TERTS: Um tutor de ensino para refatoração de *test smells*

**Antônia Deigela L. Rufino, Victor Anthony P. Alves,
Lara Gabrielly S. B. Lima, João Paulo F. Queiroz, Carla Bezerra,
Ivan Machado, Emanuel Coutinho**

[1]Universidade Federal do Ceará (UFC) - Campus Quixadá

{deigelalima, victorpa, laragabrielly, joaopfq}@alu.ufc.br

carlailane@ufc.br, ivan.machado@ufba.br, emanuel.coutinho@ufc.br

Abstract. *Teaching test smells refactoring has been a challenging practice in the academic context. This paper introduces TERTS, a tutor for teaching and practicing test smells refactoring developed for technology students on a web platform. The article describes the development of the platform and its acceptance by students. Validation of the tool included measuring (i) acceptance and use, (ii) students' acquisition of skills during and after use, and (iii) students' level of knowledge about identifying and refactoring test smells before and after using the tool. The results indicate that TERTS is an effective and promising tool to be implemented in the educational environment.*

Resumo. *O ensino de refatoração de test smells vem sendo uma prática desafiadora no contexto acadêmico. Este trabalho introduz o TERTS, um tutor de ensino e prática de refatoração de test smells desenvolvido para estudantes de tecnologia em uma plataforma web. O artigo descreve o desenvolvimento da plataforma e sua aceitação pelos estudantes. A validação da ferramenta incluiu a medição da (i) aceitação e uso, (ii) aquisição de habilidades pelos alunos durante e após o uso, e (iii) nível de conhecimento dos alunos sobre identificação e refatoração de test smells antes e após a utilização da ferramenta. Os resultados indicam que o TERTS é uma ferramenta eficaz e promissora para ser implementada no ambiente educacional.*

Palavras-chave: Tutor de Ensino, Refatoração, *Test Smells*

1. Introdução

O desenvolvimento de software tem se tornado cada vez mais complexo, demandando a utilização de técnicas e práticas que visam garantir a qualidade do produto final e a satisfação do cliente [Durelli et al. 2019]. Segundo [Kumar and Mishra 2016], o teste de software é uma atividade essencial que visa identificar defeitos no código, assegurando que o software funcione corretamente e atenda aos requisitos estabelecidos.

Entretanto, a crescente complexidade dos sistemas de software acarreta a existência de *test smells* ou sinais de problemas nos casos de teste, prejudicando a efetividade desses testes [Kim 2020]. Esses problemas podem incluir redundâncias, testes duplicados, falta de clareza ou legibilidade, entre outros aspectos que prejudicam a manutenibilidade e a confiabilidade dos casos de teste [Peruma et al. 2019].

Para solucionar esses problemas, a refatoração de *test smells* tem sido utilizada para melhorar a qualidade dos casos de teste, tornando-os mais legíveis e eficazes na detecção de falhas [Palomba et al. 2018]. Entre as técnicas de refatoração de *test smells*, destaca-se o uso de testes de unidade, que verificam o comportamento de partes individuais do software [Sun et al. 2019]. No entanto, ensinar essa prática é desafiador, exigindo um equilíbrio entre conhecimentos teóricos e práticos [Garousi et al. 2020, Bai et al. 2021]. É essencial fornecer aos estudantes de tecnologia as ferramentas e metodologias adequadas para identificar, compreender e corrigir esses problemas de maneira efetiva [Haendler et al. 2020, Bai et al. 2021].

Este trabalho visa apresentar o TERTS (Tutor de Ensino para Refatoração de *Test Smells*), direcionado a estudantes de tecnologia. A plataforma oferece exercícios práticos em Java que auxiliam os alunos a compreender e aprimorar a identificação e refatoração de práticas inadequadas presentes nos códigos de teste de unidade, que seriam os *test smells*, promovendo conhecimento para obter qualidade em seus projetos de software.

2. Fundamentação Teórica

2.1. Refatoração

A refatoração é um processo de reestruturação do código-fonte, visando melhorar sua qualidade sem alterar seu comportamento externo [Ivers et al. 2022]. Através do ensino de refatoração, os desenvolvedores são capacitados a identificar e corrigir problemas no código, tornando-o mais fácil de entender, modificar e reutilizar [Smith et al. 2006]. Ela tem recebido considerável atenção tanto no meio acadêmico quanto na indústria de software [Kaur et al. 2021]. Essa habilidade é essencial para o desenvolvimento de software sustentável a longo prazo, uma vez que a complexidade excessiva pode levar a problemas de manutenção e dificultar a colaboração entre desenvolvedores [Winters et al. 2020, Hofbauer et al. 2022].

2.2. *Test Smells*

Os *test smells* referem-se a problemas específicos que podem surgir no código de teste e prejudicar a legibilidade e a capacidade de manutenção do código [Santana et al. 2020]. Os *test smells* podem se manifestar de diferentes formas, como falta de estruturação no código de teste e lógica excessivamente complexa. A falta de estruturação pode resultar em testes longos e duplicação de código, dificultando a compreensão e manutenção do código [Kim et al. 2021]. Já a lógica complexa pode tornar os testes mais difíceis de entender e aumentar o risco de erros. Simplificar a lógica e estruturar adequadamente os casos de teste são medidas importantes para evitar esses *test smells* [Spadini et al. 2018].

O conceito de *test smells*, que se refere a testes mal projetados ou mal concebidos [Bavota et al. 2012] foi destacado por [Van Deursen et al. 2001], onde definiram um catálogo de 11 *test smells*: *Mystery Guest, Resource Optimism, Test Run War, General Fixture, Eager Test, Lazy Test, Assertion Roulette, Indirect Testing, For Testers Only, Sensitive Equality, e Test Code Duplication.* De forma mais recente, [Peruma et al. 2019] propuseram mais 12 tipos de *test smells*: *Conditional Test Logic, Constructor Initialisation, Default Test, Duplicate Assert, Empty Test, Exception Handling, Ignored Test, Magic Number Test, Redundant Assertion, Redundant Print, Sleepy Test e Unknown Test.* Neste artigo, serão abordados os seguintes *test smells*: *Assertion Roulette, Duplicate Assert, Ignored Test* e *Eager Test*.

3. Trabalhos Relacionados

[Damasceno and Bezerra 2023] descrevem uma experiência de ensino cujo objetivo é instruir os estudantes de Engenharia de Software na prática da detecção e remoção de *test smells*. Os autores investigaram as habilidades adquiridas pelos alunos e as dificuldades enfrentadas durante o processo de refatoração. Os resultados ressaltaram que essa experiência impactou positivamente os alunos na aquisição de novas habilidades pessoais e interpessoais. Além disso, os autores recomendaram mais práticas relacionadas à detecção e refatoração de *test smells*.

[Aljedaani et al. 2023] abordam o impacto dos *test smells Assertion Roulette* e *Eager Test* nas habilidades de solução de problemas e depuração de código por parte dos estudantes. Os autores realizaram um experimento para avaliar como a presença desses *test smells* afeta o tempo que os alunos levam para corrigir erros no código de produção. Os resultados indicaram que os participantes que receberam conjuntos de testes contendo esses *smells* levaram mais tempo para concluir a tarefa de correção de erros do que aqueles que receberam conjuntos de testes sem esses problemas.

[Keuning et al. 2021] apresentam o *Refactor Tutor*, um sistema de tutoria que visa ajudar programadores iniciantes a compreender e aplicar melhorias no código-fonte de maneira mais eficaz e auxiliar alunos durante a realização de exercícios de melhoria de código de produção. Já [Prokic et al. 2021], apresentam o *Clean CaDET*, uma plataforma que lida com a busca de *code smells* no desenvolvimento de software. A plataforma possui dois módulos: o *Smell Detector*, que usa Inteligência Artificial (IA) para avaliar o código em tempo real, e o *Smart Tutor*, que fornece conteúdo educacional para ajudar os desenvolvedores a resolver problemas de código e utiliza estratégias como *feedback* oportuno e *e-learning* adaptativo.

[Bai et al. 2021] abordam a forma como os estudantes realizam testes unitários em desenvolvimento de software. Os pesquisadores investigaram as percepções, práticas e dificuldades enfrentadas pelos estudantes ao realizar testes unitários. Eles exploraram questões como padrões de qualidade percebidos pelos alunos em testes unitários e desafios enfrentados na criação e manutenção de testes. O estudo fornece descobertas relevantes para a educação em testes de software e destaca a importância de integrar a prática de testes nas disciplinas de Ciência da Computação e Engenharia de Software.

Os trabalhos relacionados têm uma conexão direta com o desenvolvimento do tutor de ensino de refatoração de *test smells*. Tanto o presente trabalho quanto a pesquisa de [Damasceno and Bezerra 2023] abordam técnicas de refatoração de *test smells*. Além disso, os estudos de [Damasceno and Bezerra 2023] e [Aljedaani et al. 2023] exploram as habilidades dos estudantes na depuração e identificação dos *test smells Eager Test* e *Assertion Roulette*, fatores cruciais para direcionar o foco do trabalho sobre quais *smells* são considerados mais complexos e necessitam de atenção. As conclusões do estudo de [Bai et al. 2021] reforçam a importância de um tutor de ensino que auxilie os estudantes na prática de teste de software, destacando a necessidade de educação nessa área para a formação de profissionais mais capacitados. Embora os trabalhos de [Keuning et al. 2021] e [Prokic et al. 2021] forneçam funcionalidades fundamentais de ferramentas voltadas ao ensino e à prática de refatoração de código, eles não abordam diretamente o contexto de *test smells*. Portanto, é notável que, embora a literatura reconheça a importância de ensinar as práticas de refatoração de *test smells* e de abordar essas ati-

vidades no contexto educacional, nenhuma ferramenta voltada para os estudantes foi desenvolvida para aplicar esses conceitos no meio acadêmico.

4. Tutor de ensino TERTS

4.1. Desenvolvimento do tutor de ensino

O TERTS foi construído com a proposta de instruir os alunos nas técnicas de refatoração de *test smells* na linguagem Java, baseando-se em técnicas extraídas da literatura. Para a edição do código de teste pelos usuários, foi utilizado o CodeMirror[1], componente de edição de código para web, o qual possibilita a utilização de um editor para a manipulação de texto na própria interface. A identificação dos *test smells* nos exercícios práticos abordados pelo TERTS foi realizada com o auxílio da ferramenta JNose Test [Virgínio et al. 2020] e os seus respectivos códigos foram armazenados em um repositório no GitHub[2]. A seleção dos tipos de *test smells* que foram abordados pelo TERTS baseou-se em critérios advindos da literatura, bem como nos tipos de *smells* considerados mais desafiadores de refatorar, conforme constatado no experimento conduzido por [Damasceno and Bezerra 2023]. Assim, definiu-se que a ferramenta abordaria os seguintes *test smells*: *Assertion Roulette, Duplicate Assert, Ignored Test* e *Eager Test*. Desse modo, procedeu-se o desenvolvimento das seguintes funcionalidades da plataforma:

- Exibir conceitos dos quatro tipos de *test smell* selecionados (*Assertion Roulette, Duplicate Assert, Ignored Test* e *Eager Test*).
- Exibir um conjunto de exercícios para cada *test smells* definidos anteriormente.
- Receber dos usuários a resolução dos exercícios refatorados.
- Realizar um comparativo do exercício refatorado pelo usuário com o exercício refatorado no sistema e validado pelo JNose Test. É importante ressaltar que o sistema não leva em consideração detalhes de identação na comparação, tais como espaços, quebras de linha, texto em caixa alta ou baixa, entre outros.
- Fornecer dicas que são equivalentes as regras de refatoração baseadas no trabalho de [Van Deursen et al. 2001], um exemplo de dicas presentes no tutor:
  - **Dica:** Para refatorar o *Assertion Roulette*, é essencial incluir um comentário no parâmetro do *assert*, o que ajuda na identificação de erros durante a execução do teste, um comentário claro facilita a identificação da causa do erro, economizando tempo na depuração. Exemplo de *assert* no código de teste:
    ```
    assertTrue(verificarDataDeNascimento (dataDeNascimento), "A data de nascimento não é válida");
    ```
- Fornecer *feedback* sobre as soluções submetidas pelos usuários. Por exemplo, se o usuário corrigiu apenas uma parte do *test smell*, o sistema retorna um *feedback* negativo, indicando que o exercício não foi concluído, como na mensagem:
  `Ainda existem smells no teste.`

4.2. Arquitetura do tutor de ensino

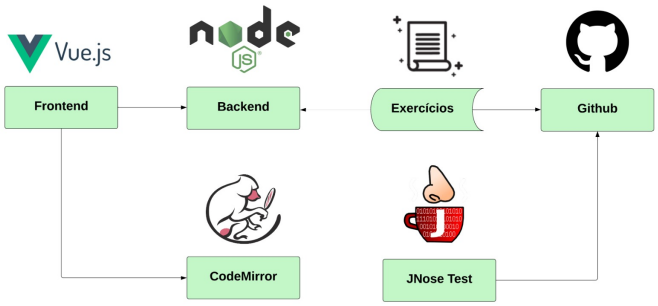
A arquitetura do TERTS, conforme mostrado na Figura 1, utiliza tecnologias como Vue.js, Node.js, Github e JNose Test. No *frontend*, construído com TypeScript e Vue.js, os usuários têm acesso a um editor de texto, CodeMirror, para refatorar exercícios e receber

[1]https://codemirror.net/

[2]https://github.com/DeigelaLima/Exercicios_Tutor_de_Ensino_TS

feedbacks. O *backend* verifica as refatorações submetidas e fornece *feedbacks* processados utilizando JavaScript e Node.js. A ferramenta JNose Test é empregada para identificar *test smells* nos códigos de teste de unidade, que são importados de um repositório no Github[1]. Após a identificação, uma análise manual é realizada pelos desenvolvedores, examinando as linhas de código afetadas e os tipos de *test smells* presentes. O aluno interage apenas com a interface do TERTS, sem executar essas etapas manualmente.

Figura 1. Arquitetura do tutor de ensino



4.3. Interface da ferramenta TERTS

Durante o desenvolvimento da interface da ferramenta, priorizou-se a máxima usabilidade para garantir que as informações fossem apresentadas de forma clara e acessível aos estudantes. Os trechos de exercícios foram organizados para proporcionar uma progressão natural no aprendizado, permitindo que os alunos desenvolvam suas habilidades a medida que resolvem as atividades. A Figura 2 apresenta a tela inicial do TERTS. O usuário pode escolher entre diferentes tipos de *test smells* para explorar. A escolha o encaminha para uma segunda tela em que pode-se selecionar o exercício. Após a escolha do exercício, o usuário é encaminhado para uma terceira tela onde pode realizar a refatoração do *test smells*. Após todas as refatorações, o sistema exibe a opção de continuar aprendendo outros tipos de *test smells* ou sair da plataforma. O TERTS foi hospedado em um servidor web[4] e está disponível para os estudantes.

Figura 2. Tela principal da ferramenta TERTS

4.4. Exemplo de uso da ferramenta TERTS

A Figura 3 representa a tela onde é exibido o conjunto de exercícios correspondente ao *test smell* selecionado na Figura 2. Do lado esquerdo da tela, há a descrição da identificação

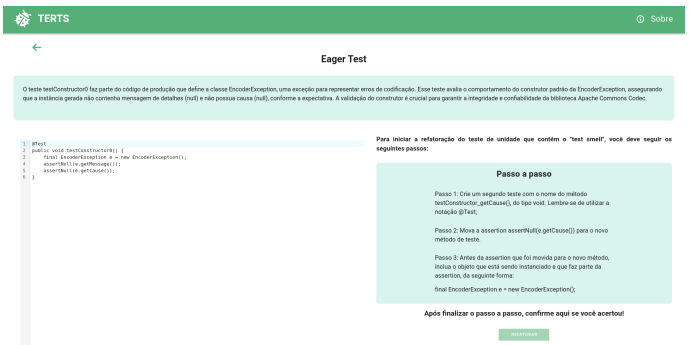
[1]https://github.com/apache/commons-codec
[4]URL do TERTS: https://tutor-de-ensino-ts.vercel.app

do comportamento do *test smell* dentro do código de teste e o método de refatoração utilizado para refatorá-lo, segundo [Deursen et al. 2001]. Do lado direito, há o conjunto de três exercícios que o aluno pode escolher aleatoriamente para começar a refatorar.

Figura 3. Tela de escolha do exercício

A Figura 4, corresponde a tela que ao selecionar um determinado *test smells* na tela da Figura 2 é apresentada a descrição do teste e a classe do código de produção associada a ele. No lado esquerdo da tela, encontra-se o editor CodeMirror, onde o exercício é carregado do *backend* e exibido no *frontend*. À direita, estão listados os passo a passo que o aluno deve seguir para refatorar o teste com sucesso. Após seguir todos os passos corretamente, o aluno deve clicar no botão de refatorar localizado abaixo dos passos. Se todas as etapas forem seguidas corretamente, um *pop-up* será exibido com *feedback* indicando sucesso; caso contrário, será exibido o *feedback* de erro e o aluno poderá continuar refatorando até corrigir todos os *smells* presentes no teste de unidade.

Figura 4. Tela de refatoração do *test smell*



5. Avaliação do tutor TERTS

Após o desenvolvimento da ferramenta, procedeu-se a etapa de validação da proposta da ferramenta pelos alunos da Universidade Federal do Ceará (UFC) - Campus Quixadá com um grupo de estudantes voluntários, a maioria alunos dos cursos de Ciência da Computação, Engenharia de Software e Sistemas de Informação. Desse modo, foi realizada uma medição da satisfação na utilização da plataforma e do perfil dos estudantes em três aspectos: (i) aceitação e uso da ferramenta, (ii) habilidades adquiridas durante / após o uso da ferramenta, (iii) nível de conhecimento sobre identificação e refatoração de *test*

smells antes / após o uso da ferramenta. Na primeira etapa, foi elaborado um questionário *online* para avaliar o perfil prévio dos estudantes que usariam o TERTS, abordando o nível de conhecimento dos alunos em testes de unidade, *test smells* e refatoração de testes. Também foram incluídas questões de autoavaliação dos conhecimentos em refatoração de *test smells*. O questionário, aplicado via *Google Forms*, forneceu informações importantes sobre os diferentes perfis de usuários do TERTS.

- 81,8% dos estudantes não tinham conhecimentos sobre *test smells*.
- Apenas 9,1% dos estudantes já tinham refatorado *test smells* durante a graduação.
- Nenhum aluno tinha utilizado ferramentas de ensino de refatoração de *test smells*.
- 90% dos alunos se autoavaliaram como inseguros quanto a detecção e refatoração de *test smells*.

Após a aplicação do questionário prévio, foi elaborado um treinamento com os alunos voluntários para introduzir os conceitos de *test smells* e de refatoração de código de teste. É importante destacar que o treinamento foi inteiramente teórico, uma vez que a prática seria realizada por meio da utilização ferramenta TERTS. Em seguida, foi reservado um tempo para que os alunos utilizassem a ferramenta para praticar e adquirir conhecimentos. Eles foram encorajados a resolver o maior número possível dos 12 exercícios disponíveis no TERTS.

5.1. Aceitação e utilização da ferramenta

Para compreender a receptividade do TERTS pelos estudantes após sua utilização, foram analisadas as respostas do questionário relacionadas ao Modelo de Aceitação de Tecnologia (TAM), totalizando 16 respostas dos alunos. O TAM é um modelo teórico amplamente utilizado para avaliar a aceitação e utilização de tecnologias pelos usuários [Davis et al. 1989]. Para medir o nível de satisfação e concordância dos alunos, foi utilizada a escala *Likert* [Likert 1932] de cinco pontos, indo de (1) Discordo Totalmente a (5) Concordo Totalmente. Essa escala permite uma avaliação da opinião dos alunos sobre aspectos da ferramenta, incluindo usabilidade, eficácia e utilidade. O formulário com o modelo de avaliação das questões referentes ao TAM estão disponíveis em um repositório[2]. Na Figura 5 estão detalhadas as análises das perguntas do TAM em relação à utilidade percebida, facilidade de uso e previsão de uso futuro.

Figura 5. Gráficos com os resultados do modelo TAM na Escala *Likert*

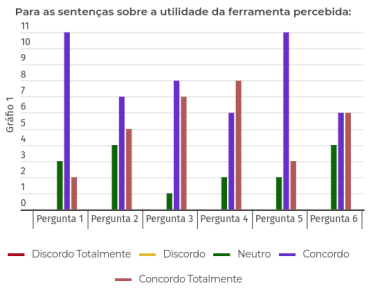


Gráfico 1

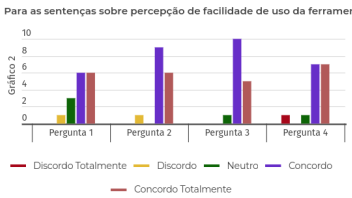


Gráfico 2

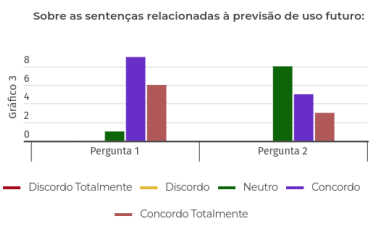


Gráfico 3

[2]https://zenodo.org/records/11205338

No Gráfico 1, pode-se observar que a maioria dos alunos concorda que a ferramenta melhora o controle do código de teste de unidade, aprimora o desempenho na refatoração de testes de unidade, ajuda a melhorar a qualidade do trabalho com refatoração de *test smells*, aumenta o conhecimento sobre refatoração de *test smells* e economiza tempo em projetos que envolvem essa atividade. No Gráfico 2, as respostas revelam que maioria dos alunos concorda ou concorda totalmente que a interação com a ferramenta é de fácil compreensão, o que sugere uma perceção de que ela é intuitiva e de simples utilização. Além disso, os alunos também concordam ou concordam totalmente que a ferramenta fornece informações úteis para implementar diferentes tipos de refatoração de *test smells*. Embora haja uma distribuição mais equilibrada de respostas quanto à facilidade geral de uso da ferramenta, a maioria ainda concorda que ela é fácil de usar.

No Gráfico 3 fica evidente um interesse significativo na utilização da ferramenta proposta como uma ferramenta de aprendizado sobre *test smells*. A maioria dos alunos concorda ou concorda totalmente que usaria a ferramenta para este fim. No entanto, em relação à preferência pela visão da ferramenta sobre refatoração de *test smells* em comparação com outras ferramentas ou *plugins*, observa-se uma divisão mais equilibrada de opiniões. Embora uma parcela significativa dos alunos não apresente uma preferência clara, uma minoria discorda ou discorda totalmente, indicando uma variedade de perspectivas entre os alunos. Essas análises destacam o interesse geral na ferramenta educacional proposta e também destacam a importância de considerar as preferências individuais dos alunos na busca por soluções educacionais.

Descoberta 1: O TERTS é uma ferramenta fácil de utilizar e suas informações estão úteis e claras para os estudantes.

5.2. *Test smells* mais difíceis de refatorar, segundo os alunos

Foram analisadas as respostas do questionário obtidas depois da refatoração dos *test smells* após a experiência prática que os alunos tiveram com a ferramenta, para identificar os tipos de *test smells* mais difíceis de refatorar. Na Tabela 1, a primeira coluna representa o tipo de *test smells* que o tutor aborda e a segunda coluna em porcentagem das vezes que o *test smells* foi considerado difícil de se refatorar.

Tabela 1. Os *test smells* mais difíceis de refatorar na perspectiva dos alunos

Test Smells	Quantidade (%)
Assertion Roulette	25%
Duplicate Assert	68,8%
Eager Test	37,5%
Ignored Test	0%

A partir da análise da Tabela 1, infere-se que o *Duplicate Assert* foi identificado como difícil em 68,8% das respostas, seguido pelo *Eager Test*, com 37,5%. Além disso, também observa-se que o *Ignored Test* não foi considerado difícil de refatorar em nenhuma das respostas. Mediante essa análise, infere-se que os níveis de dificuldade dos *smells* abordados no TERTS variam de acordo com tipo.

Descoberta 2: O TERTS aborda o ensino de refatoração tanto de *test smells* considerados fáceis, quanto de *smells* considerados mais complexos. Isso estimula a progressividade do aprendizado de quem o utiliza.

5.3. Habilidades adquiridas pelos estudantes durante / após o uso da ferramenta

Algumas das habilidades-chave que o tutor visa fornecer aos estudantes foram identificadas e incluídas no questionário *online*. Após a utilização do TERTS, os alunos puderam avaliar quais dessas habilidades adquiriram. Uma análise qualitativa das respostas coletadas revelou cinco categorias principais de habilidades desenvolvidas pelos estudantes, conforme indicado na Tabela 2. Analisando a Tabela 2, percebe-se que a compreensão do código de teste (13) e da qualidade do código de teste (8) juntamente com a identificação de *test smells* (7) e suas refatorações (6) foram as mais citadas pelos estudantes no questionário. Portanto, compreende-se que os estudantes obtiveram habilidades positivas quanto ao aprendizado de refatoração de *test smells*.

Descoberta 3: O TERTS é uma ferramenta que promove habilidades fundamentais aos estudantes quanto a compreensão da qualidade de um código de teste unitário.

Tabela 2. Habilidades adquiridas pelos estudantes durante/após o uso do TERTS

Habilidade	Descrição	Total
Compreensão do código de teste	Compreensão aprimorada do código de teste, facilitando a identificação dos *test smells* abordados no TERTS	13
Conhecimento da qualidade do código de teste	Conhecimentos sobre a melhoria geral da qualidade do código de teste	8
Identificação de *test smells*	Melhor capacidade de identificar os *test smells* abordados pelo TERTS	7
Domínio de técnicas de refatoração de teste	Habilidade para aplicar técnicas de refatoração eficazmente nos *test smells* abordados no TERTS.	6
Crescimento profissional	Desenvolvimento pessoal e profissional na área de refatoração dos *test smells* abordados pelo TERTS	4

5.4. Nível de conhecimento dos estudantes sobre refatoração de *test smells* antes / após o uso do TERTS

Para avaliar o conhecimento e a confiança dos alunos na identificação e refatoração de *test smells*, foram incluídas questões de autoavaliação antes e depois da utilização do TERTS. A Tabela 3 apresenta a distribuição das respostas dos estudantes em relação ao nível de conhecimento. Ao analisar os percentuais de resposta fornecidos na Tabela 3, é evidente um aumento expressivo no número de estudantes que se autoavaliaram nos níveis intermediário e avançado de conhecimento após a utilização do TERTS. Alguns estudantes que já tinham conhecimentos sobre *test smells* conseguiram aprofundar ainda mais o aprendizado e ter confiança na realização da refatoração de testes.

Descoberta 4: O TERTS desempenha um papel significativo na consolidação dos conhecimentos dos alunos sobre identificação e refatoração de *test smells*.

Tabela 3. Nível de conhecimento dos estudantes antes e após o uso do TERTS

Nível de Conhecimento	Descrição	% Antes do TERTS	% Após o TERTS
Nenhum conhecimento	Não tenho conhecimentos prévios sobre *test smells*	54,5%	0%
Conhecimento básico	Possuo conhecimentos fundamentais sobre *test smells*, mas com limitações	36,4%	37,5%
Conhecimento intermediário	Consigo abordar e identificar alguns *test smells*, mas preciso de prática	9,1%	50%
Conhecimento avançado	Consigo identificar *test smells* com facilidade e refatorá-los com confiança	0%	12,5%

Na Tabela 3, destaca-se que mais da metade dos estudantes se autoavaliaram como possuindo nenhum conhecimento sobre o assunto antes da utilização do TERTS. Após o

uso da ferramenta, é notável que todos os estudantes, inclusive aqueles que inicialmente tinham nenhum conhecimento, conseguiram alcançar, pelo menos, o nível básico de compreensão do tema.

Descoberta 5: O TERTS proporciona uma base mínima de entendimento sobre o assunto, independentemente do nível inicial de familiaridade com a refatoração de *test smells*.

6. Limitações do estudo

Como limitações do estudo, destacam-se: (i) a avaliação positiva de um pequeno grupo de estudantes pode não refletir adequadamente a diversidade de habilidades e experiências de alunos de outras instituições ou contextos educacionais; (ii) a avaliação do nível de conhecimento adquirido pelos estudantes foi realizada a curto prazo, sem garantia da retenção desse aprendizado a longo prazo; (iii) o TERTS aborda apenas quatro tipos de *test smells*, portanto, os resultados deste estudo não abrangem outros tipos de *test smells*; e (iv) o TERTS avalia a refatoração dos *test smells* de forma estática, conforme as especificações do JNose Test, sem utilizar nenhuma outra ferramenta como base.

7. Conclusão

Este trabalho apresentou o TERTS, uma ferramenta de tutoria sobre identificação e refatoração de *test smells*. A metodologia consistiu no desenvolvimento do tutor e sua validação com estudantes de tecnologia da Universidade Federal do Ceará (UFC) - Campus Quixadá. Para este artigo, foram analisados os aspectos de (i) aceitação e uso da ferramenta, (ii) habilidades adquiridas durante/após o uso da ferramenta e (iii) nível de conhecimento sobre identificação e refatoração de *test smells* antes/após o uso do TERTS.

Os principais achados durante a coleta de resultados foram: (i) o TERTS é uma ferramenta que foi bem aceita pelos estudantes; (ii) o tutor fornece aos alunos habilidades de compreensão de um código de teste de unidade, identificação e refatoração dos *test smells* que são abordados nele; (iii) o TERTS pode ser utilizado tanto para aprimoramento do aprendizado, quanto para obter uma base mínima de conhecimento. De acordo com os resultados obtidos, infere-se que o TERTS é uma ferramenta qualificada e eficaz no ensino e prática de refatoração dos *test smells* abordados. Como trabalhos futuros, a intenção é continuar investindo na manutenção e evolução da ferramenta, buscando consolidá-la como uma referência na comunidade estudantil de tecnologia.

Referências

Aljedaani, W., Mkaouer, M. W., Peruma, A., and Ludi, S. (2023). Do the test smells assertion roulette and eager test impact students' troubleshooting and debugging capabilities? In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 29–39.

Bai, G. R., Smith, J., and Stolee, K. T. (2021). How students unit test: Perceptions, practices, and pitfalls. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 248–254.

Bavota, G., Qusef, A., Oliveto, R., De Lucia, A., and Binkley, D. (2012). An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *2012 28th IEEE international conference on software maintenance (ICSM)*, pages 56–65. IEEE.

Damasceno, H. and Bezerra, C. (2023). Ensino da prática de refatoração de test smells. In *Anais do XXXI Workshop sobre Educação em Computação*, pages 293–304. SBC.

Davis, F. D., Bagozzi, R. P., and Warshaw, P. R. (1989). User acceptance of computer technology: A comparison of two theoretical models. *Management science*, 35(8).

Deursen, A., Moonen, L., Bergh, A., and Kok, G. (2001). Refactoring test code.

Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., and Guimarães, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability*, 68(3):1189–1212.

Garousi, V., Rainer, A., Lauvås Jr, P., and Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, 165:110570.

Haendler, T., Neumann, G., and Smirnov, F. (2020). Refactutor: an interactive tutoring system for software refactoring. In *Computer Supported Education: 11th International Conference, CSEDU 2019, Heraklion, Crete, Greece, May 2-4, 2019, Revised Selected Papers 11*, pages 236–261. Springer.

Hofbauer, M., Bachhuber, C., Kuhn, C., and Steinbach, E. (2022). Teaching software engineering as programming over time. In *2022 IEEE/ACM 4th International Workshop on Software Engineering Education for the Next Generation (SEENG)*, pages 51–58. IEEE.

Ivers, J., Nord, R. L., Ozkaya, I., Seifried, C., Timperley, C. S., and Kessentini, M. (2022). Industry's cry for tools that support large-scale refactoring. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 163–164.

Kaur, S., Awasthi, L. K., and Sangal, A. (2021). A brief review on multi-objective software refactoring and a new method for its recommendation. *Archives of Computational Methods in Engineering*, 28:3087–3111.

Keuning, H., Heeren, B., and Jeuring, J. (2021). A tutoring system to learn code refactoring. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 562–568.

Kim, D. J. (2020). An empirical study on the evolution of test smell. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, pages 149–151.

Kim, D. J., Chen, T.-H., and Yang, J. (2021). The secret life of test smells-an empirical study on test smell evolution and maintenance. *Empirical Software Engineering*, 26.

Kumar, D. and Mishra, K. K. (2016). The impacts of test automation on software's cost, quality and time to market. *Procedia Computer Science*, 79:8–15.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

Palomba, F., Zaidman, A., and De Lucia, A. (2018). Automatic test smell detection using information retrieval techniques. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 311–322. IEEE.

Peruma, A., Almalki, K. S., Newman, C. D., Mkaouer, M. W., Ouni, A., and Palomba, F. (2019). On the distribution of test smells in open source android applications: An exploratory study.

Prokic, S., Grujić, K.-G., Luburić, N., Slivka, J., Kovacevic, A., Vidaković, D., and Sladic, G. (2021). Clean code and design educational tool. pages 1601–1606.

Santana, R., Martins, L., Rocha, L., Virgínio, T., Cruz, A., Costa, H., and Machado, I. (2020). Raide: a tool for assertion roulette and duplicate assert identification and refactoring. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, pages 374–379.

Smith, S., Stoecklin, S., and Serino, C. (2006). An innovative approach to teaching refactoring. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 349–353.

Spadini, D., Palomba, F., Zaidman, A., Bruntink, M., and Bacchelli, A. (2018). On the relation of test smells to software code quality. In *2018 IEEE international conference on software maintenance and evolution (ICSME)*, pages 1–12. IEEE.

Sun, B., Shao, Y., and Chen, C. (2019). Study on the automated unit testing solution on the linux platform. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 358–361. IEEE.

Van Deursen, A., Moonen, L., Van Den Bergh, A., and Kok, G. (2001). Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, pages 92–95. Citeseer.

Virgínio, T., Martins, L., Rocha, L., Santana, R., Cruz, A., Costa, H., and Machado, I. (2020). Jnose: Java test smell detector. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, page 564–569, New York, NY, USA. Association for Computing Machinery.

Winters, T., Manshreck, T., and Wright, H. (2020). *Software engineering at google: Lessons learned from programming over time*. O'Reilly Media.