche, M., Hermans, F., and van Deursen, A. (2019). Pragmatic software testing education. In *50th Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE)*, pages 414–420.

Botelho, J., Pereira, C. H., Durelli, V. H. S., and Durelli, R. S. (2018). Diffmutanalyze: Uma abordagem para auxiliar a identificação de mutantes equivalentes. In *VI Workshop on Software Visualization, Evolution and Maintenance (VEM)*.

Cerf, V. G. (2014). Responsible programming. *Communications of the ACM*, 57(7):7–7.

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

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.

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.

Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Traon, Y. L., and Harman, M. (2019). Mutation testing advances: an analysis and survey. In *Advances in Computers*, pages 275–378.

Radhakrishnan, P., Kanmani, S., and Nandhini, M. (2015). Xsoft: A generic software teaching and learning model. *Computer Applications in Engineering Education*, 23(3):432–442.

Spacco, J. and Pugh, W. (2006). Helping students appreciate test-driven development (tdd). In *21st Symposium on Object-oriented Programming Systems, Languages, and Applications (ACM SIGPLAN)*, pages 907–913.

Zheng, W., Bai, Y., and Che, H. (2018). A computer-assisted instructional method based on machine learning in software testing class. *Computer Applications in Engineering Education*, 26(5):1150–1158.