UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII - PATOS
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE GRADUAÇÃO EM BACHARELADO EM CIÊNCIA COMPUTAÇÃO

NOBERTO NUNES DO NASCIMENTO JUNIOR

ANÁLISE DA ACEITAÇÃO E UTILIZAÇÃO DO DESENVOLVIMENTO ORIENTADO POR TESTES (TDD) EM PROJETOS ACADÊMICOS: UM ESTUDO DE CASO COM ALUNOS DE CIÊNCIA DA COMPUTAÇÃO DA UEPB CAMPUS VII

PATOS - PB
2025

NOBERTO NUNES DO NASCIMENTO JUNIOR

ANÁLISE DA ACEITAÇÃO E UTILIZAÇÃO DO DESENVOLVIMENTO ORIENTADO POR TESTES (TDD) EM PROJETOS ACADÊMICOS: UM ESTUDO DE CASO COM ALUNOS DE CIÊNCIA DA COMPUTAÇÃO DA UEPB CAMPUS VII

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e Sociais Aplicadas da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Ciência da Computação.

Orientador: Vinicius Augustus Alves Gomes

PATOS - PB

2025

N244a Nascimento Júnior, Noberto Nunes do.

Análise da aceitação e utilização do desenvolvimento orientado por testes (TDD) em projetos acadêmicos [manuscrito] : um estudo de caso com alunos de ciência da computação da UEPB campus VII / Noberto Nunes do Nascimento Júnior. - 2025.

40 f. : il. color.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Ciência da computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2025.

"Orientação : Prof. Grad. Vinícius Augustus Alves Gomes, Coordenação do Curso de Computação - CCEA".

1. Test-Driven Development. 2. TDD. 3. Testes de Software. 4. Engenharia de Software. I. Título

21. ed. CDD 005.14

Elaborada por Kaliane Eveny Martins de Oliveira - CRB - 15/986 BSC7

NOBERTO NUNES DO NASCIMENTO JÚNIOR

ANÁLISE DA ACEITAÇÃO E UTILIZAÇÃO DO DESENVOLVIMENTO ORIENTADO POR TESTES (TDD) EM PROJETOS ACADÊMICOS: UM ESTUDO DE CASO COM ALUNOS DE CIÊNCIA DA COMPUTAÇÃO DA UEPB CAMPUS VII

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação

Aprovada em: 02/06/2025.

BANCA EXAMINADORA

Documento assinado eletronicamente por:

- **Jose Jandilson de Sousa Arruda** (***.131.684-**), em **11/06/2025 14:59:49** com chave **de505cc646ed11f091092618257239a1**.
- **Vinícius Augustus Alves Gomes** (***.754.334-**), em **11/06/2025 14:23:01** com chave **ba33891c46e811f083551a7cc27eb1f9**.
- **Pablo Ribeiro Suárez** (***.806.354-**), em **13/06/2025 12:10:43** com chave **93523dda486811f0ad391a7cc27eb1f9**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Folha de Aprovação do Projeto Final
Data da Emissão: 18/06/2025
Código de Autenticação: 83e511

Dedico este trabalho aos meus pais, irmãs, amigos, professores e todos que me apoiaram nesta jornada. Graças a vocês que esse trabalho foi possivel.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus, por me conceder força, sabedoria e saúde para superar os desafios ao longo desta caminhada. Sem a sua presença, nada disso seria possível.

Aos meus pais e minhas irmãs, que me apoiaram e me incentivaram por todo esse caminho. Vocês são minha base e minha força para seguir em frente.

Aos meus professores e ao meu orientador, por todo o aprendizado que me transmitiram e pela orientação ao longo deste trabalho. Sua ajuda foi de grande importância para o meu crescimento acadêmico e profissional.

Aos meus amigos e colegas, que me acompanharam durante toda essa trajetoria universitaria, compartilhando dificuldades e aprendizados durante essa jornada.

“O desenvolvimento de software é um jogo de percepção,
e a percepção vem para a mente preparada, descansada e relaxada."
Kent Beck

RESUMO

A crescente complexidade no desenvolvimento de *software* destaca a importância de práticas que garantam a qualidade e confiabilidade dos sistemas. O *Test-Driven Development* (TDD) é uma abordagem ágil que visa reduzir defeitos e melhorar a qualidade, criando testes automatizados antes da codificação. Apesar dos benefícios reconhecidos do TDD, sua adoção é influenciada por fatores como a formação acadêmica. Pesquisas apontam que a aceitação do TDD aumenta com o nível acadêmico, sugerindo que a exposição a essa prática durante a graduação é essencial. Na Universidade Estadual da Paraíba (UEPB), Campus VII, não há disciplina específica sobre testes de *software*, o que impacta o conhecimento e aplicação do TDD entre os estudantes de Ciência da Computação. Este trabalho investigou o nível de conhecimento, aceitação e utilização do TDD entre alunos a partir do 6º período. A pesquisa, realizada por meio de questionário online, identificou que a maioria dos estudantes tem contato limitado com testes de *software* e há pouca exigência para a aplicação de TDD em atividades práticas. Embora os alunos reconheçam sua importância, muitos não se sentem preparados para utilizá-lo. Os resultados apontam para uma lacuna entre o interesse dos alunos e a preparação oferecida pela instituição, sugerindo a necessidade de abordagens didáticas mais eficazes.

Palavras-chave:Test-Driven Development; TDD; Testes de Software e Engenharia de Software .

ABSTRACT

The increasing complexity of software development highlights the importance of practices that ensure the quality and reliability of systems. Test-Driven Development (TDD) is an agile approach that aims to reduce defects and improve quality by creating automated tests before confirmation. Despite the recognized benefits of TDD, its adoption is influenced by factors such as academic background. Research indicates that the accessibility of TDD increases with academic level, indicating that exposure to this practice during undergraduate studies is essential. At the State University of Paraíba (UEPB), Campus VII, there is no specific course on software testing, which impacts the knowledge and application of TDD among Computer Science students. This study investigated the level of knowledge, ease and use of TDD among students from the 6th period onwards. The research, conducted through an online questionnaire, found that most students have limited contact with software testing and there is little requirement for the application of TDD in practical activities. Although students recognize its importance, many do not feel prepared to use it. The results point to a gap between students' interest and the preparation offered by the institution, indicating the need for more effective teaching approaches.

Keywords*: Test-Driven Development; TDD; Software Testing and Software Engineering*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASD	*Adaptive Software Development*
DSDM	*Dynamic System Development Method*
ES	Engenharia De *Software*
FDD	*Feature Driven Development*
PO	*Product Owner*
SM	*Scrum Master*
TDD	*Test Driven Development*
TLD	*Test Last Development*
UEPB	Universidade Estadual Da Paraíba
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização do problema

Teste de *software* é a execução de um produto para realizar a verificação do seu funcionamento no ambiente para o qual ele foi projetado e se alcança as especificações inicialmente planejadas. O principal objetivo dos testes é detectar possíveis falhas no projeto, para assim descobrir o que pode estar causando esses erros e então consertá-los antes da entrega do produto (Silva *et al.*, 2016)

Assim, a falta de testes em um *software* pode acarretar em um impacto negativo em sua qualidade e segurança, a ausência deles dificulta encontrar o que pode estar causando algum erro no sistema e realizar sua correção. Com isso é preferível ter uma maior garantia que o programa vai ser devidamente testado, uma das práticas que podem garantir o desenvolvimento de testes no *software* é o *Test Driven Development* (TDD).

Segundo Anwer *et al.* (2017), TDD é um modelo de desenvolvimento ágil, como em outros modelos, ele constroi o *software* em pequenas interações que exigem criar um caso de teste para em seguida desenvolver um código que pode passar nesse teste, esse código posteriormente vai ser refatorado visando melhorias. Desta forma, TDD acaba sendo diferente do modelo tradicional de desenvolvimento, onde o design e código é desenvolvido antes do teste, mas nesse caso o teste é realizado antes da codificação. A intenção por trás disso é reduzir a quantidade de defeitos e melhorar a qualidade do *software*.

No estudo realizado por SOUSA (2021), ele utilizou um *survey* para coletar opiniões sobre o TDD. O público alvo eram desenvolvedores que estivessem atuando, esses possuíam diferentes níveis profissionais e acadêmicos. Analisando os níveis acadêmicos ele constatou que o índice de utilização e aceite do TDD aumentava de acordo com o nível acadêmico do participante, a maioria dos respondentes da pesquisa estavam na graduação ou recém-formados, com isso ele associa a baixa aceitação do TDD constatado em sua pesquisa ao nível de graduação dos participantes.

No contexto atual da Universidade Estadual Da Paraíba (UEPB) campus VII não existe uma disciplina cujo o principal intuito é ensinar sobre testes de *software* (Universidade Estadual da Paraíba, 2016). Por mais que existam disciplinas em que teste de *software* é brevemente explanado como um pequeno tópico em algumas aulas, não há uma com foco no aprofundamento do tema. Por consequência, com testes de *software* sendo pouco explorados na universidade, TDD se torna algo ainda menos difundido, o que acarreta em dificuldade dos alunos em utilizá-lo na prática, levando em conta sua baixa experiência.

Utilizar o TDD pode acarretar em benefícios para os estudantes. Expor eles aos conceitos básicos e, em seguida, configurar tarefas de programação de modo que se espere que os alunos pratiquem TDD em suas tarefas desde o início. O TDD oferece benefícios tangíveis para os estudantes, ele é aplicável em pequenos projetos após um certo treinamento, torna o estudante muito confiante na correção do seu código, os incentiva a sempre manterem uma versão funcional

do que já foi concluído, e ainda os estimula a testarem as funcionalidades e o código à medida que são implementados. Isso evita problemas de integração em *"big bang"* que os alunos frequentemente enfrentam quando tentam escrever todo o código de uma atividade e só após isso tentam executar, testar e depurar (Edwards, 2003).

Esses problemas de integração *"big bang"* acontecem quando todos os componentes de um *software* são combinados e testados ao mesmo tempo, geralmente no final do projeto. Essa abordagem pode levar a dificuldades, pois se um erro aparecer, pode ser complicado descobrir onde está o problema. Além disso, resolver as falhas pode demorar mais, pois os desenvolvedores precisam revisar todo o código de uma vez. Em contraste, abordagens como o TDD incentivam testar e integrar partes do *software* gradualmente, facilitando a identificação e correção de problemas.

Se o TDD possui benefícios até mesmo no aprendizado dos alunos, qual seria a visão deles sobre o assunto? Pensando nisso, este trabalho tem como objetivo realizar uma pesquisa para averiguar o grau de aceitação e conhecimento sobre TDD no curso de Ciência da Computação na Universidade Estadual Da Paraíba, Campus VII, Patos-PB.

1.2 Objetivos

O principal objetivo deste trabalho é investigar o grau de aceitação e utilização de TDD (*Test Driven Development*) em projetos realizados pelos alunos de Ciência da Computação no Campus VII da UEPB. A fim de alcançar o objetivo geral deste trabalho, foram definidos os seguintes objetivos específicos:

- Selecionar amostra de alunos do curso de ciência da computação da uepb campus VII com base nos critérios estabelecidos para participação da pesquisa;
- Elaborar questionário para investigação e mensuração do nível de conhecimento, utilização e aceitação dos alunos em relação ao TDD;
- Aplicar o questionário no grupo de alunos selecionados para obtenção de dados para estudo;
- Realizar a análise dos dados coletados através do estudo;
- Basear-se nos dados coletados e analisados para chegar a uma conclusão sobre o índice de utilização e aceitação de TDD entre os estudantes do campus VII da UEPB em Patos-PB.

1.3 Questões de pesquisa

Para complementar a análise deste trabalho, foram levantadas as seguintes questões de pesquisa.

- QP1. Os alunos do campus VII da UEPB foram expostos às práticas de desenvolvimento de testes de *software*?
- QP2. Houve preparação dos estudantes para o uso do Desenvolvimento Orientado a Testes?
- QP3. Existe cobrança dos professores aos estudantes para praticarem testes de *software* em seus trabalhos acadêmicos?
- QP4. Foi demonstrado interesse no aprendizado e prática quanto ao desenvolvimento de testes de *software* e práticas como o TDD?

1.4 Estrutura do trabalho

Este trabalho possui cinco capítulos específicos estruturados da seguinte forma: no primeiro capítulo é apresentado uma visão geral do trabalho através da contextualização do problema, objetivo geral e especificos, e as questões de pesquisa; no segundo capítulo apresenta-se o referencial teórico com os principais tópicos necessários para a compreensão do trabalho; no terceiro capítulo é apresentada a metodologia utilizada nessa pesquisa; no quarto capítulo é realizada a análise e discussão dos resultados obtidos; no quinto capítulo é apresentada as considerações finais e sugestões para trabalhos futuros; e ao final, encontra-se as referências utilizadas.

Considerando o contexto apresentado, bem como os objetivos e questões de pesquisa que norteiam este trabalho, faz-se necessário aprofundar a compreensão teórica sobre o tema. Na próxima seção, são apresentados os principais conceitos que servirão de base para o desenvolvimento do estudo.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão abordados os tópicos essenciais para a compreensão deste trabalho, todos fundamentados no referencial teórico. Esta seção é vital para estabelecer as bases conceituais da pesquisa, oferecendo uma visão abrangente ve detalhada das teorias e conceitos pertinentes.

2.1 Métodos ágeis

As metodologias ágeis para criação de *software* foram criadas como uma alternativa às metodologias tradicionais de desenvolvimento, conhecidas como pesadas. Mesmo com os avanços dos computadores, técnicas e ferramentas nos últimos anos, a criação de *software* confiável, preciso e entregue dentro de prazos e custos estabelecidos permanece sendo um grande desafio (Soares, 2004).

O termo *agile* foi cunhado em 2001, quando dezessete desenvolvedores de *software* reconhecidos se reuniram em *Utah*, Estados Unidos, para buscar formas novas e aprimoradas para desenvolvimento de *software*. Eles dividiram suas experiências e perceberam algumas práticas comuns de engenharia de *software*, úteis no desenvolvimento de programas dentro de um prazo pré-definido (Fowler; Highsmith *et al.*, 2001).

Nessa reunião de 2001, autores e representantes de várias metodologias ágeis, como *eXtreme Programming* (XP), *Scrum*, *Dynamic System Development Method* (DSDM), *Adaptive Software Development* (ASD), *Crystal, Feature Driven Development*(FDD) e *Lean Development*, se reuniram para discutir os padrões e práticas comuns entre as abordagens de desenvolvimento ágil de *software*, procurando identificar elementos similares entre essas diferentes técnicas (Pereira; Torreão; Marçal, 2007)

Essas metodologias ágeis de desenvolvimento de *software* fornecem uma forma de desenvolvimento mais eficiente e leve que constrói um programa de forma iterativa e incremental. A intenção por trás dos modelos ágeis de desenvolvimento de *software* era descobrir formas novas e mais eficientes de desenvolvimento de código que pudessem superar as limitações dos modelos tradicionais de construção de sistema (Anwer *et al.*, 2017). Neste tópico serão abordadas algumas dessas metodologias de desenvolvimento ágeis.

2.1.1 Scrum: um framework ágil

O *Scrum* é um *framework* leve que ajuda pessoas, equipes e organizações a gerar valor por meio de soluções adaptáveis para problemas complexos. ele possui alguns papéis em sua implementação, como o *Scrum Master*(SM), o *Product Owner* (PO) e o time *Scrum* (Schwaber; Sutherland, 2011).

Esse modelo ágil possui uma abordagem empírica, e aplica princípios da teoria de controle de processos industriais a criação de *software*, e tem como objetivo reintroduzir conceitos como

flexibilidade, adaptabilidade e produtividade. Essa metodologia tem como intuito estabelecer uma forma de trabalho em equipe que possibilite o desenvolvimento de *software* de forma ágil e eficiente, mesmo em um ambiente em constante mudanças (Soares, 2004).

De acordo com Schwaber e Sutherland (2011), a unidade fundamental do *Scrum* é uma pequena equipe de pessoas, denominada de equipe, ou time *Scrum*, que é composta por um *Scrum Master*, um *Product Owner* e desenvolvedores, eles são multifuncionais e autogerenciados.

O *Scrum Master* é o responsável por estabelecer a metodologia ágil como foi definido no guia do *Scrum*. Responsável por ajudar no entendimento da teoria e a prática do modelo ágil para equipe e a organização. Também é responsável pela eficácia da equipe de projeto, facilitando a melhoria das práticas da equipe dentro do *framework* ágil.

Por sua vez, o *Product Owner* é responsável por maximizar o valor do produto resultante do trabalho da equipe de projeto, também é responsabilidade dele a gestão eficaz do *Product Backlog*, toda a organização deve respeitar suas decisões, essas decisões são visíveis no conteúdo e na ordenação do *Product Backlog* e através do Incremento que pode ser inspecionado na Revisão do *Sprint*.

Quanto aos desenvolvedores, são as pessoas na equipe *Scrum* comprometidas a criação de qualquer aspecto de um Incremento utilizável a cada *sprint*. Eles são responsáveis por criar um plano para a iteração, o *sprint backlog*, incorporar qualidade ao aderir uma definição de pronto as implementações, e adaptar diariamente o plano em direção ao objetivo da iteração.O modelo ágil também possui diversos eventos e artefatos que o caracterizam.

Por fim, o ciclo do *Scrum* é composto por *sprints*, com duração de 2 a 4 semanas. Predecente a cada ciclo de desenvolvimento, é realizada uma Reunião de Planejamento *(Sprint Planning)*, onde o time e o *Product Owner* priorizam e estimam as tarefas que serão realizadas na iteração. Durante a execução, há reuniões diárias com duração máxima de 15 minutos para monitorar o progresso. Ao final, ocorre a Reunião de Revisão *(Sprint Review)* para apresentar o produto e verificar se os objetivos foram atingidos, seguida pela Reunião de Retrospectiva *(Sprint Retrospective)*, que visa melhorar o processo e o time para os próximos ciclos de desenvolvimento (Pereira; Torreão; Marçal, 2007).

Uma das partes mais essenciais do *Scrum* é o *Product Backlog*, nele contém uma lista de itens priorizados que possuem tudo o que é necessário para a conclusão do projeto, podendo ser requisitos funcionais ou não. Cada item no *Backlog* do produto deve estar associado a um valor de negócio, o que possibilita medir o retorno do projeto e priorizar a execução dos itens com base nesse valor (Pereira; Torreão; Marçal, 2007).

Outro item importante é o *Sprint Backlog*, composto pelo conjunto de itens do *Product Backlog* selecionados para a iteração, bem como um plano prático para entregar o Incremento. O *Sprint Backlog* é um plano feito por e para os desenvolvedores, é uma imagem altamente visível e em tempo real do trabalho que os desenvolvedores planejam realizar durante o ciclo de desenvolvimento para atingir a meta da *Sprint* (Schwaber; Sutherland, 2011).

2.1.2 Extreme Programming: uma abordagem ágil

O XP é uma metodologia leve que tem ganhado muita aceitação e popularidade, ela promove uma maneira de desenvolvimento de *software* baseada nos princípios de simplicidade, comunicação, *feedback* e coragem. Projetada para ser utilizada por equipes pequenas que precisam desenvolver *software* rapidamente em um ambiente de requisitos em constante mudança (Beck, 1999 *apud* Kircher *et.al.*, 2001).

No XP, são necessárias medidas mínimas de acompanhamento, o que significa que a criação de documentação e requisitos do projeto não é exigida, é muito orientada para o trabalho em equipe, a responsabilidade de concluir o projeto é conjunta de toda a equipe de desenvolvimento, não apenas do responsável pela equipe. O XP é mais indicado para equipes pequenas, geralmente de 12 a 14 membros, também promove o envolvimento do usuário e do cliente em uma fase inicial do projeto, assim economizando tempo perdido por falhas de comunicação (Shrivastava *et al.*, 2021).

Segundo Beck (2008) algumas práticas importantes do XP são:

- **Jogo do Planejamento**: Definir rapidamente o escopo da próxima versão, combinando prioridades de negócio com estimativas técnicas. Caso a realidade não corresponda ao planejamento, ajustar de acordo;
- **Entregas Frequentes**: Colocar um sistema simples em produção rapidamente e lançar novas versões em ciclos curtos;
- **Metáfora**: Orientar o desenvolvimento com uma história simples e compartilhada por todos, que explique como o sistema funciona de forma geral;
- **Projeto Simples**: O sistema precisa ser projetado da maneira mais simples possível em cada momento. Qualquer complexidade desnecessária deve ser removida assim que for identificada;
- **Testes**: Os programadores escrevem testes de unidade continuamente, que devem ser executados sem falhas para que o desenvolvimento continue. Os clientes escrevem testes que demonstram quando as funcionalidades estão completas;
- **Refatoração**: Os programadores reestruturam o sistema sem alterar seu comportamento, a fim de eliminar duplicidades, melhorar a comunicação, simplificar e aumentar a flexibilidade;
- **Programação em Pares**: Todo código de produção é desenvolvido por dois programadores utilizando o mesmo computador;
- **Propriedade Coletiva**: Qualquer membro da equipe pode modificar qualquer parte do código, em qualquer momento;

- **Integração Contínua**: Integre e atualize as versões do sistema diversas vezes ao dia, sempre que uma tarefa for concluída;
- **Semana de 40 Horas**: Como regra geral, trabalhar no máximo 40 horas por semana, evitando horas extras por mais de duas semanas consecutivas;
- **Cliente Presente**: Um cliente real deve estar disponível na equipe o tempo todo para esclarecer dúvidas e fornecer *feedback*;
- **Padrões de Codificação**: Os programadores devem escrever o código seguindo regras que priorizem a comunicação clara através do próprio código.

De acordo com Shrivastava *et al.* (2021) uma das várias características do XP é que, os clientes devem testar continuamente o produto com vários casos de teste para garantir que ele esteja funcionando corretamente e comunicar as falhas. Antes mesmo de escrever o código, os desenvolvedores escrevem os casos de teste, também podem usar drivers de teste depois de escrever o código para verificar se tudo está funcionando perfeitamente. Este é justamente o pilar do funcionamento do TDD.

2.2 Testes de software

Falhas de *software* são responsáveis por um grande aumento nos custos e tempo de desenvolvimento. Mesmo que seja impossível remover todos os erros, ainda é possível reduzir consideravelmente a sua quantidade utilizando uma infraestrutura de testes mais elaborada, que possibilite identificar e remover defeitos mais cedo e de forma mais eficaz (Borges, 2006).

Teste é o processo de executar um programa com a intenção de encontrar erros (Myers; Sandler; Badgett, 2011). Teste é essencial para a Garantia de Qualidade de *Software (Software Quality Assurance)*, mas é uma atividade crítica e dinâmica, pois defeitos podem ser inseridos durante todo o processo de desenvolvimento e quanto mais demorar para serem identificados, maiores serão os custos para resolvê-los e contornar seus efeitos (Pinto, 2019).

Teste de *software* é um termo amplo que abrange muitos tipos diferentes de atividades, desde testar um pequeno pedaço de código pelo desenvolvedor (teste unitário), até a validação do cliente em um sistema (testes de aceitação), ao monitoramento em tempo de execução de um sistema centrado na rede aplicada orientada a serviços. Os casos de testes podem ser criados visando diferentes objetivos, como por exemplo expor desvios dos requisitos do usuário, ou avaliar a conformidade com uma especificação padrão, ou então avaliar a resistência a condições de carga estressantes ou ataques maliciosos, ou medir determinados atributos, como desempenho e usabilidade, ou estimando a confiabilidade operacional, e assim por diante. Além disso, os testes podem ser realizados de acordo com um procedimento formal controlado, exigindo planejamento e documentação rigorosos, ou de uma maneira mais informal e *ad hoc* (testes exploratórios)(Bertolino, 2007).

Segundo Palma (2007), teste unitário ou de unidade é um processo que consiste na verificação da menor unidade do projeto de *software*, eles são considerados os primeiros de uma cadeia de testes à qual um *software* pode ser submetido. Nesta fase não se pretende testar toda a funcionalidade de uma aplicação, mas sim os diversos componentes de forma isolada.

Por outro lado, teste de integração é aplicado quando todos os módulos individuais são combinados para formar um programa funcional, ele é realizado no nível do módulo, ao invés de ser realizado no nível de instruções, como ocorre no teste unitário, teste de integração enfatiza as interações entre os módulos e suas interfaces (Leung; White, 1990).

Por sua vez, os testes de aceitação representam os interesses do cliente. Eles são os responsáveis por transmitir para o cliente a confiança de que o sistema possui todos os recursos necessários e que está funcionando corretamente. Em teoria, caso todos os testes de aceitação sejam validados, então o projeto está terminado (Miller; Collins, 2001).

Em adição, o Teste de regressão refere-se à reexecução de um teste que já foi realizado com sucesso em um conjunto de funcionalidades do sistema. Esse tipo de teste é utilizado após o sistema ter sofrido alterações, sua função é garantir que o programa continue funcionando perfeitamente de acordo com o especificado (Oliveira *et al.*, 2007).

Finalmente, o Teste exploratório é caracterizado como qualquer tipo de teste em que o desenvolvedor controla ativamente o design deles enquanto são executados, e então utiliza as informações que foram obtidas durante a execução para desenvolver novos e melhores testes (Bach, 2003).

2.3 Test Driven Development (TDD)

O *Test Last Development* (TLD) consiste na prática de criar testes apenas após a etapa de desenvolvimento de uma funcionalidade ou de um módulo de um sistema. Na maioria das vezes, para testar um *software*, são utilizadas ferramentas de apoio, como por exemplo as bibliotecas de testes automatizados. Uma vez criados, os testes poderão ser executados quantas vezes for preciso (Barros; Junior, 2019).

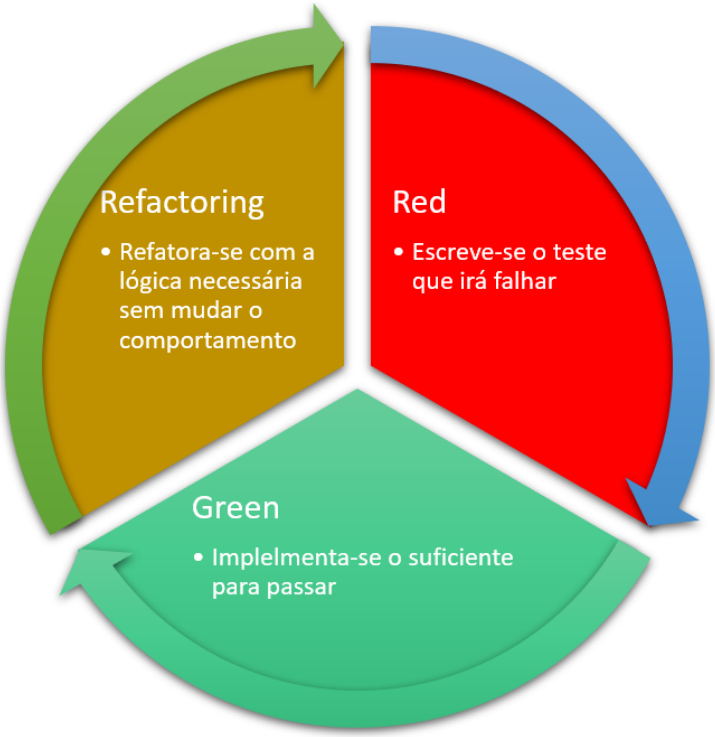
Após o surgimento das metodologias ágeis, tornou-se necessário aperfeiçoar as técnicas utilizadas para testar *software*. Não fazia mais sentido a prática de *test-last*, pois as entregas eram constantes. Houve então a utilização de uma abordagem de iterações para os testes também, não aguardando mais para testar apenas após a finalização do produto. Com isso, o TLD foi deixado de lado em favor de um estilo diferente conhecido como *test-first* (Fucci et al., 2016 *apud* Barros; Pereira Junior, 2019).

Uma técnica que utiliza uma abordagem *test-first* é o TDD, idealizada por Kent Beck, é um modelo de desenvolvimento de *software*, nela são seguidos uma série de passos, em que os testes são desenvolvidos antes do código de produção e com base nesses testes que é desenvolvida a aplicação (SOUSA, 2021). A abordagem de testes no TDD é pragmática, nele os testes servem como um meio para alcançar um objetivo, este que seria um código final no qual se possa ter plena confiança (Beck, 2010).

O TDD é um processo ágil de desenvolvimento de *software*. Como em outros modelos ágeis, ele primeiro constrói o programa em pequenas iterações, nas quais é preciso primeiro criar um teste, seguido por um pequeno trecho de código que consiga passar nesse teste, sendo o código posteriormente refatorado para melhorias. O TDD é o oposto da abordagem tradicional de desenvolvimento, que começa com o *design* e a codificação, para então fazer os testes. No caso do TDD, os testes são criados antes do código, a intenção do TDD é diminuir a taxa de defeitos e melhorar a qualidade do *software* (Anwer *et al.*, 2017).

A abordagem comum do TDD é conhecida como *Red-Green-Refactor*. O processo começa com a elaboração de um teste que, ao ser executado, falha ou não compila, esta parte é denominada *red*, logo após uma funcionalidade é implementada de forma bem simplificada e o mais rápido possível, apenas o suficiente para ser validado no teste escrito, esse seria o *green*, por fim, o código é refatorado e os testes são novamente executados até que o código alcance o nível desejado, nessa parte os próprios testes podem ser refatorados (Madeyski; Kawalerowicz, 2013). Isso pode ser visualizado através da figura 1.

Figura 1 – Ciclo Red-Green-Refactor



Fonte: (Araújo, 2023).

O embasamento teórico construído ao longo desta seção oferece o suporte necessário para a definição dos caminhos metodológicos seguidos na pesquisa. Na seção a seguir, são detalhados os procedimentos adotados, com base nas abordagens previamente discutidas e com foco em sua aplicação prática no contexto estudado.

3 METODOLOGIA

Neste capítulo, sera abordado a metodologia utilizada neste estudo, definindo os métodos de coleta de dados que serão utilizados, e fornecer uma melhor compreensão dos processos metodológicos que sustentam nossa análise e interpretação dos resultados.

3.1 Tipo de Pesquisa

A presente pesquisa, segundo Wazlawick (2021), é de caráter prímário, classificada como qualitativa, de natureza aplicada, com abordagem exploratória e descritiva. A parte quantitativa busca mensurar o grau de conhecimento, aceitação e uso do TDD entre os estudantes, a partir das respostas fechadas coletadas por meio de um questionário estruturado. Já a parte qualitativa está presente nas questões com alternativas abertas ou justificativas, permitindo uma melhor compreensão das percepções dos participantes sobre o tema abordado.

Quanto aos seus objetivos, trata-se de uma pesquisa exploratória, por buscar identificar e compreender aspectos ainda pouco investigados no contexto local, e descritiva, por procurar retratar as características e opiniões dos estudantes com base nos dados obtidos ao longo do estudo.

3.2 Coleta de Dados

A realização da coleta dos dados foi realizada por meio da aplicação de um questionário desenvolvido especificamente para este estudo, tendo sido aplicado aos estudantes do curso de Ciência da Computação da Universidade Estadual Da Paraíba (UEPB), Campus VII, com a finalidade de conseguir informações sobre seu grau de familiaridade com TDD, e experiências prévias caso possuam, assim como suas percepções em relação à utilização dessa metodologia de desenvolvimento de *software*.

O questionário foi composto por perguntas fechadas e semiabertas, possibilitando assim a coleta de respostas objetivas, e de dados qualitativos, com o objetivo de obter dados qualiquantitativos. As perguntas fechadas servirão para medir o conhecimento básico e o nível de aceitação do TDD, e serão utilizadas para geração de gráficos, já as perguntas semiabertas permitiram uma visão mais detalhada das percepções dos estudantes, possibilitando que expressem suas opiniões de maneira mais livre.

3.2.1 Estruturação do instrumento de coleta

O questionário aplicado foi desenvolvido com base nos objetivos da pesquisa, buscando explorar a aceitação e a utilização do Desenvolvimento Orientado por Testes no contexto acadêmico. Para isso, as questões foram elaboradas considerando diferentes aspectos da experiência dos alunos com TDD, desde o contato inicial com o tema até sua opinião crítica sobre a prática.

Visando uma análise mais estruturada e coerente dos dados, as questões foram agrupadas em quatro categorias principais, conforme descrito a seguir:

1. **Exposição e Aprendizado Formal**
   Investiga onde e como os alunos foram expostos a conteúdos relacionados ao TDD e testes de *software*, além da percepção sobre o suporte oferecido pela instituição para o aprendizado dessas práticas. A intenção é identificar se o conteúdo está presente no currículo e se a formação tem sido efetiva.

2. **Apropriação e Capacidade Técnica**
   Avalia o entendimento dos alunos sobre o TDD, sua capacidade de aplicar a técnica em projetos e o uso prático de testes. Busca identificar se os alunos não apenas conhecem o TDD, mas também se sentem aptos a aplicá-lo.

3. **Fatores Influenciadores da Adoção**
   Explora as barreiras percebidas e os fatores motivacionais relacionados à adoção do TDD. Busca compreender o que impede ou poderia incentivar os alunos a utilizarem essa prática.

4. **Atitude e Julgamento sobre o TDD**
   Reúne opiniões dos alunos sobre a utilidade, os benefícios e os possíveis malefícios do TDD, permitindo avaliar o nível de aceitação da técnica.

A seguir, a Tabela 1 apresenta a relação entre as questões do questionário e as categorias temáticas às quais pertencem.

Tabela 1 – Questões e categorias às quais pertencem

Nº	Questão	Categoria
1	A disciplina de Engenharia de *Software* me proveu conhecimentos sobre métodos ágeis	Exposição e Aprendizado Formal
2	A disciplina de Engenharia de *Software* me proveu conhecimentos sobre testes de *software*	Exposição e Aprendizado Formal
3	A disciplina de Engenharia de *Software* me promoveu conhecimento sobre *Test Driven Development* (TDD)	Exposição e Aprendizado Formal
4	Compreendo o que é e sei utilizar *Test Driven Development* (TDD)	Apropriação e Capacidade Técnica
5	Costumo realizar testes de *software* em meus projetos.	Apropriação e Capacidade Técnica
6	O que te impediria de utilizar testes automatizados em seus projetos?	Fatores Influenciadores da Adoção
7	Me sinto confiante para aplicar TDD em um projeto real de *software*.	Apropriação e Capacidade Técnica
8	Onde foi que o adquiriu conhecimento sobre TDD	Exposição e Aprendizado Formal
9	Eu acho que a faculdade oferece suporte suficiente para o aprendizado de TDD	Exposição e Aprendizado Formal
10	Se você nunca usou TDD, qual a principal razão?	Fatores Influenciadores da Adoção
11	O que te motivaria a aprender e usar TDD?	Fatores Influenciadores da Adoção
12	Acredito que o TDD é útil para desenvolvimento de *software*.	Atitude e Julgamento sobre o TDD
13	Eu acredito que o uso de TDD aumentaria a produtividade do desenvolvedor.	Atitude e Julgamento sobre o TDD
14	Em sua opinião, qual o principal benefício do TDD	Atitude e Julgamento sobre o TDD
15	Em sua opinião qual o principal malefício	Atitude e Julgamento sobre o TDD

Fonte: Elaborado pelo autor(2025).

3.3 Universo de Pesquisa

O universo de pesquisa utilizado foi composto por estudantes do curso de Ciência da Computação da UEPB, Campus VII, matriculados nos períodos tardios do curso, que têm maior probabilidade de terem sido expostos a práticas de desenvolvimento de *software* como o TDD. Sendo considerados participantes apenas aqueles que estivessem cursando a partir do 6º período, garantindo que possuam conhecimento suficiente para responder às questões.

A opção por selecionar estudantes do 6º período em diante também se deve ao fato de ser nesse período que eles cursam a disciplina de engenharia de *software*, uma das mais relevantes no quesito teste de *software*, assim possibilitando um maior conhecimento sobre o tema da pesquisa para os respondentes (Universidade Estadual da Paraíba, 2016).

3.4 Aplicação

O questionário foi disponibilizado de forma *online*, utilizando a plataforma do *Google Forms*, facilitando o acesso para garantir uma maior adesão por parte dos estudantes. A aplicação do questionário teve uma duração de duas semanas, período em que a pesquisa foi divulgada para os alunos participarem voluntariamente, sendo garantido a eles o anonimato e confidencialidade dos dados coletados, também tendo sido permitido apenas uma resposta por estudante, para garantir a acurácia da pesquisa.

3.5 Análise de Dados

Após a finalização do questionário houve a coleta dos dados para realização da análise, as perguntas fechadas foram analisadas estatisticamente, com a utilização de ferramentas como gráficos e tabelas para detectar padrões e tendências nas respostas, as perguntas semiabertas foram sujeitas à análise de conteúdo, com a intenção de categorizar e interpretar as opiniões e percepções dos estudantes sobre o TDD. Dessa forma, foi possível obter dados qualiquantitativos, integrando tanto aspectos quantitativos quanto qualitativos para uma visão abrangente.

Utilizando essa abordagem, foi esperado conseguir uma visão abrangente sobre o nível de conhecimento e aceitação do TDD pelos alunos da UEPB, Campus VII, e com isso identificar possíveis barreiras ou motivações para a adoção dessa prática.

3.6 Ameaças à Validade da Pesquisa

Embora esta pesquisa tenha sido cuidadosamente planejada e conduzida com o objetivo de garantir a confiabilidade dos dados, é importante reconhecer algumas possíveis ameaças à validade dos resultados obtidos.

Em primeiro lugar, apesar de o questionário ter sido direcionado exclusivamente a estudantes a partir do 6º período do curso de Ciência da Computação da UEPB, Campus VII, não há como garantir com total certeza que apenas alunos dentro desse perfil tenham respondido à pesquisa. Como a aplicação foi feita de forma *online* e aberta, existe a possibilidade de participação de estudantes que não atendiam aos critérios estabelecidos.

Outro fator importante diz respeito à disciplina de engenharia de *software*, que representa uma etapa importante para o contato dos alunos com práticas como o *Test Driven Development*. Durante determinado período letivo, o professor titular da disciplina esteve afastado, sendo substituído temporariamente por outro docente. Essa mudança pode ter impactado significativamente o conteúdo abordado e a forma como os temas foram trabalhados em sala de aula, o que pode ter gerado variações na experiência de aprendizado entre os alunos que cursaram a disciplina com diferentes professores.

Esses aspectos devem ser considerados na interpretação dos resultados, pois podem influenciar diretamente a percepção dos estudantes em relação ao tema pesquisado, especialmente

no que diz respeito ao nível de conhecimento e à aplicação prática do TDD.

Com a metodologia devidamente estruturada e aplicada, foi possível obter dados relevantes para a análise proposta neste estudo. A seguir, são apresentados os resultados obtidos a partir da coleta, bem como a discussão das principais observações identificadas com base nas respostas dos participantes.

4 RESULTADOS E DISCUSSÕES

Neste capítulo serão apresentados e discutidos os resultados desta pesquisa, visando entender a visão dos discentes de Ciência da Computação, do campus VII da UEPB, sobre a utilização e aceitação da técnica de desenvolvimento orientado a testes (TDD).

4.1 Obtenção de Conhecimento Sobre Métodos Ágeis.

Como *Test Driven Development* pode ser considerado um dos pilares de metodologias ágeis como a *Extreme Programming*, torna-se importante averiguar se os estudantes possuem conhecimentos relacionados a estes métodos. Em relação ao aprendizado de métodos ágeis na disciplina de Engenharia de *Software*, a maioria dos estudantes indicou ter adquirido conhecimentos durante a disciplina, com 65,5% posicionando-se de forma positiva quanto ao aprendizado, 23,5% mantendo-se neutros e 11% demonstrando uma percepção negativa sobre o aprendizado obtido.

Assim é possível notar um certo grau de aprendizagem das metodologias durante a matéria, isso pode ser considerado algo bom, pois como dito por (Pereira; Torreão; Marçal, 2007), entender os valores dos métodos ágeis acarreta em aperfeiçoamento em técnicas e processos de desenvolvimento e gestão de projetos.

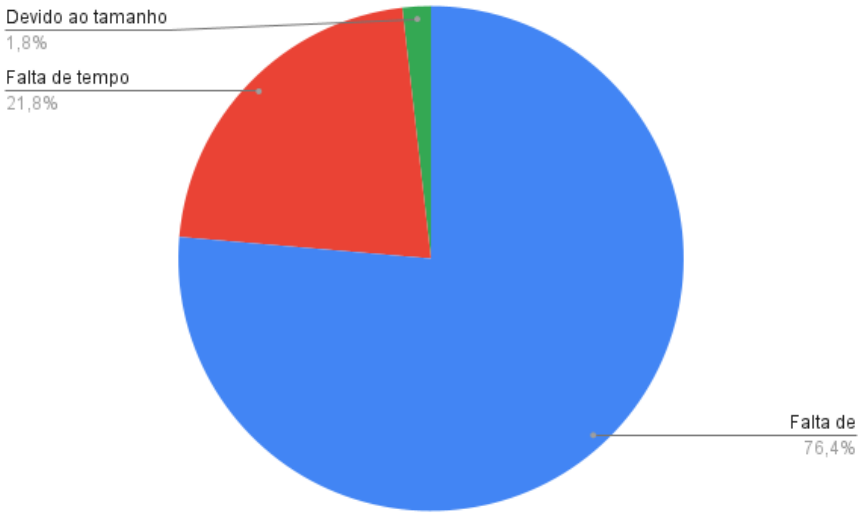
4.2 Obtenção e Aplicação de Conhecimento Sobre Testes de *Software*.

Em relação ao aprendizado de testes de *software* é possível perceber que os estudantes foram mais divergentes sobre terem ou não estudado, apenas 34.5% que afirmam terem explorado o tema durante a disciplina, outros 31.0% não acreditam terem aprendido sobre, e mais 34.5% ficaram neutros. Assim torna-se mais complexo averiguar qual foi a real situação do seu ensino durante a matéria.

Então já era esperado que a utilização de testes em seus projetos fosse ainda pior, considerando que poucos afirmaram terem aprendido sobre. Entre os entrevistados, apenas 21.9% declaram que utilizam a prática da realização de testes ao desenvolverem algo, 32.7% ficaram neutros sobre a sua situação, e entre eles 45.4% asseguram que não utilizam durante a realização de seus projetos.

Nesse contexto, os estudantes apontam que a principal razão pela qual não realizam testes de *software* em seus projetos é a falta de conhecimento técnico. Em segundo lugar, indicam a falta de tempo como um dos principais fatores que dificultam a realização dos testes, como pode ser observado no figura 2 abaixo:

Figura 2 – Motivos da não utilização de testes em projetos



Fonte: Elaborado pelo autor(2025).

Esta então se torna uma situação complexa, visto que segundo (Bertolino, 2007), a realização de testes é uma atividade essencial na engenharia de *software*, além de ser amplamente utilizado na indústria para garantia de qualidade, pois ao realizar a análise de um *software* em execução é possível obter um *feedback* realista do seu comportamento, e como tal, e se torna um complemento imprescindível juntamente de outras técnicas .

4.3 Aprendizado sobre *Test Driven Development.*

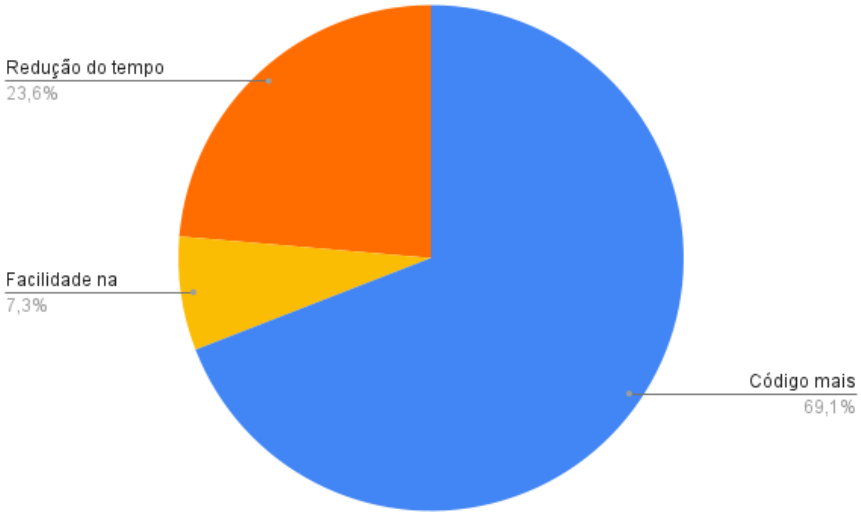
Entre os entrevistados, apenas 16,4% afirmaram ter estudado *Test-Driven Development* durante a disciplina de Engenharia de *Software*, outros 29,1% permaneceram neutros quanto a essa questão, enquanto 54,5% acreditam não ter tido contato com esse conteúdo na disciplina. Em relação ao domínio da técnica, apenas 7,3% se consideram capazes de compreender e aplicar o TDD em seus projetos, já 36,4% se mantiveram neutros quanto à própria compreensão, e 56,3% demonstraram não ter confiança suficiente em seus conhecimentos para utilizá-la na prática.

Além disso, no que diz respeito ao suporte oferecido pela faculdade para o aprendizado de TDD e outras técnicas de testes de *software*, apenas 14,5% dos entrevistados consideram que receberam apoio suficiente para desenvolver esses conhecimentos. Por outro lado, 70,9% relataram não ter tido suporte adequado nesse aspecto, o que pode ter contribuído para a baixa familiaridade e confiança dos alunos em aplicar tais práticas em seus projetos, já outros 14.5% decidiram ficar neutros sobre a assistência ofertada pela faculdade para a aprendizagem da técnica.

Embora a maioria dos entrevistados não possuam um amplo conhecimento sobre TDD, 87.3% acreditam que é uma técnica muito útil para o desenvolvimento de *software*, com apenas 3.6% não achando que seja benéfica. Entre os principais benefícios, eles apontam principalmente

o código mais confiável e com menos *bugs* e a redução do tempo de correção de erros. Isso pode ser visto na figura 3 abaixo:

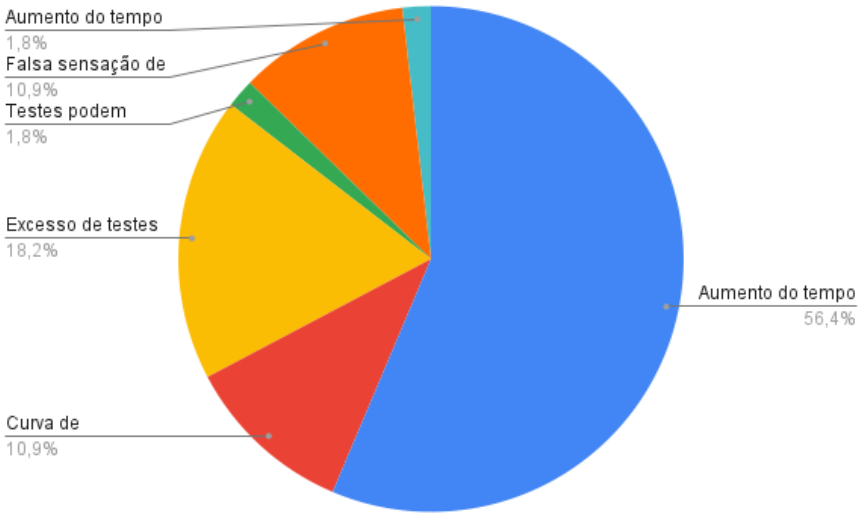
Figura 3 – Principal benefício do TDD



Fonte: Elaborado pelo autor(2025).

Já como malefício mencionam principalmente o aumento do tempo de desenvolvimento inicial e uma manutenção mais complexa devido ao excesso de testes, como visto na figura 4 abaixo:

Figura 4 – Principal malefício do TDD



Fonte: Elaborado pelo autor(2025).

Embora os dados mostram uma baixa familiaridade e confiança na aplicação do TDD por parte dos estudantes, a percepção de que a técnica é útil é amplamente compartilhada entre os entrevistados, isso reforça a importância de abordagens pedagógicas mais eficazes no ensino da prática. Edwards (2003) destaca que a exposição precoce dos alunos ao TDD, aliada à aplicação

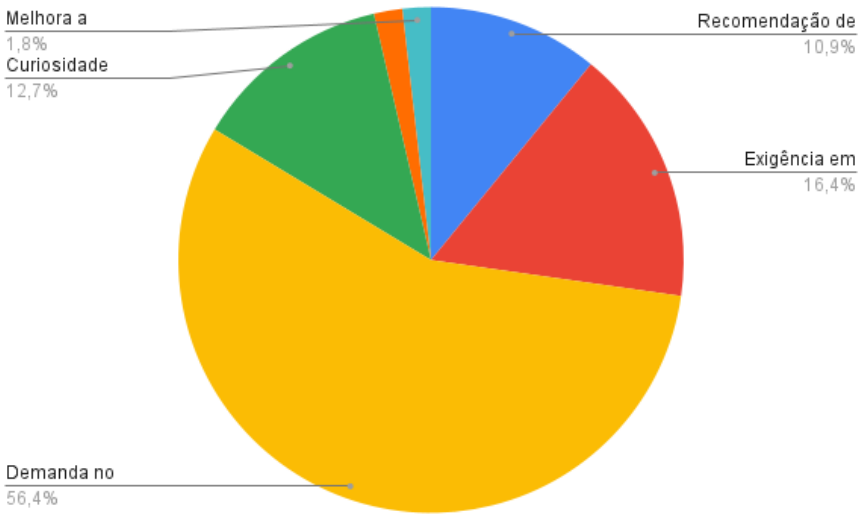
prática em pequenos projetos, pode não apenas tornar o aprendizado mais eficiente, mas também aumentar a confiança dos estudantes em seu próprio código, promovendo boas práticas como a manutenção de versões funcionais e o teste incremental durante o desenvolvimento. Esses benefícios, apontados por Edwards, coincidem com as vantagens percebidas pelos participantes, como a maior confiabilidade do código e a redução de *bugs*.

4.4 Prática e Incentivo ao uso de TDD.

A principal razão apontada pelos entrevistados para não utilizarem o desenvolvimento orientado a testes em seus projetos é a falta de conhecimento técnico, mencionada por 57,1% deles. Além disso, 20,4% declararam nunca ter tido a oportunidade de aplicar a técnica, enquanto 10,2% acreditam que sua utilização não é necessária e apenas 8,2% afirmaram utilizar o TDD em seus projetos.

No entanto, isso não significa que os entrevistados sejam totalmente contrários ao uso do TDD. A maioria afirmou que o principal fator que os motivaria a adotar a técnica seria uma maior demanda por ela no mercado de trabalho. Outros se sentiriam incentivados a utilizá-la caso fosse exigida em projetos acadêmicos, ou então por um interesse pessoal em conhecê-la melhor, conforme ilustrado no gráfico 4 a seguir:

Figura 5 – Principais motivações para a adoção de TDD



Fonte: Elaborado pelo autor(2025).

Sendo que, na realidade, o TDD é uma técnica bastante relevante no mercado. Segundo Neto (2015), o TDD é uma prática popular entre os desenvolvedores de software no setor de desenvolvimento, uma vez que funcionalidades que não operam corretamente não agregam valor ao negócio. O desenvolvimento orientado a testes aumenta a expectativa de funcionamento adequado do *software*, proporcionando assim um valor real às organizações.

A análise dos dados obtidos permitiu identificar percepções, dificuldades e motivações relacionadas ao uso do TDD entre os estudantes de Ciência da Computação do Campus VII

da UEPB. Com base nessas observações, a próxima e última seção apresenta as conclusões do estudo, bem como reflexões sobre os resultados encontrados e sugestões para trabalhos futuros.

5 CONSIDERAÇÕES FINAIS E SUGESTÕES PARA TRABALHOS FUTUROS

Este trabalho teve como objetivo investigar o nível de conhecimento, aceitação e utilização da técnica de desenvolvimento orientado a testes (TDD) entre os estudantes do curso de Ciência da Computação da UEPB, Campus VII. Por meio da aplicação de um questionário *online* direcionado a alunos a partir do 6º período, buscou-se compreender como o tema é abordado ao longo do curso, especialmente na disciplina de Engenharia de *Software*, e quais são os principais fatores que influenciam sua adoção ou rejeição nos projetos acadêmicos. Os resultados foram:

- **(QP1.)** Os alunos do campus VII da UEPB aprenderam sobre as práticas de desenvolvimento de testes de *software* de forma introdutória.
- **(QP3.)** Não houve a necessidade de utilizar testes de *software* por meio de cobrança dos professores.

A análise das respostas relacionadas às questões QP1 e QP3 permitiu verificar que, embora a maioria dos alunos tenha tido contato teórico com testes de *software*, essa exposição prática é bastante limitada, o conteúdo é geralmente apresentado em disciplinas como Engenharia de *Software*, mas de forma introdutória, sem aprofundamento prático. Além disso, foi possível constatar que há pouca ou nenhuma cobrança por parte dos professores para que os alunos incluam testes em seus projetos ou trabalhos acadêmicos, essa ausência de exigência institucional contribui diretamente para a baixa aplicação prática das técnicas de teste durante a formação.

- **(QP2.)** Os estudantes não foram preparados para realizar a aplicação de Desenvolvimento Orientado a Testes.
- **(QP4.)** Existe interesse em aprender e realizar a aplicação de desenvolvimento de testes de *software* e metodologias como o TDD.

Referente às questões QP2 e QP4, os resultados indicaram que os alunos não se sentem devidamente preparados para aplicar metodologias como o Desenvolvimento Orientado a Testes. Essa falta de preparo está diretamente relacionada à escassez de exemplos práticos em sala de aula e à ausência de um ambiente que promova a prática constante. Apesar disso, uma parte significativa dos estudantes demonstrou interesse em aprender e utilizar tais técnicas, reconhecendo sua importância no desenvolvimento profissional. O descompasso entre interesse e preparo evidencia uma lacuna que pode ser suprida com a reformulação do currículo ou a oferta de atividades práticas extracurriculares.

Diante dos resultados obtidos, este trabalho pode servir como base para diferentes iniciativas voltadas ao ensino de testes de *software*, uma possibilidade é a replicação da pesquisa em outras instituições de ensino, a fim de comparar contextos e ampliar a compreensão sobre a formação em testes de *software* no ensino superior. Também pode ser interessante realizar uma análise aprofundada da percepção dos docentes sobre o ensino de testes de *software*, buscando

identificar possíveis barreiras, motivações e práticas adotadas em sala de aula, ou então propor planos de ensino ou ementas que incorporem o ensino de testes de *software* e metodologias como o TDD, promovendo sua inserção gradual e prática nas disciplinas de programação. Por fim, sugere-se o desenvolvimento de uma ferramenta didática voltada especificamente para o ensino de testes de *software*, que auxilie tanto professores quanto estudantes na aprendizagem e aplicação desses conceitos no contexto acadêmico.

REFERÊNCIAS

ANWER, F. *et al.* Agile software development models tdd, fdd, dsdm, and crystal methods: A survey. **International journal of multidisciplinary sciences and engineering**, v. 8, n. 2, p. 1–10, 2017. Citado 3 vezes nas páginas 12, 15 e 20.

ARAúJO, R. **TDD: A abordagem orientada a testes**. 2023. Acesso em: 15 out. 2024. Disponível em: https://blog.grancursosonline.com.br/tdd-a-abordagem-orientada-a-testes/. Citado na página 20.

BACH, J. Exploratory testing explained. **Online: http://www. satisfice. com/articles/et-article. pdf**, p. 1–10, 2003. Citado na página 19.

BARROS, G. S.; JUNIOR, L. R. P. **Análise experimental entre as técnicas TDD e Test-Last no processo de manutenção corretiva de software**. [S.l.]: Universidade Católica do Salvador, 2019. Citado na página 19.

BECK, K. **Programação extrema explicada [recurso eletrônico]: acolha as mudanças**. Porto Alegre: Bookman, 2008. Dados eletrônicos. Editado também como livro impresso em 2004. ISBN 978-85-7780-306-4. Citado na página 17.

BECK, K. **TDD: Desenvolvimento Guiado por Testes**. Porto Alegre: Bookman, 2010. ISBN 978-85-7780-747-5. Citado na página 19.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. *In*: IEEE. **Future of Software Engineering (FOSE'07)**. [S.l.], 2007. p. 85–103. Citado 2 vezes nas páginas 18 e 27.

BORGES, E. N. Conceitos e benefícios do test driven development. **Universidade federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brasil**, 2006. Citado na página 18.

EDWARDS, S. H. Improving student performance by evaluating how well students test their own programs. **Journal on Educational Resources in Computing (JERIC)**, ACM New York, NY, USA, v. 3, n. 3, p. 1–es, 2003. Citado 2 vezes nas páginas 13 e 28.

FOWLER, M.; HIGHSMITH, J. *et al.* The agile manifesto. **Software development**, [San Francisco, CA: Miller Freeman, Inc., 1993-, v. 9, n. 8, p. 28–35, 2001. Citado na página 15.

KIRCHER, M. *et al.* Distributed extreme programming. **Extreme Programming and Flexible Processes in Software Engineering, Italy**, p. 66–71, 2001. Citado na página 17.

LEUNG, H. K.; WHITE, L. A study of integration testing and software regression at the integration level. *In*: IEEE. **Proceedings. Conference on Software Maintenance 1990**. [S.l.], 1990. p. 290–301. Citado na página 19.

MADEYSKI, L.; KAWALEROWICZ, M. Continuous test-driven development-a novel agile software development practice and supporting tool. *In*: SCITEPRESS. **International Conference on Evaluation of Novel Software Approaches to Software Engineering**. [S.l.], 2013. v. 2, p. 260–267. Citado na página 20.

MILLER, R.; COLLINS, C. T. Acceptance testing. **Proc. XPUniverse**, v. 238, 2001. Citado na página 19.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011. Citado na página 18.

NETO, A. M. B. **Em direção a um ambiente de desenvolvimento de software orientado por comportamento**. Dissertação (Dissertação (Mestrado Profissional)) — Universidade Federal de Pernambuco, Centro de Informática, Recife, 2015. Orientador: Vinicius Cardoso Garcia. Inclui referências e apêndices. Citado na página 29.

OLIVEIRA, R. B. de *et al.* **Utilização de Padrões para Otimizar a Automação de Testes Funcionais de Software**. 2007. Citado na página 19.

PALMA, N. A. **Testes unitários evolutivos**. Dissertação (Mestrado) — Universidade de Évora, 2007. Citado na página 19.

PEREIRA, P.; TORREãO, P.; MARçAL, A. S. Entendendo Scrum para gerenciar projetos de forma ágil. **Mundo PM**, São Paulo, v. 1, n. 14, p. 64–71, 2007. Citado 3 vezes nas páginas 15, 16 e 26.

PINTO, V. H. S. C. **Estudo e Definição de Teste de Software para o contexto de Sistemas Multi-tenants**. Tese (Doutorado) — Universidade de São Paulo, 2019. Citado na página 18.

SCHWABER, K.; SUTHERLAND, J. The scrum guide. **Scrum Alliance**, v. 21, n. 1, p. 1–38, 2011. Citado 2 vezes nas páginas 15 e 16.

SHRIVASTAVA, A. *et al.* A systematic review on extreme programming. *In*: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2021. v. 1969, n. 1, p. 012046. Citado 2 vezes nas páginas 17 e 18.

SILVA, J. dos S. *et al.* O processo de teste de software. **Tecnologias em Projeção, Distrito Federal**, v. 7, n. 2, p. 99, 2016. Citado na página 12.

SOARES, M. dos S. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, v. 3, n. 1, 2004. Citado 2 vezes nas páginas 15 e 16.

SOUSA, R. B. A. **TDD: um estudo de caso sobre sua utilização nos projetos de desenvolvimento de software.** [S.l.]: Universidade Federal de Campina Grande, 2021. Citado 2 vezes nas páginas 12 e 19.

Universidade Estadual da Paraíba. **Projeto Pedagógico do Curso de Computação – Campus VII**. Patos: Universidade Estadual da Paraíba, 2016. Disponível em: urlhttps://uepb.edu.br/download/projeto-pedagogico-do-curso-de-computacao-campus-vii/. Acesso em: 10 set. 2024. Citado 2 vezes nas páginas 12 e 23.

WAZLAWICK, R. S. **Metodologia de pesquisa para ciência da computação**. 3. ed. Rio de Janeiro: LTC, 2021. Citado na página 21.

APÊNDICE A – QUESTIONÁRIO SOBRE APRENDIZADO E USO DE TDD NA UEPB CAMPUS VII

Pesquisa sobre TDD no ensino universitário no campus VII da UEPB

Está é uma pesquisa realizada para um trabalho de conclusão de curso, visando obter dados sobre o conhecimento sobre TDD dos estudantes da UEPB do Campus VII

O Autor desse trabalho é o estudante da UEPB Campus VII Noberto Nunes Do Nascimento Junior, visando obter dados para o seu trabalho "ANÁLISE DA ACEITAÇÃO E UTILIZAÇÃO DO DESENVOLVIMENTO ORIENTADO POR TESTES (TDD) EM PROJETOS ACADÊMICOS: UM ESTUDO DE CASO COM ALUNOS DE CIÊNCIA DA COMPUTAÇÃO DO CAMPUS VII DA UEPB, PATOS, PB", sob orientação do professor Vinicius Augustus Alves Gomes.

Esta pesquisa tem como alvo os estudantes do campus que tenham cursado ou estejam cursando a disciplina de Engenharia de *Software* I, sendo um questionário curto de aproximadamente 7 minutos para responder em completude.

Introdução do tema:
Test-Driven Development (TDD) é uma técnica de desenvolvimento de *software* onde os testes automatizados são escritos antes do código-fonte, seguindo um ciclo básico de três etapas:

1. Escrever um teste que falha;
2. Implementar o código mínimo para passar no teste e;
3. Refatorar o código para melhorar sua qualidade.

Com essa abordagem busca-se melhorar a qualidade do software, reduzir erros e facilitar a manutenção.

* Indica uma pergunta obrigatória

1. E-mail *

2. A disciplina de ES me proveu conhecimentos sobre métodos ágeis *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

3. A disciplina de ES me proveu conhecimentos sobre testes de software *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

4. A disciplina de ES me promoveu conhecimento sobre Test Driven Development (TDD) *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

5. Compreendo o que é e sei utilizar Test Driven Development (TDD) *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

6. Costumo realizar testes de software em meus projetos. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

7. O que te impediria de utilizar testes automatizados em seus projetos? *

Marcar apenas uma oval.

- ◯ Falta de conhecimento técnico
- ◯ Falta de tempo
- ◯ Não vejo necessidade
- ◯ Outro: _____

8. Me sinto confiante para aplicar TDD em um projeto real de software. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

9. Onde foi que o adquiriu conhecimento sobre TDD: *

Marque todas que se aplicam.

- ☐ Faculdade
- ☐ Trabalho
- ☐ Estudos pessoais
- ☐ Não possuo
- ☐ Outro: _____

10. Eu acho que a faculdade oferece suporte suficiente para o aprendizado de TDD * e outras práticas de testes de software.

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

11. Se você nunca usou TDD, qual a principal razão?

Marcar apenas uma oval.

- ◯ Não conheço a técnica
- ◯ Nunca tive oportunidade
- ◯ Acredito que não é necessário
- ◯ Faço uso de TDD
- ◯ Outro: _____

12. O que te motivaria a aprender e usar TDD? *

Marcar apenas uma oval.

- ◯ Recomendação de professores
- ◯ Exigência em projetos acadêmicos
- ◯ Demanda no mercado de trabalho
- ◯ Curiosidade pessoal
- ◯ Outro: _____

13. Acredito que o TDD é útil para desenvolvimento de software. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

14. Eu acredito que o uso de TDD aumentaria a produtividade do desenvolvedor. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo totalmente

15. Em sua opinião, qual o principal benefício do TDD *

Marcar apenas uma oval.

- ◯ Código mais confiável e com menos bugs
- ◯ Melhoria no design e na modularidade do código
- ◯ Facilidade na refatoração
- ◯ Documentação viva
- ◯ Redução do tempo de correção de erros
- ◯ Outro: _____

16. Em sua opinião qual o principal malefício *

Marcar apenas uma oval.

- ◯ Aumento do tempo de desenvolvimento inicial
- ◯ Curva de aprendizado alta
- ◯ Excesso de testes pode levar a manutenção complexa
- ◯ Testes podem limitar a criatividade do design
- ◯ Falsa sensação de segurança
- ◯ Outro: _____