UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

EMERSON PAULO SOARES DE SOUZA

Criação de diretrizes para a prevenção e correção de *smells* em testes manuais de regressão da indústria de *smartphones*

Recife-PE
2024

EMERSON PAULO SOARES DE SOUZA

Criação de diretrizes para a prevenção e correção de *smells* em testes manuais de regressão da indústria de *smartphones*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de mestre em Ciência da Computação

.

Área de concentração: Engenharia de Software e Linguagens de programação.

Orientador: Prof. Dr. André Luís de Medeiros Santos

Coorientador: Prof. Dr. Elvys Alves Soares

Recife-PE

2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Souza, Emerson Paulo Soares de.
Criação de diretrizes para a prevenção e correção de Smells em testes manuais de regressão da indústria de smartphones / Emerson Paulo Soares de Souza. - Recife, 2024.
79f.: il.

Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2024.
Orientação: André Luis de Medeiros Santos.
Coorientação: Elvys Alves Soares.
Inclui referências e anexos.

1. Testes de Software; 2. Testes de Regressão; 3. Bad Smells; 4. Dispositivos Móveis. I. Santos, André Luis de Medeiros. II. Soares, Elvys Alves. III. Título.

UFPE-Biblioteca Central

Emerson Paulo Soares de Souza

“Criação de diretrizes para a prevenção e correção de smells em testes manuais de regressão da indústria de smartphones”

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software e Linguagens de Programação.

Aprovado em: 29/10/2024.

BANCA EXAMINADORA

_____

Prof. Dr. Breno Alexandro Ferreira de Miranda
Centro de Informática / UFPE

_____

Prof. Dr. Márcio de Medeiros Ribeiro
Instituto de Computação / UFAL

_____

Prof. Dr. André Luis de Medeiros Santos
Centro de Informática / UFPE
(**orientador**)

Dedico este trabalho a todos que torceram positivamente por mim!

Obrigado!

AGRADECIMENTOS

Este trabalho simboliza um grande passo em minha vida acadêmica e para chegar até aqui, agradeço primeiramente a Deus pela permissão em sonhar, pois sem acreditar que seria possível, esse dia não chegaria.

Aos meus pais, Geraldo Bernardino de Souza e Ozileide Soares Bezerra pelo apoio de sempre… Não é todo dia que um filho de pescador e agricultora chega até aqui.

Às minhas irmãs, Eliane Soares e Elane Soares pela companhia nos momentos de pouca fé. Vocês são minhas inspirações diárias!

Aos professores, André Santos e Elvys Soares pela confiança, pela oportunidade, pelas orientações e pela humanidade. Em momentos turbulentos, a voz e os ouvidos de vocês fizeram toda a diferença.

Aos professores convidados para contribuir e avaliar, Breno Miranda e Márcio Ribeiro, ao qual já tive a oportunidade de aprender um pouco mais sobre o universo de testes de software com cada um.

Aos meus amigos […] Só eles sabem o que tiveram que aturar durante esse período de curso. Grato pelo apoio de todos!

Ao projeto CIn/Motorola, pela oportunidade de aprender tanto, todos os dias. Espero que outros bons frutos sejam plantados e colhidos futuramente.

Muito obrigado!

“A qualidade nunca é um acidente. É sempre o resultado do esforço inteligente.”

(John Ruskin).

RESUMO

A norma ISO/IEC 9126 define qualidade de software como a capacidade de um produto satisfazer necessidades explícitas e implícitas dos usuários. A garantia de qualidade é essencial para desenvolver sistemas confiáveis e eficientes, com os testes de software desempenhando um papel crucial. No entanto, a presença de *test smells* pode comprometer a qualidade dos testes, dificultando sua legibilidade, manutenção e eficácia. Para mitigar esses problemas, a refatoração de testes é uma estratégia eficaz, melhorando a clareza e a manutenção dos testes sem alterar seu comportamento. Embora a maioria dos estudos se concentre em testes automatizados, *test smells* também afetam testes manuais, que carecem de uma estrutura formal e dependem de processos humanos. Este trabalho propõe diretrizes para prevenir *smells* em testes manuais, especialmente em testes de regressão na indústria de *smartphones*. A metodologia incluiu a análise de suítes de testes manuais, validação com profissionais da indústria e a proposição de diretrizes baseadas nos resultados. As diretrizes visam melhorar a qualidade dos testes manuais e, consequentemente, do *software*.

Palavras-chave: Engenharia de Software, Testes de software, Testes Manuais, Testes de Regressão, Bad Smells, Refatoração, Linguagem Natural, Dispositivos móveis.

ABSTRACT

The ISO/IEC 9126 standard defines software quality as the ability of a product to satisfy explicit and implicit user needs. Quality assurance is essential to develop reliable and efficient systems, with software testing playing a crucial role. However, the presence of test smells can compromise the quality of tests, hindering their readability, maintainability, and effectiveness. To mitigate these problems, test refactoring is an effective strategy, improving the clarity and maintainability of tests without changing their behavior. Although most studies focus on automated testing, test smells also affect manual testing, which lacks a formal structure and relies on human processes. This work proposes guidelines to prevent smells in manual testing, especially in regression testing in the smartphone industry. The methodology included the analysis of manual test suites, validation with industry professionals, and the proposal of guidelines based on the results. The guidelines aim to improve the quality of manual testing and, consequently, of software.

Keywords: Software Engineering, Software Testing, Manual Testing, Regression Testing, Bad Smells, Refactoring, Natural Language, Mobile Devices

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

BSTQB	Brazilian Software Testing Qualifications Board (Conselho Brasileiro de Qualificação em Teste de Software)
CIN	Centro de Informática
IEC	International Electrotechnical Commission
ISO	International Organization of Standardization
TCS	Test Case Selection (Seleção de Casos de Testes)
TCP	Test Case Prioritization (Priorização de Casos de Teste)
TDD	Test Driven Development (Desenvolvimento orientado por testes)
TSE	Tribunal Superior Eleitoral
TSR	Test Suite Reduction (Redução do Conjunto de Testes)
NLP	Natural Language Processing (Processamento de Linguagem Natural)
SUT	System Under Test (Sistema sob teste)
UFPE	Universidade Federal do Pernambuco

SUMÁRIO

1 INTRODUÇÃO

A norma ISO/IEC 9126, citada por Maciel (2011), define qualidade de software como a totalidade de características de um produto de software que lhe confere a capacidade de satisfazer necessidades explícitas (aquelas citadas pelos usuários clientes de software) e implícitas (necessidades subjetivas dos usuários, ou seja, aquelas que não são citadas, mas que são consideradas óbvias no desenvolvimento e manutenção do produto de software). Assim, a garantia de qualidade de software é um processo que objetiva o desenvolvimento de sistemas confiáveis e eficientes.

Entre as diversas estratégias adotadas para assegurar a qualidade, os testes de software desempenham um papel fundamental. Testes de software são métodos sistemáticos utilizados para verificar se um software atende aos requisitos especificados e para identificar comportamentos incorretos. Eles são essenciais para garantir que o software esteja livre de defeitos e funcione conforme o esperado, aumentando a confiança dos usuários finais (Aboradan, 2022).

No entanto, a qualidade dos testes de software pode ser comprometida pela presença de *test smells*, que são indicadores de problemas de qualidade nos testes, semelhantes aos *code smells* no código de produção. Eles representam aspectos mal projetados e implementados que podem dificultar a legibilidade, manutenção e eficácia dos testes. Exemplos de *test smells* incluem duplicação de código, verificações ausentes e comportamento de execução não determinístico (Garousi, Vahid; Küçük, Barış, 2018, em tradução livre). Como a presença de *test smells* pode levar a uma cobertura de teste inadequada e a resultados de teste não confiáveis, comprometendo a qualidade geral do software, desde sua conceituação eles têm sido foco de diversos estudos na área de teste de software (Bavota et al. 2012, Kim 2020, citado por Damasceno e Bezerra, 2023).

Para mitigar os efeitos negativos dos *test smells*, a refatoração é uma estratégia eficaz. Refatorar código consiste em aplicar uma reestruturação ou remodularização no sistema sem alterar suas funcionalidades (Fowler;Beck,1999), e refatorar testes envolve a reestruturação do código de teste para melhorar sua legibilidade, manutenção e eficácia, sem alterar seu comportamento observável (van

Deursen et al. 2001). Este último processo inclui a remoção de duplicações, a simplificação de estruturas complexas e a melhoria da clareza dos testes. A refatoração contínua dos testes é essencial para manter a qualidade dos testes ao longo do ciclo de vida do software.

Embora a maioria dos estudos sobre *test smells* tenha se concentrado em testes automatizados, é importante reconhecer que *test smells* também podem existir em testes manuais. Testes manuais são frequentemente utilizados em cenários onde a automação é impraticável ou inviável. No entanto, a ausência de uma estrutura formal e a dependência de processos humanos podem levar à introdução de *test smells* em testes manuais. Tal afirmação foi feita no estudo de Hauptmann et al. (2013), que investigou como identificar *smells* em testes escritos em linguagem natural, propondo, inclusive, uma metodologia para sua detecção com foco em aspectos como a clareza, consistência e precisão das descrições dos testes. Esses *smells*, em específico, podem incluir procedimentos de teste redundantes, falta de documentação adequada e inconsistências nos resultados dos testes.

Apesar da pesquisa em *test smells* de testes manuais ter sido recentemente retomada (Soares et al. 2023 e Aranda et al. 2024), incluindo propostas de ferramentas para a análise automatizada destes testes manuais de software, é importante reconhecer que estes testes não possuem padrão definido para a sua estrutura, o que gera esforço de implementação – considerando que as ferramentas teriam que ser adequadas – em situações onde a automação já não se paga e por isto os testes manuais estão sendo empregados. Tal é o caso dos testes de regressão da indústria de smartphones, que são usados para garantir que o desenvolvimento e integração de novas funcionalidades não causará impacto negativo em funcionalidades pré-existentes e de funcionamento conforme requisitos já atestado por testes (Viana, 2006).

Assim, dada a importância dos testes manuais na garantia de qualidade de software e a falta de padronização destes – dificultando a ação de ferramentas de análise automática de testes manuais –, a existência de um conjunto de diretrizes para a prevenção de *test smells* nesses testes pode ser vista como importante contribuição. Neste sentido, diretrizes práticas podem ajudar os testadores a

reconhecer e corrigir problemas de qualidade em seus testes manuais, melhorando a eficácia e a confiabilidade dos testes.

Este trabalho, portanto, tem como objetivo geral, propor um conjunto de diretrizes para abordar *test smells* em testes manuais de software, desenvolvidos sob a ótica dos testes manuais de regressão da indústria de *smartphones*, com a intenção de aprimorar a qualidade dos testes e, consequentemente, a qualidade do software. Para isso, como objetivos específicos: (1) Fazer revisão de literatura levando em consideração publicações da última década. (2) Analisar os *Smells* encontrados nos testes manuais em diferentes sistemas, (3) Validar os resultados encontrados e (4) Sugerir diretrizes para evitar *Smells*.

Para desenvolver as diretrizes, a seguinte metodologia foi utilizada: primeiro, uma análise para estudar e identificar *Smells* das suítes de testes manuais do Sistema Brasileiro de Votação e também da suíte de testes de regressão de um parceiro da indústria de smartphones; segundo, um estudo foi realizado com profissionais deste parceiro industrial para validar a importância dos *smells* identificados no estudo anterior; em seguida, um novo estudo foi conduzido para verificar a opinião de profissionais de testes do parceiro industrial quanto às estratégias de refatoramento dos smells identificados; por fim, um conjunto de diretrizes para a prevenção de *smells* neste tipo de testes foi proposto com base nos resultados dos estudos e propostas validadas, também formalizados em ferramenta voltada a testes com estrutura específica (Soares et al. 2023 e Aranda et al. 2024).

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os conceitos necessários para a compreensão do método utilizado na abordagem e para a interpretação dos resultados obtidos.

2.1 TESTES DE SOFTWARE

Conforme descrito no Syllabus versão 4.0, desenvolvido pelo Conselho Brasileiro de Qualificação em Teste de Software (BSTQB), o teste de software é essencial para avaliar a qualidade do software e mitigar o risco de falhas durante sua operação. Trata-se de um conjunto de atividades destinadas a identificar defeitos e avaliar a qualidade dos artefatos de software (Stapp; Roman; Pilaeten, 2024).

Abordan (2022) define o teste de software como um método para verificar se um sistema de software atende aos seus requisitos e para identificar situações em que o comportamento do software é incorreto. O objetivo principal é assegurar que o sistema funcione conforme o esperado. Para isso, é fundamental considerar a forma como os documentos de requisitos foram elaborados, as regras de negócios e as expectativas para cada funcionalidade do software.

Na prática, o teste de software geralmente inclui pelo menos um teste para cada requisito especificado no documento de requisitos, sendo realizado de forma manual ou automatizada (Abordan, 2022). A qualidade do software é definida pela conformidade com os requisitos funcionais e de desempenho explicitamente declarados, pelos padrões de desenvolvimento documentados e pelas características implícitas esperadas de qualquer software profissionalmente desenvolvido (Pressman, 1995).

Dada a importância dos testes no processo de garantia de qualidade, conclui-se que o sucesso de um sistema está diretamente relacionado à qualidade dos testes aos quais foi submetido (Orso e Rothermel, 2014, citado por Damasceno, 2023).

2.1.1 Tipos de Testes de Software

A atividade de teste de software é um elemento crítico na garantia de qualidade, representando a última revisão das especificações, projeto e codificação (Pressman, 1995). Os testes podem ser automatizados ou manuais e devem ser planejados e projetados conforme o processo de desenvolvimento do software (Galin, 2004).

2.1.1.1 Testes Automatizados

Os testes automatizados envolvem a execução de *scripts* de teste sem a necessidade de interação manual. Segundo Bernardo (2011), a automação de testes é uma prática eficaz para prevenir erros durante a implementação e manutenção do sistema. Métodos como Testes *a priori* e Desenvolvimento Dirigido por Testes (TDD) são utilizados para criar testes antes da implementação, garantindo alta cobertura de verificação.

Bernardo (2011) também destaca que a automação de testes é uma prática recomendada em metodologias ágeis para assegurar a qualidade do software. As novas técnicas e ferramentas de desenvolvimento facilitam a escrita de testes para trechos específicos de código, integração de módulos, interfaces gráficas e bancos de dados. Embora os testes automatizados sejam mais caros devido aos custos de configuração, eles são preferidos quando há orçamento disponível (Lopes, 2023).

2.1.1.2 Testes Manuais

Realizados por humanos após a implementação de uma funcionalidade, os testes verificam se o software funciona conforme esperado. A compreensão das especificações do software é crucial para a execução correta dos testes (Aboradan, 2022).

Os testes manuais considerados neste trabalho são descritos seguindo os elementos de acordo com a Tabela 1, abaixo:

Tabela 1: Elementos obrigatórios para escrita de testes de software

Nome do Teste:		
Objetivo:		
Pré condições:		
Passos:		
1	ação	verificação
…	…	…
n	ação	verificação

Fonte: Soares (2023), tradução nossa

As saídas devem ser pré-computadas com base nos requisitos do programa para evitar a interpretação errônea de resultados plausíveis, mas incorretos (Naik, Tripathy, 2011; Galin, 2004; Soares et. al, 2023).

2.1.1.2.1 Testes de Regressão

Os Testes de Regressão (TR) verificam se modificações no software não introduziram novas falhas ou reintroduziram falhas previamente corrigidas. À medida que o software evolui, o conjunto de testes de regressão tende a aumentar, tornando essa técnica dispendiosa.

Esses testes confirmam que nenhuma consequência adversa foi causada por alterações, incluindo correções já testadas. As consequências podem afetar o componente alterado, outros componentes do sistema ou sistemas conectados (Stapp; Roman; Pilaeten, 2024). A análise de impacto é recomendada para otimizar a extensão dos testes de regressão, identificando partes do software potencialmente afetadas.

Os conjuntos de testes de regressão são executados repetidamente, com o número de casos de teste aumentando a cada iteração ou versão (Stapp; Roman; Pilaeten, 2024). Para tornar os testes de regressão mais econômicos, técnicas como Seleção de Casos de Teste (TCS), Redução do Conjunto de Testes (TSR) e Priorização de Casos de Teste (TCP) são propostas na literatura (Engström;

Runeson; Skoglund, 2010; Yoo; Harman, 2012; Catal; Mishra, 2013; Khan et al., 2018, citado por Siqueira, 2022).

2.2 TEST SMELLS

De acordo com Peruma et al. (2019), os *test smells* são desvios de como os casos de teste devem ser organizados, implementados e como devem interagir entre si. Embora os “smells” sejam subjetivos e abertos a debate, esses desvios indicam potenciais problemas de concepção no código de teste. Um exemplo é o *Redundant Assertion*, disponível no *The Open Catalog of Smells*[1] visto abaixo e descrito com mais detalhes no item 20 da seção 2.2.1.

```
@Test
public void testTrue() {
    assertEquals(true, true);
}
```

No exemplo acima, o teste deveria retornar um resultado binário que indica se o resultado pretendido está correto ou não, e não devolver o mesmo resultado independentemente da entrada. Problemas como este prejudicam o desempenho dos testes e a manutenção do software.

Estudos indicam que os *test smells* afetam negativamente a manutenção e a compreensão do código de teste (Bavota et al., 2012; Spadini et al., 2018; Santana, 2021). Uma das razões para esses problemas é a pressão do tempo ou a inexperiência, que leva os programadores a ignorar regras de boas práticas conhecidas na literatura. Isso resulta em código de testes unitários que são difíceis de manter e compreender. Portanto, é essencial entender seus efeitos potenciais e propor mecanismos, estratégias e ferramentas para mitigá-los.

Infelizmente, os testes manuais são frequentemente escritos sem as melhores práticas de engenharia de software, como abstração e reutilização. Descrições de testes manuais em linguagem natural, assim como testes unitários, sofrem de problemas semelhantes (Hauptmann, 2013). Pesquisas anteriores concluíram que testes de sistema em linguagem natural contêm uma quantidade

[1] Disponível em https://test-smell-catalog.readthedocs.io/

significativa de redundância, o que pode aumentar consideravelmente os custos de manutenção e execução de testes, necessitando de refatorações (Hauptmann, 2013).

2.2.1 Smells em testes automatizados

Inicialmente, Van Deursen et al. (2001) documentaram um conjunto inicial de 11 *test smells*: *Assertion Roulette, Eager Test, For Testers Only, General Fixture, Indirect Testing, Lazy Test, Mystery Guest, Resource Optimism, Sensitive Equality, Test Code Duplication e Test Run War*. Posteriormente, Peruma et al. (2019) catalogaram 12 novos tipos: *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*, descritos na Tabela 2, abaixo:

Tabela 2: Lista de smells em testes automatizados

N.	***Smell***	**Descrição**
1	Assertion Roulette	Testes com múltiplas asserções sem mensagens claras, dificultando a identificação de falhas.
2	Eager Test	Testes que verificam mais do que um comportamento, tornando difícil identificar a causa de falhas.
3	For Testers Only	Testes que não são compreensíveis para desenvolvedores, limitando sua utilidade.
4	General Fixture	Uso de fixtures compartilhadas entre muitos testes, levando a dependências ocultas.
5	Indirect Testing	Testes que verificam o comportamento de uma unidade através de outra, em vez de diretamente.
6	Lazy Test	Testes que não verificam todos os aspectos necessários, deixando brechas na cobertura.

7	Mystery Guest	Testes que dependem de dados externos ou configurações não óbvias, dificultando a compreensão.
8	Resource Optimism	Testes que assumem a disponibilidade de recursos externos, sem verificações adequadas.
9	Sensitive Equality	Testes que falham devido a comparações excessivamente sensíveis, como diferenças mínimas em valores numéricos.
10	Test Code Duplication	Código de teste repetido em vários testes, dificultando a manutenção.
11	Test Run War	Testes que interferem uns com os outros, causando falhas intermitentes.
12	Conditional Test Logic	Testes que contêm lógica condicional, tornando-os difíceis de entender e manter.
13	Constructor Initialisation	Testes que dependem de inicializações complexas no construtor, dificultando a configuração.
14	Default Test	Testes que usam valores padrão inadequados, não refletindo cenários reais.
15	Duplicate Assert	Múltiplas asserções verificando a mesma condição, sem necessidade.
16	Empty Test	Testes que não contêm verificações, sendo inúteis.
17	Exception Handling	Testes que não lidam adequadamente com exceções, podendo mascarar problemas.
18	Ignored Test	Testes que são ignorados ou desativados, sem justificativa clara.
19	Magic Number Test	Testes que usam números “mágicos” sem explicação, dificultando a compreensão.
20	Redundant Assertion	Asserções que não adicionam valor, sendo desnecessárias.

21	Redundant Print	Impressões no código de teste que não contribuem para a depuração.
22	Sleepy Test	Testes que dependem de delays artificiais, tornando-os lentos e não confiáveis.
23	Unknown Test	Testes cujo propósito não é claro, dificultando a manutenção.

Fonte: Autor (2024).

Após a identificação dos Smells em códigos de testes, foi observado que em testes manuais, há comportamentos semelhantes. Nesse sentido, na próxima seção descrevemos *Smells* encontrados na literatura até o momento.

2.2.1 *Smells* em testes manuais

De acordo com a literatura, o primeiro autor a escrever sobre Smells em testes manuais escritos em Linguagem Natural foi o Hauptmann (2013), no artigo intitulado como *Hunting for Smells in Natural Language Tests*. Neste artigo, o autor identifica 7 *smells* e os problemas causados por eles. Em sua publicação 10 anos depois, Soares et al. (2023) complementou a lista adicionando 6 novos *smells*. Na Tabela 3, abaixo, está descrito o conjunto de *test smells* conhecidos para testes manuais:

Tabela 3: Lista de *Smells* em testes manuais

N.	Smell	Descrição
1	Hard-Coded Values	Testes que usam valores fixos em vez de variáveis, dificultando a manutenção e a flexibilidade.
2	Long Test Steps	Testes com passos excessivamente longos, tornando-os difíceis de ler e entender.
3	Conditional Tests	Testes que contêm lógica condicional, complicando a compreensão e a manutenção.

4	Badly Structured Test Suite	Conjunto de testes mal organizado, dificultando a navegação e a execução eficiente.
5	Test Clones	Testes duplicados ou muito semelhantes, aumentando a redundância e a manutenção.
6	Ambiguous Tests	Testes que não são claros em seu propósito ou verificações, causando confusão.
7	Inconsistent Wording	Uso inconsistente de terminologia nos testes, dificultando a compreensão.
8	Eager Action	Testes que executam ações desnecessárias ou prematuras, complicando a depuração.
9	Misplaced Action	Ações de teste colocadas fora de contexto, dificultando a lógica do teste.
10	Misplaced Precondition	Pré-condições de teste mal posicionadas, afetando a precisão dos resultados.
11	Misplaced Verification	Verificações de teste fora de lugar, comprometendo a validade do teste.
12	Tacit Knowledge	Testes que dependem de conhecimento implícito, não documentado, dificultando a compreensão por novos membros da equipe.
13	Unverified Action	Ações de teste que não são verificadas, deixando possíveis falhas passarem despercebidas.

Fonte: Autor (2024).

2.2.2 Refatoração de *test smells*

Os *test smells* podem ser eliminados por meio da refatoração, o que melhora a qualidade do código de teste (Spadini et al., 2018; Campos et al., 2021, citado por Damasceno, 2023). O termo “refatoração” foi introduzido por Opdyke (1992) como um processo para aprimorar projetos de software através de transformações que não alteram o comportamento externo do sistema. Por exemplo, corrigir “smells” na estrutura do código sem comprometer suas funcionalidades.

Segundo Fowler e Beck (1999), a refatoração consiste em pequenas transformações no código que preservam seu comportamento original. Cada transformação promove uma reestruturação que torna o código mais reutilizável, legível e coeso. Um código mantém o mesmo comportamento se, após sua execução, os dados gerados e mantidos são os mesmos. Na Figura 01, um exemplo de refatoração do *smell Magic Number Test*, proposto por Peruma et al. (2019), citado por Soares (2020), mostra como valores numéricos não documentados (valores mágicos) podem ser substituídos por constantes ou variáveis com nomes descritivos.

Figura 1: Refatoração do *smell Magic Number Test*

Fonte: Peruma et al. (2019), citado por Soares (2020).

A refatoração pode incluir renomear variáveis para nomes mais significativos, dividir funções grandes, eliminar código repetitivo e reduzir instruções complexas, melhorando a estrutura interna do código sem alterar seu comportamento externo (Fowler, 2020). Inicialmente, a refatoração visava corrigir deficiências de código, mas agora também é aplicada a deficiências arquiteturais e de requisitos, com o objetivo de facilitar a manutenção e evolução do sistema (Opdyke, 1992).

A refatoração é amplamente aceita no Desenvolvimento Orientado por Testes (TDD), uma técnica de desenvolvimento de software que envolve pequenas iterações para desenvolver novas funcionalidades. O processo começa com a implementação de um caso de teste, seguido pelo código necessário para passar no teste, e finalmente a refatoração do código para acomodar as mudanças (Hazin, 2017). Essa prática força os desenvolvedores a criar código guiado pelos testes, resultando em testes automatizados (Hazin, 2017).

Um estudo de Damasceno e Bezerra (2023) identificou que *smells* que requerem técnicas de refatoração como *Extract Method* ou *Inline Resource* são os mais difíceis de remover. Em contraste, os *test smells Assertion Roulette* e *Magic Number Test* são mais fáceis de refatorar, pois necessitam de poucas alterações no código.

Em sistemas orientados a objetos, a refatoração visa melhorar a estrutura e modularização do sistema sem alterar suas funcionalidades (Fowler; Beck, 1999). A refatoração pode ser aplicada manualmente ou automaticamente. A aplicação manual não é recomendada devido ao risco de erros humanos e ao tempo necessário. Ferramentas de refatoração são uma ótima solução para evitar erros e acelerar o processo (Nogueira, 2023).

A ideia principal é permitir que os desenvolvedores sempre busquem maneiras de tornar o código mais fácil de manter e reutilizar. A refatoração frequente é difícil sem uma boa cobertura de testes automatizados (Oliveira, 2020). Os testes devem fazer parte do processo de refatoração para garantir que qualquer alteração no comportamento do código seja detectada, minimizando o risco de interferir em componentes funcionais (Oliveira, 2020).

Barros (2015) destaca a importância da refatoração:

1. **Para manter o código limpo:** Evita problemas como código duplicado, muitos parâmetros e métodos longos, melhorando a legibilidade e reduzindo o tempo de depuração.
2. **Para melhorar o desempenho de um produto:** Garante que a adição de novos recursos ou dados não comprometa a experiência do usuário.
3. **Para economizar tempo e dinheiro:** Mantém o código claro, facilitando a adição de novos recursos no futuro sem a necessidade de desembaraçar código complexo.
4. **Para reduzir débito técnico:** Contribui para a redução do custo contínuo de manutenção do software.
5. **Código desatualizado:** Identifica oportunidades de evolução no código, mantendo-o legível e fácil de manter.

3 ANÁLISE DE SUÍTES DE TESTES MANUAIS

Este capítulo descreve o processo de aquisição de conhecimento sobre *test smells* em testes de software escritos em linguagem natural. O processo se divide em duas etapas: primeiro, são analisados testes manuais do sistema eletrônico de votação brasileiro; em seguida, é analisada uma suíte de testes de uma empresa global de telecomunicações.

3.1 ANÁLISE DE TESTES DO SISTEMA ELETRÔNICO DE VOTAÇÃO BRASILEIRO

Através de uma parceria do Tribunal Superior Eleitoral (TSE) com universidades brasileiras, entre elas a Universidade Federal do Pernambuco (UFPE) surgiu o projeto-piloto para acesso ao código-fonte da urna eletrônica fora do TSE. A parceria foi estabelecida num projeto com 4 etapas, a saber: implantação, execução dos testes automatizados existentes, amostragem e análise dos testes automatizados, amostragem e análise dos testes manuais. Para o escopo desta dissertação, detalharemos as atividades da última etapa.

3.1.1 Caracterização

O ecossistema da urna eletrônica brasileira é formado por diversos módulos, separados por suas funcionalidades. Após a etapa de instalação do ecossistema em computador com as restrições de acesso solicitadas pelo Tribunal Superior Eleitoral, escolheu-se o módulo VOTA, que é um módulo central responsável pelo registro dos votos, como software alvo das investigações.

3.1.2 Amostragem

A suíte de testes recebida contém 136 testes. Como se tratam de testes de regressão, alcançando muitas dezenas de passos cada um, para tornar nosso

esforço de análise manual viável, usamos a Fórmula do Tamanho da Amostra de Cochran para calcular a amostra necessária para obter um nível de confiança de 80% com uma margem de erro de 5% para a suíte de testes analisada. Assim, chegamos ao total necessário de 75 testes na amostra.

A análise da amostra foi feita individualmente por dois pesquisadores, que se reuniram para resolver os pontos de dúvidas. Divergências representaram menos de 5% dos casos e foram solucionadas nestas reuniões.

3.1.3 Resultados

Nesta atividade, foram observados os seguintes *test smells*: *Random Data, Tacit Knowledge, Misplaced Step, Conditional Test, Magic Value, Eager Step* e *Calculating expected results on the fly*. Destes, os smells *Tacit Knowledge, Misplaced Step e Conditional Result Verification* são propostas desta pesquisa. O Quadro 01, abaixo, apresenta exemplos de testes e os *smells* identificados na amostragem.

Quadro 1: *Test smells* identificados em testes manuais da urna eletrônica

Caso de Teste SEVIN-986: VOTA - Encerramento - UESEÇÃO - Validar com sucesso os arquivos de saída do VOTA - BU com 01 voto. [Versão : 2]
As pré-condições não especificam que sessão, especificamente, utilizar. Os cálculos realizados são todos relativos à sessão aleatoriamente escolhida. **(Random Data)** No passo 1, não foi explicado o que significa votação nominal **(Tacit Knowledge)** No passo 2, o resultado do passo 2 é mais um passo **(Misplaced Step)** No passo 3, existe uma lógica de branching nos resultados esperados. É exibido como verificar em sessão normal, mas não como verificar em qualquer outro tipo de sessão. **(Conditional Test)**
Caso de Teste SEVIN-3834: Verificar a integridade da assinatura dos dados de entrada alterados [Versão : 6]
Nas pré-condições, é especificado o horário de 8h para a urna, sem explicação do motivo **(Magic Value)** O passo 2, na verdade, é uma coleção de passos verificados somente ao final **(Eager Step)** O passo 3 assume que a urna está desligada, sem ter comandado seu desligamento em passos anteriores **(Tacit Knowledge)**
Caso de Teste SEVIN-985: VOTA-SEÇÃOBIMÉTRICA - Validar os arquivos de saída, considerando um dos mod. 2009, 2010, 2011,2013. [Versão : 1]

As pré-condições não especificam que sessão, especificamente, utilizar. Os cálculos realizados são todos relativos à sessão aleatoriamente escolhida. **(Random Data)**

As verificações realizadas nos resultados esperados do passo 3 não são previstas pelo caso de teste e devem ser computadas em tempo de execução, de acordo com os dados escolhidos aleatoriamente **(calculating expected results on the fly)**

Caso de Teste SEVIN-988: RED - Encerrar BU para o tot. UESeção - Validar com sucesso os arquivos de saída. BU 00 votos.
[Versão : 1]

Nas pré-condições, é especificado o horário de 16:10 para a urna, sem explicação do motivo **(Magic Value)**
As pré-condições não especificam que sessão, especificamente, utilizar. Os cálculos realizados são todos relativos à sessão aleatoriamente escolhida. **(Random Data)**
No passo 1, há uma espera indeterminada **(Undefined Wait)**

Fonte: Dados da pesquisa (2023)

Em seguida, na Tabela 4, estão as ocorrências dos *smells* encontrados nesta análise.

Tabela 4: Ocorrências de *Smells* no sistema eletrônico de votação brasileiro

***Smell* identificado**	**Quantidade**
Ambiguous Test	2
Conditional Test Logic	23
Eager Step	37
Misplaced Step	2
Unverified Step	11
Tacit Knowledge	38
Undefined Wait	5
Total	**118**

Fonte: Dados da pesquisa (2023)

3.2 ANÁLISE DE TESTES DE REGRESSÃO DE UMA EMPRESA GLOBAL DE TELECOMUNICAÇÕES

O Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE) mantém uma parceria duradoura com uma renomada empresa de telecomunicações, exemplificando a colaboração entre academia e indústria voltada para a inovação tecnológica e o desenvolvimento de soluções avançadas. Esta colaboração proporciona benefícios mútuos, incluindo a oferta de talentos e conhecimento técnico de alto nível, suporte financeiro, infraestrutura e oportunidades de pesquisa aplicada.

A parceria abrange iniciativas como desafios de inovação, cursos de capacitação, eventos técnicos e colaboração em publicações científicas. Essa constante troca de conhecimento e recursos contribui significativamente para o avanço da indústria tecnológica no Brasil, destacando o papel do CIn na formação de mão de obra qualificada e na criação de tecnologias que impulsionam o setor de tecnologia da informação, tanto nacional quanto internacionalmente.

Nesse contexto, a pesquisa de soluções para a melhoria de uma suíte de testes em um cenário real de uma empresa global torna-se um desafio relevante, permitindo a aplicação prática de conhecimentos teóricos. Serão descritas todas as etapas do processo de identificação da suíte de testes e os resultados obtidos.

3.2.1 Caracterização

O plano de testes da suíte de experiências, desenvolvido por diversos engenheiros, apresentava divergências no entendimento de algumas etapas, sendo considerado de difícil compreensão. Por esse motivo, essa suíte foi selecionada para a presente pesquisa.

Os testes são mantidos em uma ferramenta de gestão ágil de projetos. Por questões de privacidade, foram exportados para uma planilha compartilhada entre os pesquisadores que assinaram um acordo de confidencialidade.

3.2.2 Amostragem

O primeiro passo foi realizar um estudo exploratório com os testes disponibilizados, totalizando 898 casos de teste, todos escritos em inglês. A análise manual de todos esses casos seria demorada e onerosa. Para viabilizar o esforço, utilizou-se a fórmula de Cochran para calcular a amostra necessária, visando obter um nível de confiança de 80% com margem de erro de 5%. Assim, foram considerados 139 casos de teste na amostra.

3.2.3 Resultados

Após a análise realizada inicialmente, foram encontrados os seguintes *Smells*: *Ambiguous test, Conditional Test, Eager Action, Misplaced Action, Misplaced precondition, Misplaced Verification e Unverified Action*, descritos na Tabela 04. Este resultado, aliado ao resultado da análise dos testes manuais da Urna Eletrônica (Seção 3.1) serviu de base para o desenvolvimento de ferramentas para identificação e refatoração de *smells,* exploradas e aplicadas em diferentes contextos de testes de software (Soares et al. 2023 e Aranda et al. 2024). A tabela 5 apresenta as ocorrências dos *smells* encontrados nesta análise.

Tabela 5: Smells encontrados em testes de regressão de uma empresa global de telecomunicações

Smell	**Quantidade**
Eager Step	97
Ambiguous Test	44
Misplaced Result	39
Misplaced Step	21
Conditional Test Logic	9
Unverified Step	9
Misplaced Pre-Condition	6
Undefined Wait	1
Total	**226**

Fonte: Dados da pesquisa (2023)

3.3 DISCUSSÃO

É evidente que há um número significativo de testes e a falta de um padrão na escrita desses testes torna a análise ainda mais desafiadora. Testes manuais escritos em linguagem natural tendem a ser extensos, e aqueles que não são especialistas na área eleitoral podem encontrar dificuldades para entender e validar os testes devido à falta de clareza nos resultados esperados.

Como sugestão de melhorias, é recomendável buscar uma maior padronização na nomenclatura e uniformização na escrita dos testes, sejam eles manuais ou automatizados. Conforme apontado na literatura, isso facilita a manutenção e o entendimento dos testes pelos profissionais envolvidos no desenvolvimento e execução.

Quanto à empresa de telecomunicações, os testes do Plano de Experiências selecionado foram escritos por diferentes engenheiros de testes de software, e uma das dores (leia-se necessidades) da empresa, é a ausência de padronização na escrita e, em consequência, o entendimento correto dos passos. Esta suíte de testes foi selecionada por unanimidade pelos responsáveis pelo time de regressão da empresa por ser uma suíte grande, e com dificuldades de compreensão. Chama a atenção a presença de caracteres de idiomas asiáticos, o que dificulta a sua análise, e a falta de padronização de descrição de ações e de escrita. Por vezes, fluxos de tela são representados com ->, outras com >>. A análise automatizada por mecanismo de NLP (Processamento de Linguagem Natural, do inglês, Natural Language Processing), tal como descrito por Soares et al. (2023), encontra dificuldades no reconhecimento destes tipos de caracteres.

Estas análises contribuíram para o desenvolvimento dos artigos *Manual Tests Do Smell! Cataloging and Identifying Natural Language Test Smells,* disponível em: https://ieeexplore.ieee.org/xpl/conhome/10304838/proceeding e *A Catalog of Transformations to Remove Smells From Natural Language Tests,*disponível em: https://arxiv.org/abs/2404.16992.

4 VALIDAÇÃO DAS PROPOSTAS DE *SMELLS* E SEUS REFATORAMENTOS

Este capítulo tem como objetivo descrever as etapas percorridas para a validação das propostas de identificação e correção dos smells encontrados nos estudos exploratórios da Seção 3. Para validação das propostas de smells e seus refatoramentos, dois questionários foram necessários. O primeiro, usado para validar as propostas dos *smells* em testes escritos em linguagem natural (Seção 4.1) e o segundo para validar as propostas de refatorações (Seção 4.2).

4.1 VALIDAÇÃO DAS PROPOSTAS DE SMELLS

O primeiro questionário para Identificação de Smells em Testes Manuais de Software escritos em Linguagem Natural (no inglês, *Identifying Smells in Manual Software Testing written in Natural Language).*

4.1.1 Caracterização

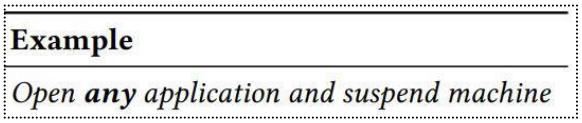
O questionário foi conduzido de forma completamente anônima e estruturado em três seções: (i) consentimento informado, (ii) dados demográficos e (iii) questões específicas. Na primeira seção, os participantes foram informados sobre os objetivos do estudo, os pesquisadores responsáveis e as opções de consentimento para participação. Na segunda seção, foram coletados dados sobre a ocupação principal dos participantes, categorizada como Academia ou Indústria, com o intuito de analisar a experiência profissional e identificar possíveis divergências entre esses setores. Adicionalmente, foi solicitado aos participantes que indicassem seu tempo de experiência com testes de software utilizando uma escala *Likert*, variando de 0 (menos de um ano) a 10 (dez ou mais anos). Também foi perguntado sobre a profissão específica, com opções como Analista de Teste de Software, Engenheiro de Teste de Software, Analista de Garantia de Qualidade ou Outros, com um campo para respostas discursivas. Ainda nesta seção, foi solicitado que os participantes indicassem o país de origem.

Na terceira seção, composta por oito perguntas, os participantes foram convidados a expressar seu grau de concordância utilizando uma escala *Likert* de seis pontos, com opções que variavam de “Concordo plenamente” a “Discordo plenamente”, incluindo “Indiferente” e um campo para respostas discursivas. Além disso, foi disponibilizado um espaço para comentários adicionais, caso os participantes desejassem compartilhar suas impressões.

As perguntas específicas (em inglês), com os seus respectivos exemplos, foram as seguintes:

Question 1: *When a test uses vague words, It may impact its comprehension (since the aim needs to be clarified) and execution (since multiple test executions are not comparable), making it non-deterministic.*

Figura 2: Exemplo da questão 01



Fonte: Autor (2023)

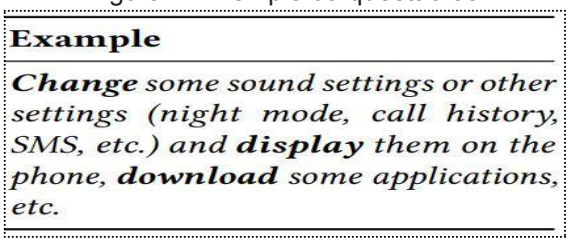
Question 2: *Conditions in actions or verification steps may hide implementation problems because, if not met, they skip the step execution.*

Figura 3: Exemplo da questão 02

Fonte: Autor (2023)

Question 3: *Steps that perform more than one action may hide implementation problems if any of such activities lack verification.*

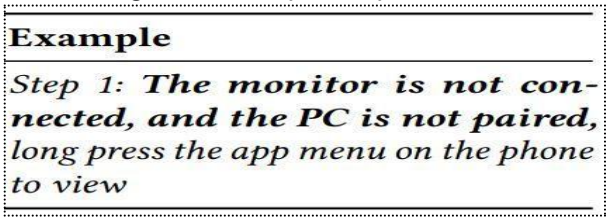
Figura 4: Exemplo da questão 03



Fonte: Autor (2023)

Question 4: *System Under Test (SUT) state declarations before any action performed are pre-conditions.*

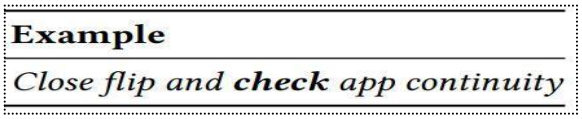
Figura 5: Exemplo da questão 04



Fonte: Autor (2023)

Question 5: *Writing verification together with action steps negatively impacts test maintenance and execution.*

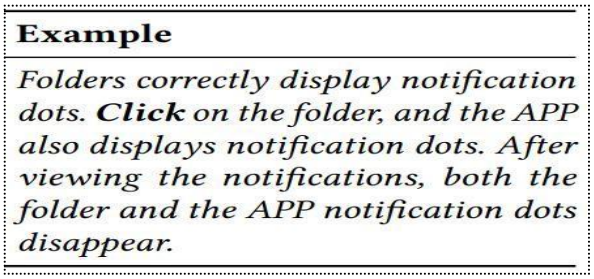
Figura 6: Exemplo da questão 05



Fonte: Autor (2023)

Question 6: *Writing action together with verification steps negatively impacts test maintenance and execution.*

Figura 7: Exemplo da questão 06



Fonte: Autor (2023)

Question 7: *The excessive use of unexplained, domain-specific terms and abbreviations presuming the tester's familiarity may cause difficulties in test comprehension and execution.*

Question 8: *Every action should have a verification step giving instructions on how the system should behave.*

4.1.2 Resultados

Obtivemos 24 respostas de profissionais atuantes na área, engenheiros de testes de software de uma empresa referência em telefonia móvel. Todas as respostas podem ser conferidas no repositório público (Souza, 2025). Na primeira parte sobre dados demográficos, pôde-se perceber que quando perguntado sobre a principal ocupação dos participantes, 83,3% responderam que são da Indústria enquanto 16,7% são da Academia. Esses números ajudam a validar a importância

de boas práticas na indústria, que são tão estudadas e analisadas pela academia, reforçando o interesse e a necessidade das duas áreas avançarem juntas.

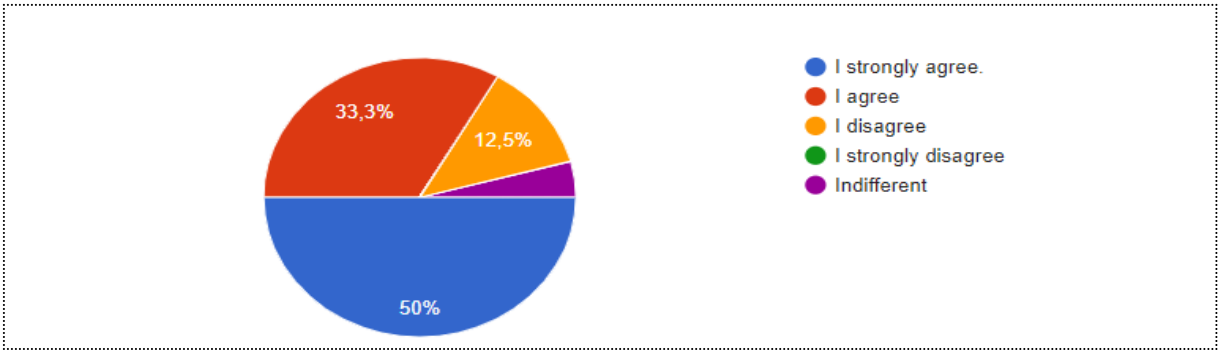
Com relação ao tempo de experiência na área de testes de software, até um ano, houve apenas um participante. Com um ano, três e dez anos tivemos o correspondente a 20,8%, ou seja, 05 participantes em cada período. Com dois, quatro, cinco e seis anos tivemos 8,3% em cada período, ou seja, 02 participantes em cada período. Essa variação de experiências foi fundamental para pontuar diferentes visões, no conhecimento tácito que cada um já tem na bagagem, além de enriquecer a discussão e sugestões de melhorias.

Entre as ocupações listadas, 54,2% foram Engenheiros de Testes de Software, 26,3% atuam como Analista de Teste de Software e/ou Analista de garantia de qualidade e os demais estão divididos entre Analista de software, Líder técnico, estagiário e estudante de residência na área de testes de software.

Seguindo para a seção das questões, começamos com a primeira: **Quando um teste utiliza palavras vagas, isso pode afetar a sua compreensão (uma vez que o objetivo precisa de ser clarificado) e a sua execução (uma vez que as múltiplas execuções do teste não são comparáveis), tornando-o não determinístico**. Com 83,3% de concordância, os engenheiros reafirmam a dificuldade em compreender um teste quando este não está escrito de maneira objetiva, nesse sentido, comentaram que “diferentes significados de palavras podem ser um problema na linguagem natural”. Outro comentário pontua que “a experiência pode melhorar a cobertura do teste, porém, para um testador iniciante não estão claras as maneiras de testar uma interrupção, e isso pode induzi-lo a repetir o mesmo procedimento/rotina ou tentar algumas maneiras diferentes de suspender o aplicativo” deixando margem para uma lacuna de cobertura ou contribuir para um falso positivo.

Em contrapartida, 12,5% discordam, e pontuaram que a “liberdade de abrir qualquer aplicação poderia facilitar ao testador a utilização de uma aplicação como escolha principal” durante um teste exploratório, por exemplo. Enquanto 4,2% se mostraram indiferentes e não opinaram, de acordo com a Figura 8, abaixo.

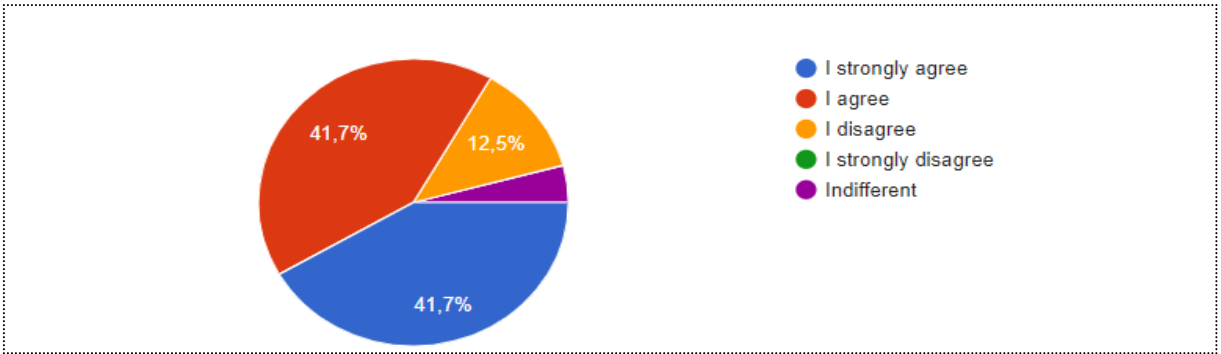
Figura 8 - Gráfico referente às respostas do questionário 01 - Questão 01



Fonte: Autor (2024)

Na segunda questão, **as condições nas ações ou etapas de verificação podem ocultar problemas de implementação porque, se não forem atendidas, pularão a execução da etapa**? Nesta pergunta, 83,4 % concordaram que as condições, uma vez que não são atendidas, podem não ser executadas, deixando uma lacuna na cobertura do caso de teste. Uma sugestão, dada por uma pessoa respondente, seria “em vez de condições, usar pré-condições no início dos casos de testes” além desta, outra sugestão seria “se uma etapa ou acessório não puder ser verificado, todo o teste não será bloqueado e o teste se tornará aplicável a diferentes tipos de produtos”, nesse ponto há o viés, também importante, de expertise do testador e/ou conhecimento sobre os testes, porém pode não ser garantia de cobertura eficiente. Por outro lado, com 16,6% entre discordância e indiferente, uma sugestão/opinião deixada foi “Se a etapa não for obrigatória, sua existência é questionável”, o que reforça a necessidade de deixar os casos de testes mais objetivos e direcionados a ausência de ambiguidades, além de atualizados e refatorados sempre que possível, conforme a Figura 9.

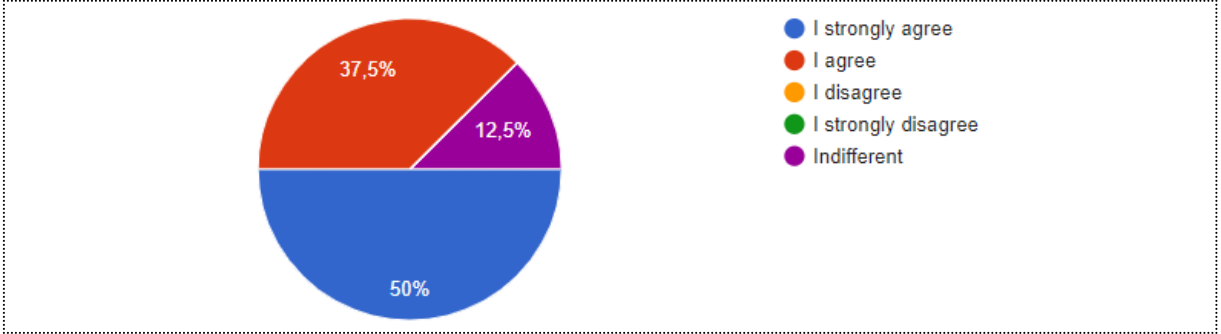
Figura 9 - Gráfico referente às respostas do questionário 01 - Questão 02



Fonte: Autor (2024)

A terceira questão, **As etapas que realizam mais de uma ação podem ocultar problemas de implementação se alguma dessas atividades não tiver verificação?** De acordo com a Figura 10, metade das respostas, ou seja, 50%, concordaram plenamente e outros 37,5% também concordaram com a pergunta, significando 87,5% e confirmam que o ideal é para cada ação, uma verificação, mesmo que isso implique em mais casos de testes. Um participante disse que “Não há garantia que o testador tenha verificado todas as configurações disponíveis” e com um olhar para a automatização de casos de testes, outro participante complementa que “Cada etapa com uma única ação facilita a automatização e a verificação de sua funcionalidade”. Nesta questão não houve discordância, um participante pontuou que “acredita que o exemplo apresentado seja outro caso extremo” e 3 participantes se sentiram indiferentes ao caso questionado.

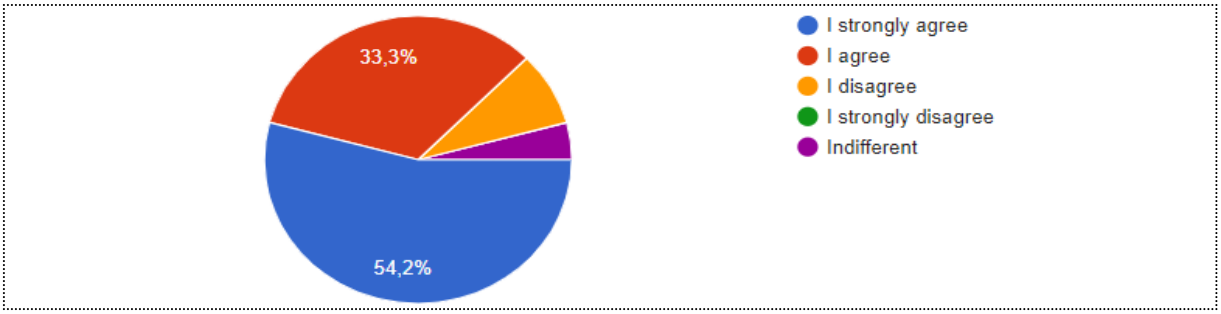
Figura 10 - Gráfico referente às respostas do questionário 01 - Questão 03



Fonte: Autor (2024)

Na quarta questão, **As declarações de estado do sistema em teste (SUT) antes de qualquer ação executada são pré-condições?** Com 87,5% os participantes concordaram e não quiseram opinar, apenas 8,3% discordaram e também não opinaram e 4,2% se sentiram indiferentes, visto na Figura 11.

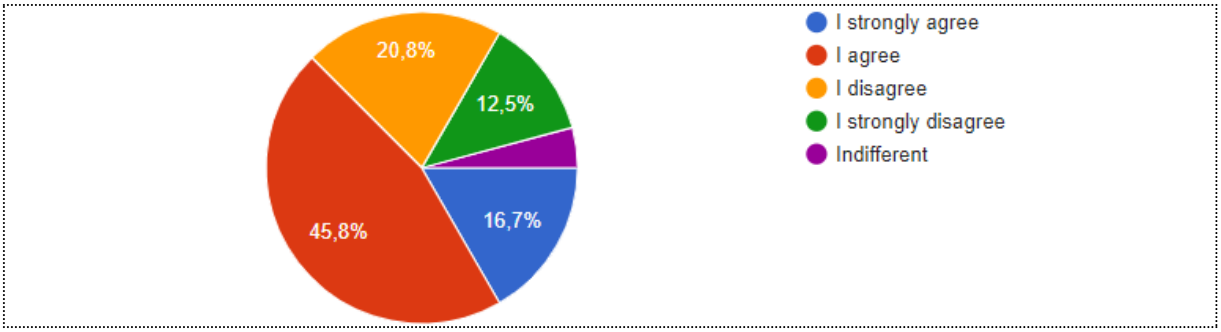
Figura 11 - Gráfico referente às respostas do questionário 01 - Questão 04



Fonte: Autor (2024)

A quinta questão, **Escrever a verificação junto com as etapas de ação impacta NEGATIVAMENTE a manutenção e execução do teste?** Essa questão trouxe mais divergências entre as respostas, mesmo totalizando 62,5% de concordância positiva, visto na Figura 12. Um participante pontuou que “concordo porque acho que é mais claro e organizado para fazer o teste em etapas separadas”. Enquanto que o somatório entre pessoas que discordam e discordam completamente chegou a 33,3%, sendo o número mais alto de discordância, trazendo uma reflexão para a necessidade de aprofundar as análises sobre os impactos da escrita de testes de maneira agrupada, “Quando a etapa de verificação está INTIMAMENTE relacionada à ação, não há problema”, destacou um participante. Além deste, outro reafirmou que “essas ações ajudam a evitar muitas etapas em um script e reduzem o esforço na manutenção de testes”, chamando atenção para quais técnicas para uma melhor escrita de casos de testes.

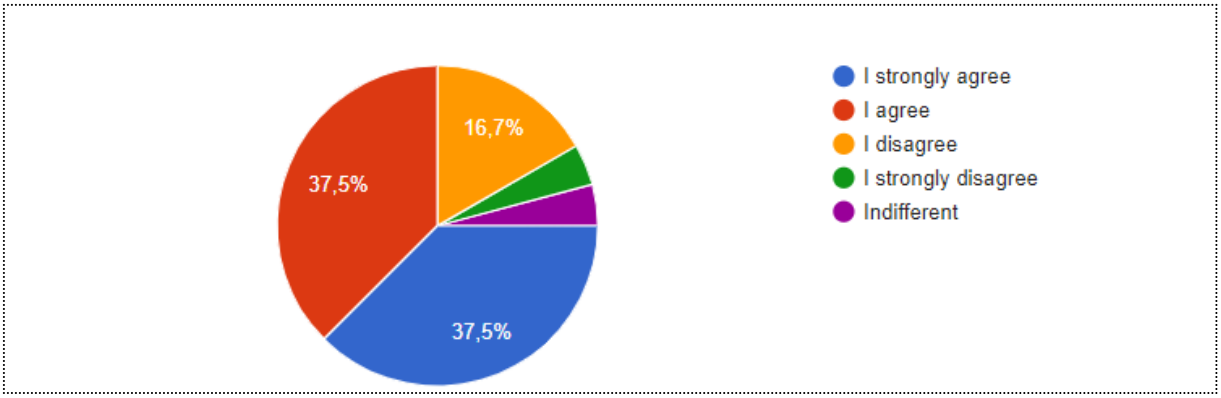
Figura 12 - Gráfico referente às respostas do questionário 01 - Questão 05



Fonte: Autor (2024)

A sexta questão, **Escrever ações junto com etapas de verificação impacta negativamente a manutenção e execução de testes?** Em relação à pergunta anterior, nesta, de acordo com a Figura 13, 75% concordam positivamente e pontuaram que “As etapas de verificação devem ocorrer no final dos casos de teste, pré condições no início e ações no meio”, outro destaca que “Se a ação mantém o passo válido como único, faz sentido ser escrito”, nesta opinião cabe atenção em relação às possibilidades no entendimento do caso de teste, se não há dúvidas ou ambiguidades. Em oposição, 20,9% discordam, mas não opinaram e 4,2% se mantiveram apáticos.

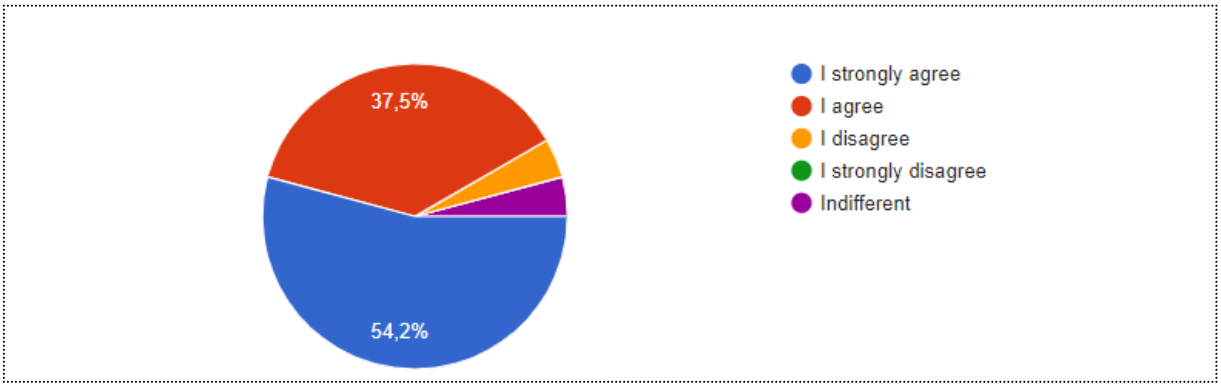
Figura 13 - Gráfico referente às respostas do questionário 01 - Questão 06



Fonte: Autor (2024)

A sétima e penúltima questão, **O uso excessivo de termos e abreviações inexplicáveis e específicos do domínio, presumindo a familiaridade do testador, pode causar dificuldades na compreensão e execução do teste?** Expressivamente 91,7% concordaram positivamente, visto na Figura 14, onde trouxeram importantes contribuições para complementar a importância do detalhamento das informações necessárias para o entendimento do caso de teste. Uma das opiniões pontua a importância da “clareza e simplicidade” e “deve ser escrito com palavras específicas”, outra diz que “é melhor escrever a frase completa e colocar abreviatura entre parênteses” pois “se o testador não estiver acostumado com as abreviaturas e termos da empresa, pode ser difícil”. Além dessas, outro destaque relevante é que “faz parte da responsabilidade do testador, conhecer o vocabulário necessário, mas também é responsabilidade do designer do teste torná-lo o mais simples possível”.

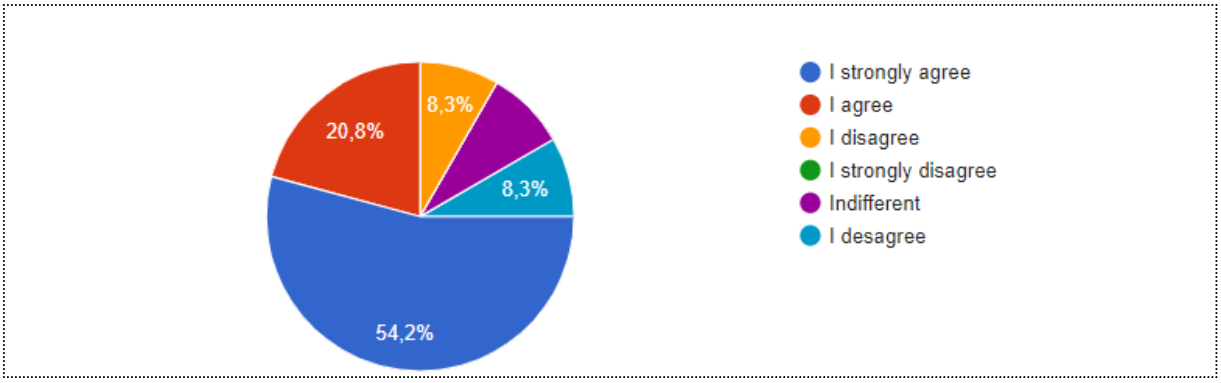
Figura 14 - Gráfico referente às respostas do questionário 01 - Questão 07



Fonte: Autor (2024)

A oitava e última questão, **Toda ação deve ter uma etapa de verificação dando instruções sobre como o sistema deve se comportar?** Com concordância de 75%, os engenheiros confirmam o interesse em ter disponível a confirmação das ações a serem executadas, de acordo com a Figura 15. Na tentativa de não deixar o leitor cansado, uma sugestão deixada foi que “em algumas situações o resultado esperado é demasiado óbvio e pode ser dispensado”. Nesse quesito, a expertise do engenheiro de teste que escreve o teste pode fazer uma diferença positiva quanto a passos e resultados esperados, uma vez que não deixa equívocos na interpretação. Uma outra sugestão deixada foi que “ nem toda ação precisaria de instruções, pois numa sequência de ações gera um resultado relevante a ser verificado, mas ao final do caso de teste, deverá haver uma verificação do resultado final”. Uma sugestão salutar, que cabe experimentos futuros para descobrir qual a melhor estratégia para deixar o caso de teste coeso e menos cansativo.

Figura 15 - Gráfico referente às respostas do questionário 01 - Questão 08



Fonte: Autor (2024)

4.2 VALIDAÇÃO DOS REFATORAMENTOS

O segundo questionário teve como objetivo validar a Refatoração de Smells em testes manuais de linguagem natural (do Inglês, *Refactoring Smells in Manual Natural Language Tests*).

4.2.1 Caracterização

O questionário foi conduzido de forma completamente anônima e estruturado em três seções: (i) consentimento informado, (ii) dados demográficos e (iii) questões específicas. Na primeira sessão, os participantes foram informados sobre os objetivos do estudo, os pesquisadores responsáveis e as opções de consentimento para participação. Na segunda seção, foram coletados dados sobre a ocupação principal dos participantes, categorizada como Academia ou Indústria, com o intuito de analisar a experiência profissional e identificar possíveis divergências entre esses setores. Adicionalmente, foi solicitado aos participantes que indicassem seu tempo de experiência com testes de software utilizando uma escala Likert, variando de 0 (menos de um ano) a 10 (dez ou mais anos). Também foi perguntado sobre a profissão específica, com opções como Analista de Teste de Software, Engenheiro de Teste de Software, Analista de Garantia de Qualidade ou Outros, com um campo para respostas discursivas. Ainda nesta seção, foi solicitado que os participantes indicassem o país de origem.

A terceira seção do questionário apresentou uma estrutura diferenciada, composta por sete questões, cada uma abordando um *smell* específico, incluindo sua definição, os problemas potenciais associados, métodos de identificação e exemplos práticos. Os participantes foram convidados a expressar seu grau de concordância utilizando uma escala *Likert* de seis pontos, com opções que variavam de “Concordo plenamente” a “Discordo plenamente”, incluindo “Indiferente” e um campo para respostas discursivas.

Q1: Sobre o Smell Unverified Action (Ação não verificada)**,** concorda que no exemplo abaixo, Quadro 2, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 2: Unverified Action - Exemplo

Unverified Action			
Before		**After**	
Action Steps	**Verifications**	**Action Steps**	**Verifications**
Open "Notes"	Notes opens and an icon appears in the indicator bar	Open "Notes"	Notes opens and an icon appears in the indicator bar
Type some text	Text appears	Type some text	Text appears
Click "X"		Click "X"	[FILL VERIFICATION]
Click the indicator icon for notes	All your text is still there	Click the indicator icon for notes	All your text is still there
Restart your computer, Then open notes	Your text is still there	Restart your computer	[FILL VERIFICATION]
		Then open notes	Your text is still there

Fonte: Autor (2024)

a) Definição: Passos de ação que não possuem passos de verificação correspondentes.

b) Problema: A ausência de passos de verificação afeta negativamente a execução e correção dos testes, uma vez que não há instruções sobre como o sistema se deve comportar, deixando espaço para a interpretação dos testadores.

c) Identificação: Passos de ação sem passos de verificação correspondentes.

Q2: Sobre o *Smell Eager Action* (Ação ansiosa), concorda que no exemplo abaixo, Quadro 3, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 3 - Eager Action - Exemplo

Eager Action			
Before		**After**	
Action Steps	**Verifications**	**Action Steps**	**Verifications**
Start Nautilus, maximize its window. Start Firefox, maximize its window.	Launcher should be visible	Start Nautilus	[FILL VERIFICATION]
		maximize its window	[FILL VERIFICATION]
		Start Firefox	[FILL VERIFICATION]
		maximize its window.	Launcher should be visible
Minimize the Firefox window	The Firefox icon's border on the Launcher should animate slightly	Minimize the Firefox window	The Firefox icon's border on the Launcher should animate slightly

Fonte: Autor (2024)

a) Definição: Etapas de ação única que agrupam várias ações.

b) Problema: Este *test smell* pode esconder problemas de implementação quando alguma ação carece de verificação, afetando negativamente a eficácia do teste.

c) Identificação: Os verbos imperativos representam ações.

Q3: Sobre o *Misplaced Action* (Ação ansiosa), concorda que no exemplo abaixo, Quadro 4, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 4 - Misplaced Action - Exemplo

Misplaced Action			
Before		**After**	
Action Steps	**Verifications**	**Action Steps**	**Verifications**
Tap super to open the dash, click on a lens	Make a search, open the filters and choose some options	Tap super to open the dash	[FILL VERIFICATION]
Close the dash with super		click on a lens	[FILL VERIFICATION]
		Make a search	[FILL VERIFICATION]
		open the filters	[FILL VERIFICATION]
		choose some options	[FILL VERIFICATION]
		Close the dash with super	[FILL VERIFICATION]
Reopen the dash:	The main dash home shouldn't contain previous search	Reopen the dash:	The main dash home shouldn't contain previous search
Open on the previous lens	You should see the same search results, with the same filter enabled	Open on the previous lens	You should see the same search results, with the same filter enabled
Click on other lenses:		Click on other lenses:	[FILL VERIFICATION]

Fonte: Autor (2024)

a) Definição: Indicativo de um teste estruturalmente malformado, o *smell* de ação mal colocada surge quando passos de ação são escritos como resultados.

b) Problema: Impacta negativamente a manutenibilidade do teste, uma vez que a estrutura do teste não é consistente.

c) Identificação: Os verbos imperativos, excluindo os verbos de verificação, estão presentes nas etapas de verificação.

Q4: Sobre o *Smell Ambiguous Test* (Teste Ambíguo), concorda que no exemplo abaixo, Quadro 5, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 5 - Ambiguous test - Exemplo

Ambiguous Test

Before

Action Steps	Verifications
Move the mouse to the very bottom of the screen	The launcher panel pops up
Click the Application Finder icon and run any application from it	Application Finder opens and the application opens as expected

After

[FILL IN MORE INFORMATION ABOUT (7)]	
Action Steps	**Verifications**
Move the mouse to the very bottom of the screen	The launcher pops up
Click the Application Finder icon	[FILL_VERIFICATION]
run application A(7) from it	Application Finder opens and the application (7) opens as expected

Fonte: Autor (2024)

a) Definição: Este *smell* indica um "teste subespecificado que deixa espaço para interpretação".

b) Problema: Tem um impacto negativo na compreensão e execução do teste, uma vez que o objetivo tem de ser clarificado e as múltiplas execuções do teste não são comparáveis.

c) Identificação: A regra de deteção original era a ocorrência de qualquer palavra de uma lista fixa de "palavras vagas".

Q5: Sobre o *Smell Misplaced Verification* (Verificação incorreta), concorda que no exemplo abaixo, Quadro 6, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 6 - Misplaced Verification - Exemplo

Misplaced Verification			
Before		**After**	
Action Steps	**Verifications**	**Action Steps**	**Verifications**
Open the Terminal pressing CTRL+ALT+T type "lsb_release -r" without the quotes		Open the Terminal pressing CTRL+ALT+T type "lsb_release -r" without the quotes	* Verify that FAMILY upgraded to the latest release * You should verify the system upgraded correctly to the latest FAMILY release testing that the default applications installed work correctly
Verify that FAMILY upgraded to the latest release	You should verify the system upgraded correctly to the latest FAMILY release testing that the default applications installed work correctly		

Fonte: Autor (2024)

a) Definição: Outro indicativo de testes estruturalmente malformados, esse cheiro surge quando passos de verificação são escritos como passos de ação.

b) Problema: Impacta negativamente a manutenibilidade do teste, uma vez que a estrutura do teste não é consistente.

c) Identificação: Sentenças contendo verbos de verificação escritos como ou junto com passos de ação.

Q6: Sobre o *Smell Misplaced Precondition* (Precondição mal colocada), concorda que no exemplo abaixo, Quadro 7, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 7 - Misplaced Precondition - Exemplo

Misplaced Precondition			
Before		**After**	
Action Steps	**Verifications**	**Preconditions: *Make sure auto-hide is enabled.**	
		Action Steps	**Verifications**
Make sure auto-hide is enabled.		Move the mouse on the edge and: - continue pushing to the left (as if you want to put the mouse offscreen) until the launcher reveals	The launcher should appear
Move the mouse on the edge and: - continue pushing to the left (as if you want to put the mouse offscreen) until the launcher reveals.	The launcher should appear		
Put the mouse outside of the launcher area	The launcher should disappear again after around a second	Put the mouse outside of the launcher area	The launcher should disappear again after around a second

Fonte: Autor (2024)

a) Definição: Também um indicativo de testes estruturalmente mal formados, aqui, as pré-condições são escritas como passos de ação.

b) Problema: Dificuldades na correção do teste, uma vez que a colocação incorrecta de condições prévias pode influenciar o testador a reportar falha no teste caso uma condição prévia não seja atendida.

c) Identificação: Quando o primeiro passo de ação declara o estado do SUT.

Q7: Sobre o *Smell Conditional Test Logic* (Precondição mal colocada), concorda que no exemplo abaixo, Quadro 8, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?

Quadro 8 - Conditional Test Logic - Exemplo

<table>
<tr><th colspan="4">Conditional Test Logic</th></tr>
<tr><th colspan="2" rowspan="2">Before</th><th colspan="2">After (CASE TRUE)</th></tr>
<tr><th colspan="2">Ensure the device is a USB3.0 storage device and you have a USB3.0 port (1)</th></tr>
<tr><th>Action Steps</th><th>Verifications</th><th>Action Steps</th><th>Verifications</th></tr>
<tr><td rowspan="4">Plug a USB device in and attempt to use it</td><td rowspan="4">*The device is correctly recognized
*The software normally used with the device functions normally
*The device behaves as expected
*The USB device works in every port
*You are able to disconnect and re-connect the USB device correctly without errors</td><td>Plug a USB device in</td><td>[FILL VERIFICATION]</td></tr>
<tr><td>attempt to use it</td><td>*The device is correctly recognized
*The software normally used with the device functions normally
*The device behaves as expected
*The USB device works in every port
*You are able to disconnect and re-connect the USB device correctly without errors</td></tr>
<tr><td>transfer a large file between the two (1)</td><td>The transfer is above USB2.0 speed</td></tr>
<tr><td>Repeat for each USB device you have</td><td>[FILL VERIFICATION]</td></tr>
<tr><td rowspan="4">If the device is a USB3.0 storage device and you have a USB3.0 port, transfer a large file between the two</td><td rowspan="4">The transfer is above USB2.0 speed</td><th colspan="2">After (CASE FALSE)</th></tr>
<tr><th>Action Steps</th><th>Verifications</th></tr>
<tr><td>Plug a USB device in</td><td>[FILL VERIFICATION]</td></tr>
<tr><td>attempt to use it</td><td>*The device is correctly recognized
*The software normally used with the device functions normally
*The device behaves as expected
*The USB device works in every port
*You are able to disconnect and re-connect the USB device correctly without errors</td></tr>
<tr><td>Repeat for each USB device you have</td><td></td></tr>
</table>

Fonte: Autor (2024)

a) Definição: Testes contendo lógica condicional redigida em linguagem natural.

b) Problema: O Teste Condicional torna os testes muito complexos e difíceis de manter, afetando negativamente a compreensão e a correção do teste, uma vez que é difícil entender a intenção, e testes complexos são mais propensos a ter erros.

c) Identificação: Propõe uma lista fixa de palavras para a sua deteção.

4.2.2 Resultados

No segundo questionário, Refatorando Smells em Testes Manuais em Linguagem Natural, tivemos 19 respostas de profissionais atuantes na área, engenheiros de testes de software de uma empresa referência em telefonia móvel a mais de 50 anos no mercado global. Na primeira parte sobre dados demográficos, pôde-se perceber que quando perguntado sobre a principal ocupação dos participantes, 63,2% responderam que são da Indústria enquanto 36,8% são da Academia. Esses números ajudam a validar a importância de boas práticas na indústria, que são tão estudadas e analisadas pela academia, reforçando o interesse e a necessidade das duas áreas avançarem juntas.

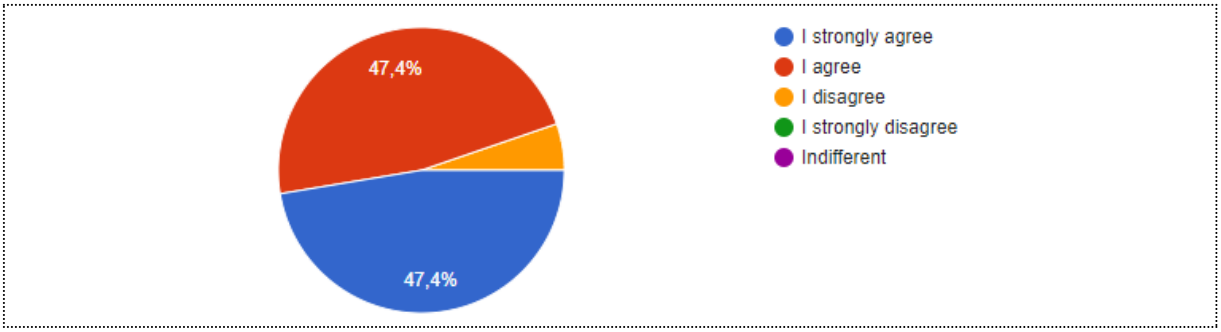
Com relação ao tempo de experiência na área de testes de software, até um ano, houveram três participantes, representando 15,8% do público. Com três anos, quatro participantes (21,1%). Com quatro anos, cinco participantes (26,3%) e o somatório de quem tem acima de oito anos, corresponde a quatro participantes (21,1%). Essa variação de experiências contribuiu diretamente para a amplitude dos resultados obtidos, facilitando análises e trazendo *insights* para trabalhos futuros.

Entre as ocupações listadas, 31,6% foram Engenheiros de Testes de Software, 26,3% atuam como Analista de Teste de Software e/ou Analista de garantia de qualidade e os demais estão divididos entre Analista de software, Líder técnico, estagiário e estudante de residência na área de testes de software.

Iniciando a seção de questões, iniciamos com o *Smell* Unverified Action (Ação não verificada), perguntamos se o participante **concorda que no exemplo o problema identificado foi resolvido de acordo com a definição?**

De acordo com a Figura 16, tivemos 94,8% de concordância. Um participante disse que “Seguindo a lógica de que toda ação tem uma reação, cada etapa do teste deve ter um resultado esperado, pois, de certa forma, é um algoritmo a ser executado por uma pessoa ou uma máquina”, o que confirma a boa prática de passos de testes pequenos e com resultados, mesmo que óbvio, bem definido no caso de teste.

Figura 16 - Gráfico referente às respostas do questionário 02 - Questão 01

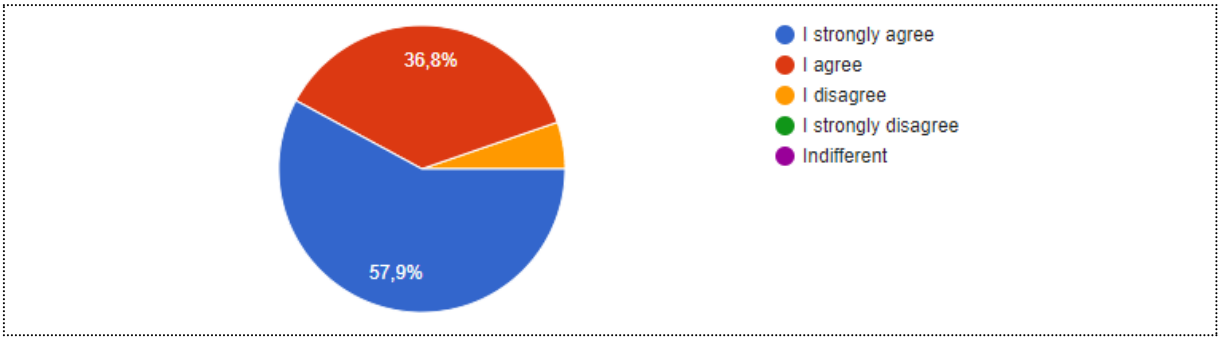


Fonte: Autor (2024)

Na segunda questão, Sobre o *Smell Eager Action* (Ação ansiosa), **concorda que no exemplo, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?**

Concordaram plenamente 57,9% e outros 36,8% também concordam com a questão, conforme Figura 17. Complementaram que: “Nesse caso, algumas etapas poderiam ser incluídas em uma coluna 'configuração de teste', para que, antes que a pessoa/máquina execute as etapas, ela garanta que tenha os recursos necessários para realizar o teste.” Por outro lado, um participante não concordou, e acrescentou não ser necessária uma verificação.

Figura 17 - Gráfico referente às respostas do questionário 02 - Questão 02

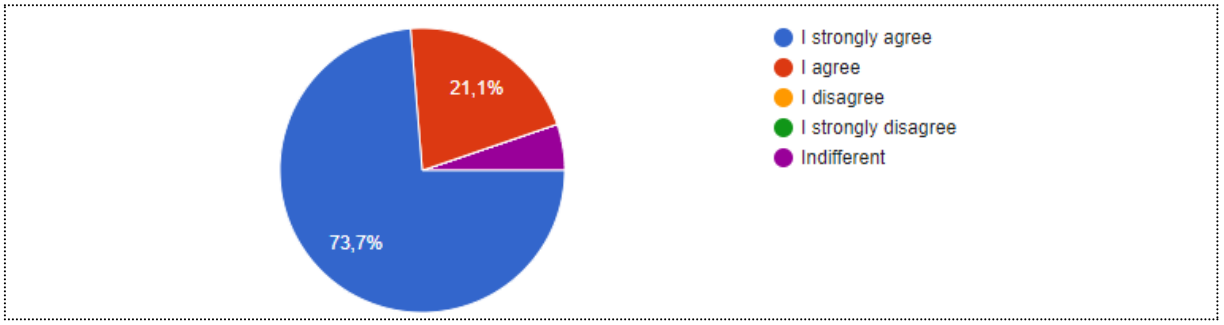


Fonte: Autor (2024)

A terceira questão, Sobre o *Smell Misplaced Action* (Ação ansiosa), **concorda que no exemplo, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?**

Positivamente 18 participantes concordam com a questão, totalizando 94,8%, de acordo com a Figura 18. E complementaram que “esse tipo de teste também pode levar o testador a não saber qual ação deve ser executada. Segundo o participante, “Já vi testes em que o fluxo "correto" de ações começa em Action steps e continua em Verification fields.”

Figura 18 - Gráfico referente às respostas do questionário 02 - Questão 03

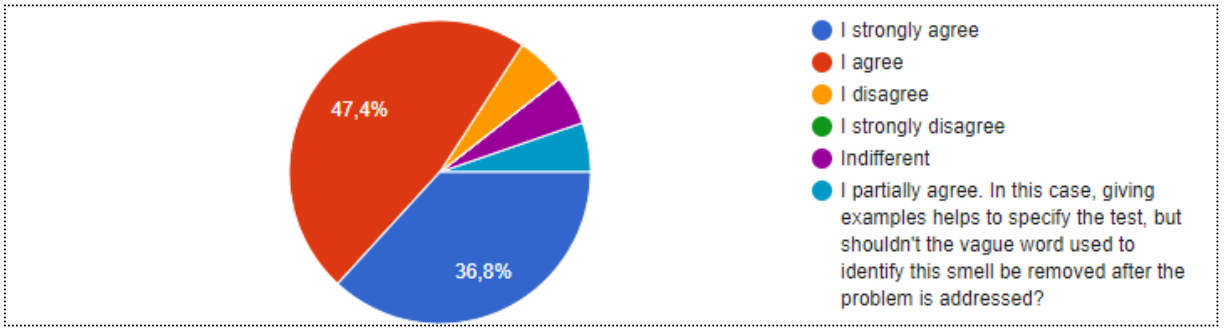


Fonte: Autor (2024)

Seguindo para a quarta questão, Sobre o *Smell* Ambiguous Test (Teste Ambíguo), **concorda que no exemplo, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?**

Concordaram plenamente 36,8% e outros 47,4% também concordam com a pergunta, visto na Figura 19. Um participante destacou que concorda parcialmente e complementa que: “Neste caso, dar exemplos ajuda a especificar o teste, mas a palavra vaga utilizada para identificar este smell não deveria ser removida depois de o problema ser resolvido?”

Figura 19 - Gráfico referente às respostas do questionário 02 - Questão 04

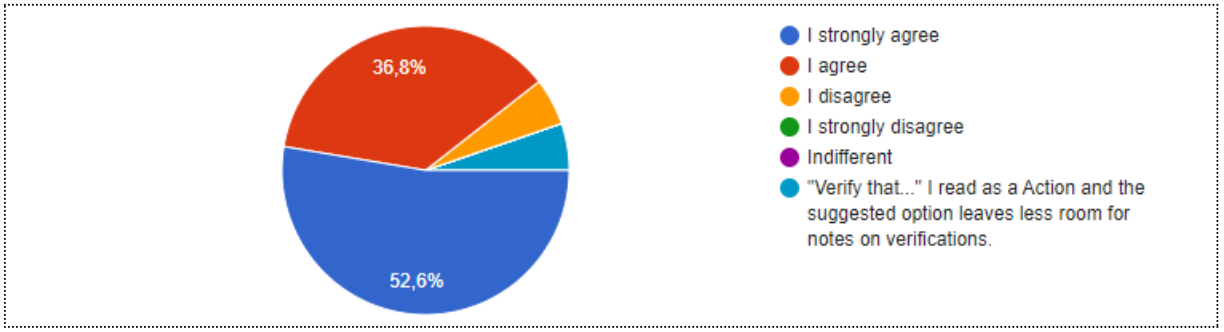


Fonte: Autor (2024)

A quinta questão, Sobre o *Smell* Misplaced Verification (Verificação incorreta), **concorda que, no exemplo, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?**

De acordo com a Figura 20, concordaram plenamente 52,6%, concordaram 36,8%, e houve discordância de 5,3%.

Figura 20 - Gráfico referente às respostas do questionário 02 - Questão 05



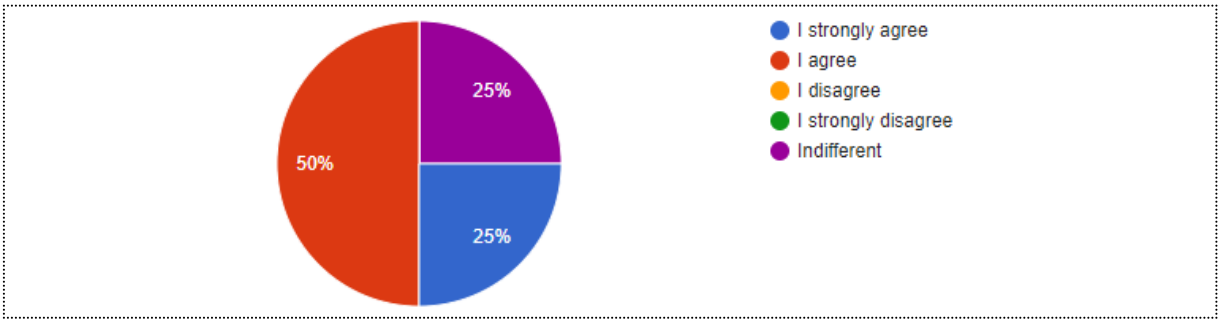
Fonte: Autor (2024)

Na sexta questão, Sobre o *Smell* Misplaced Precondition (Precondição mal colocada), **concorda que, no exemplo, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?**

De maneira indiferente foi a resposta de 25% dos participantes, e o restante, 75% concordaram com a questão, Figura 21. Sugeriram que: “A pré-condição "*Make sure auto-hide is enable*" poderia ser escrita como um comentário para avisar o testador sobre um requisito para executar o caso de teste. Além disso, se houvesse

uma ação relacionada a ela, ela poderia ser escrita como "Enable auto-hide" caso o software não a tenha habilitado por padrão”.

Figura 21 - Gráfico referente às respostas do questionário 02 - Questão 06

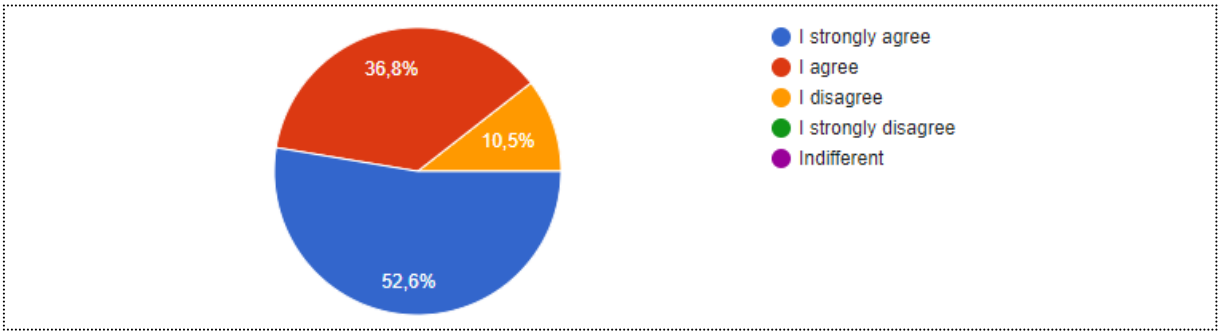


Fonte: Autor (2024)

A sétima e última questão, Sobre o *Smell Conditional Test Logic* (Precondição mal colocada), **concorda que, no exemplo, o problema identificado foi resolvido (pode ainda ser necessária a intervenção humana) de acordo com a definição?**

Plenamente houve concordância de 52,6% adicionando 36,8% que também concordam com a questão, porém um número maior discorda, 10,5%, de acordo com a Figura 22 e complementaram que: “Concordo com a descrição da definição, problema e identificação. No entanto, ao passar para o exemplo prático, acredito que talvez este exemplo específico não esteja exemplificando claramente a descrição. O exemplo fornecido à esquerda tem um problema de ambiguidade, mas a solução à direita não é necessariamente a melhor solução.” Nesse sentido, exige mais pesquisas sobre este *smell* para tentar tornar a ferramenta mais inteligente e minimizar essa impressão do participante.

Figura 22 - Gráfico referente às respostas do questionário 02 - Questão 07



Fonte: Autor (2024)

4.3 DISCUSSÃO

Ao analisar os testes em dois diferentes ambientes, algumas sugestões de boas práticas podem ser descritas com o objetivo de melhorar os processos existentes. No contexto do sistema da urna eletrônica, identificação de pontos de melhorias no desenvolvimento e padronização do código e escrita dos testes manuais, tornando-os mais objetivos e autoexplicativos, melhorando a legibilidade e manutenção dos testes. Estes atributos são desejáveis para a manutenção de sua robustez a curto e longo prazo, bem como para minimizar a inserção de falhas.

No contexto dos testes manuais para dispositivos móveis, após a análise e resultados encontrados, algumas sugestões podem contribuir para melhorias internas nas execuções dos testes, por exemplo:

- Definir o que deve ocorrer em cada condição lógica, evitando que o responsável pela execução tenha dúvidas, especialmente se o resultado divergir do cenário da primeira condição.
- Descrever detalhadamente cada ação e o resultado esperado, mesmo nos testes mais simples, garantindo consistência e evitando mal-entendidos durante a execução.
- Focar cada caso de teste em um único comportamento ou funcionalidade, evitando múltiplas verificações em um único passo, pois isso pode tornar os testes confusos e difíceis de manter.

Além disso, é fundamental definir as pré-condições, verificações e resultados esperados, de modo que não restem dúvidas para quem executa os testes. Também é necessário especificar a configuração inicial, os passos a serem seguidos e os resultados esperados em cada etapa.

Assim como no código, os testes devem ser revisados e atualizados regularmente para refletir as mudanças no software. Testes desatualizados ou irrelevantes podem introduzir “smells”, levando a falsos negativos ou positivos.

Em cada área, há termos e siglas específicas; portanto, é útil explicar abreviações ou criar um glossário. Isso torna o objetivo do teste mais claro, mesmo para quem não é da área.

Durante a escrita de casos de teste, é essencial compreender o objetivo de cada teste, suas etapas e os resultados esperados. Para isso, é importante manter a simplicidade dos casos, definir o ambiente necessário antes da execução e garantir que, para cada ação, haja um resultado esperado.

De maneira geral, tomando como público-alvo os profissionais responsáveis pela escrita e execução dos testes manuais de determinado software, apresentamos, nas próximas seções, as diretrizes elaboradas contendo título, motivação, ações e exemplos dos resultados esperados. Ao fim destas diretrizes, pretende-se garantir que os testes manuais contribuam de maneira significativa para a qualidade geral do software.

Conforme destacado por Hauptmann (2016), as descrições de testes manuais frequentemente utilizam linguagem natural e são, em geral, de baixa qualidade, sendo redigidas sem a adoção das melhores práticas de engenharia de software. Nesse contexto, enfatizamos que a identificação e a refatoração de testes constituem atividades onerosas, porém necessárias ao longo de todo o processo de desenvolvimento e execução de testes de software.

Os resultados deste estudo indicam um interesse significativo da indústria em pesquisas que contribuam para a compreensão da importância de padrões e *layouts* que promovam a redação de casos de teste de maneira objetiva, clara e útil. Além disso, tais padrões são essenciais não apenas para a criação, mas também para a manutenção eficaz dos casos de teste.

A adoção de boas práticas na escrita de testes manuais pode melhorar substancialmente a qualidade dos testes, tornando-os mais robustos e eficazes na detecção de defeitos. A padronização das descrições de testes, aliada à utilização de *layouts* consistentes, facilita a compreensão e a execução dos testes, reduzindo ambiguidades e aumentando a confiabilidade dos resultados.

Portanto, a indústria necessita continuar investindo em estudos que explorem e promovam a implementação de melhores práticas na redação de testes manuais. Isso não apenas aprimora a qualidade dos testes, mas também contribui para a eficiência e a eficácia do processo de desenvolvimento de software como um todo.

4.4 AMEAÇAS À VALIDADE

A validade de um estudo da opinião de profissionais de teste de software sobre propostas de *test smells* e seus refatoramentos pode ser comprometida por diversas ameaças:

Nas ameaças à validade interna, a amostra de profissionais de teste de software pode não ser representativa da população geral. Como os participantes foram selecionados apenas da empresa de telecomunicações, isso pode introduzir vieses que afetam a generalização dos resultados. Ainda, as respostas dos participantes podem ser influenciadas pelo contexto específico de suas experiências de trabalho, como o tipo de projetos em que estão envolvidos ou as práticas de teste adotadas na empresa. Mitigamos esta ameaça com sua experiência de anos no domínio de testes de software, apresentada nas seções de resultados.

Quanto à validade externa, os resultados obtidos podem não ser generalizáveis para todos os profissionais de teste de software, especialmente aqueles que trabalham em diferentes domínios ou com diferentes tecnologias. Neste quesito, a diversidade da amostra é crucial para garantir a aplicabilidade ampla dos achados. No entanto, como o objetivo desta dissertação é apresentar diretrizes adequadas ao público selecionado, a incapacidade dos dados serem generalizáveis não representam problema para o estudo.

Quanto à validade do construto, a compreensão e interpretação de *test smells* podem variar entre os participantes. Se os conceitos não forem definidos e compreendidos de maneira uniforme, isso pode afetar a consistência das respostas. Minimizamos esta ameaça fornecendo definições claras de *test smells* e refatoramentos para garantir uma compreensão uniforme entre os participantes, que se demonstrou efetiva dados os resultados majoritariamente positivos.

5 DIRETRIZES PARA PREVENÇÃO DE *TEST SMELLS* EM TESTES MANUAIS DE SOFTWARE

Este Capítulo tem como objetivo fornecer diretrizes práticas para a prevenção de *test smells* em testes manuais de software. Serão abordadas boas práticas para a criação, execução e manutenção de testes manuais, considerando os *smells* existentes, destacando os sinais de alerta que devem ser observados e as estratégias para mitigá-los.

5.1 DIRETRIZ 1: PREVENÇÃO DE AMBIGUIDADES

5.1.1 Motivação

As ambiguidades em testes manuais de software deixam espaço para interpretação por quem executa o teste (Hauptmann, 2013). Caso uma ambiguidade esteja colocada numa ação, esta pode causar dificuldades ao testador quanto ao que realizar no sistema sob teste (SUT). Caso a ambiguidade esteja presente numa verificação, o testador poderá ter dificuldades quanto ao que deve ser verificado na resposta do SUT.

5.1.2 Recomendações

1. Não usar artigos e pronomes indefinidos. Evitar termos vagos como “aproximadamente”, “talvez”, “possivelmente” ou qualquer outra expressão que possa ser interpretada de diferentes maneiras.

Exemplos:

1. “Abra **qualquer** aplicação e suspenda o computador”
   Como resolver: *Definir qual a aplicação (incluindo o número da versão) deve ser aberta.*
2. “Em ‘Gravar alterações’, verifique se **tudo** está correto e selecione SIM”.
   Como resolver: *Definir o que é esperado como correto.*
3. “Verifique se **todos** os apps estão dentro da pasta [nome da pasta]”.

Como resolver: *Listar quais aplicativos são esperados dentro da pasta [nome da pasta].*

4. "Insira **um** valor alto no campo 'Quantidade' e verifique o resultado."
   Como resolver: *"Insira o valor '5000' no campo 'Quantidade' e verifique se o sistema exibe a mensagem 'Limite máximo excedido'."*
5. "Certifique-se de que a temperatura está dentro de **uma** faixa aceitável."
   Como resolver: *"Certifique-se de que a temperatura está entre 20°C e 25°C durante a execução do teste."*
6. "Certifique-se de que **o arquivo** seja salvo."
   Como resolver: *"O arquivo 'documento.docx' deve ser salvo no diretório padrão 'C:/Documentos' com a data e hora atual visíveis no nome do arquivo."*
7. “Verifique que o botão ‘Enviar’ funciona **corretamente**.”
   Como resolver: *"O sistema deve exibir a mensagem 'Dados enviados com sucesso' e redirecionar o usuário para a página 'Resumo do Pedido'."*
8. "Confirme que **o** item foi adicionado ao carrinho."
   Como resolver: *"O item aparece no resumo do carrinho com as informações de nome, quantidade e preço."*
9. "Verifique se **a mensagem** aparece na página."
   Como resolver: *"Verifique se a mensagem 'Cadastro realizado com sucesso' aparece no topo da página 'Registro de Usuário'."*

2. Substitua advérbios e adjetivos por critérios quantificáveis.

Exemplos:

1. Certifique-se de que o processo de carregamento é **rápido.**
   Como resolver: *"O processo de carregamento do relatório é concluído em até 5 segundos."*
2. "O sistema deve apresentar a **melhor** performance durante o teste."
   Como resolver: *"O sistema deve processar 1000 requisições simultâneas com um tempo de resposta inferior a 200ms para cada requisição."*

5.2 DIRETRIZ 2: PREVENÇÃO DE TERMOS ESPECÍFICOS DO DOMÍNIO

5.2.1 Motivação

A ausência de definição de termos específicos do domínio ou siglas não definidas pode causar insegurança na execução do caso de teste.

5.2.2 Recomendações

1. Explicar termos específicos do domínio. Definir siglas, valores, tamanhos e/ou qualquer informação referente ao domínio do conhecimento necessário para a execução do teste, tal como os resultados esperados para cada ação

Exemplo:

1. Pré-condição: Conecte o dispositivo à internet
2. Ação: Acesse configurações
3. Verificação: O menu de configurações foi acessado com sucesso
4. Ação: Atualize os serviços de **GPS**
5. Verificação: Os serviços foram atualizados

Como resolver:

1. *Pré-condição: Conecte o dispositivo a internet*
2. *Ação: Acesse configurações*
3. *Verificação: O menu de configurações foi acessado com sucesso*
4. *Ação: Atualize os serviços de GPS **(Google Play Store)***
5. *Verificação: Os serviços foram atualizados*

5.3 DIRETRIZ 3: PREVENÇÃO DE PRÉ-CONDIÇÕES DESLOCADAS

5.3.1 Motivação

As diretrizes para a prevenção de *smells* relacionados à pré-condições visam garantir que as pré-condições respeitem a estrutura canônica de um teste (Seção 2.1.1.2) para suportar a execução do teste.

5.3.2 Recomendações

1. Definir as Pré-condições no início do caso de teste antes de qualquer ação ser executada. Isso garante que o ambiente e os estados necessários estejam prontos antes do início do teste.

Exemplo:

1. Ação: "Acesse o relatório de vendas e selecione o período desejado."

Como resolver:

Pré-condição 01: Certifique-se de que o usuário está autenticado.

Pré-condição 02: Certifique-se de que o usuário tem permissão para acessar relatórios.

Ação: Acesse o relatório de vendas e selecione o período desejado.

Exemplo:

1. "Realize o login no sistema para acessar o painel administrativo”.

Como resolver:

Pré-condição: Certifique-se de que o usuário possui uma conta válida e está registrado como administrador.

Ação: Realize o login no sistema para acessar o painel administrativo.

Exemplo:

1. Passo 01: "Verifique se o sistema está conectado à internet antes de enviar o formulário."
   Passo 02: "Ao validar a resposta, garanta que o usuário esteja autenticado."

Como resolver:

1. *Pré-condições agrupadas no início:*
   a. *O sistema deve estar conectado à internet.*
   b. *O usuário deve estar autenticado.*

Ação: Preencha o formulário e envie-o para validação.

5.4 DIRETRIZ 4: PREVENÇÃO DE AÇÕES AGRUPADAS

5.4.1 Motivação

As diretrizes para prevenção de *smells* em ações agrupadas de casos de teste buscam evitar a falta de cobertura de ações não verificadas que podem ocultar problemas no sistema.

5.4.2 Recomendações

1. Evitar passos com múltiplas ações.

Exemplos:

1. "Navegue até 'Relatórios', escolha 'Relatório Mensal', clique em 'Filtrar', selecione o mês desejado, aplique o filtro e exporte o relatório."

   Como resolver:

   1. Acesse a aba 'Relatórios'.
   2. Escolha a opção 'Relatório Mensal'.
   3. Clique no botão 'Filtrar'.
   4. Selecione o mês desejado na lista.
   5. Clique em 'Aplicar'.
   6. Clique no botão 'Exportar Relatório'.

5.5 DIRETRIZ 5: PREVENÇÃO DE AÇÕES DESLOCADAS

5.5.1 Motivação

As diretrizes para prevenção de *smells* de ações mal colocadas visam garantir que as ações respeitem a estrutura canônica de um teste (Seção 2.1.1.2), para simplificar tanto o fluxo de execução do teste quanto futuras manutenções.

5.5.2 Recomendações

1. Pré-condições antes de ações. Assegurar que todas as pré-condições sejam satisfeitas antes de executar uma ação. Garantir que o ambiente e o estado inicial estejam prontos antes de realizar a ação.

Exemplo:

1. "Realize uma transferência bancária e envie o comprovante"

   Como resolver:

   Pré-condição: Verifique se há saldo suficiente na conta para realizar a transferência.

   Ação 1: Execute a transferência bancária para o destinatário.

   Verificação 1: A transferência foi realizada

   Ação 2: Envie o comprovante para o email cliente@cliente.com.

   Verificação 2: Email enviado com o comprovante.

Exemplo:

1. Ação: "Verifique se a mensagem de sucesso é exibida após clicar no botão 'Salvar'."

   Como resolver:

   Ação 1: Clique no botão 'Salvar'.

   Verificação 1: Confirme que a mensagem 'Salvo com sucesso' é exibida.

5.6 DIRETRIZ 6: PREVENÇÃO DE AÇÕES NÃO VERIFICADAS

5.6.1 Motivação

As diretrizes para prevenção de *smells* relacionados a ações não verificadas visam garantir que todas as ações executadas tenham uma verificação correspondente, assegurando que os testes validem o comportamento esperado do sistema.

5.6.2 Recomendações

1. Verificar Todas as Ações. Cada ação executada no teste deve ser acompanhada por uma verificação e ter definido o resultado esperado.

Exemplo:

1. Ação: Altere o status do pedido para "Processando"
2. Ação: Clique no botão "Enviar".
3. Verificação: Certifique-se de que o status do pedido foi atualizado para “Processando”.

Como resolver:

1. *Ação: Altere o status do pedido para "Processando"*
2. *Verificação: Certifique-se de que o status do pedido foi atualizado para “Processando”.*
3. *Ação: Clique no botão "Enviar".*
4. *Verificação: Verifique a mensagem de confirmação: “O pedido foi enviado corretamente”.*

2. Evitar múltiplas ações.

Exemplo:

1. Preencher um formulário com várias informações do usuário (nome, e-mail, endereço, telefone, senha) e enviar o cadastro.

Como resolver:

1. *Passo 1: Preencher o campo "Nome".*
2. *Verificação: Confirmar se o campo aceita apenas caracteres válidos.*

3. *Passo 2: Preencher o campo "E-mail".*
4. *Verificação: Garantir que o campo rejeita formatos inválidos e aceita endereços apenas que possuam @.*
5. *Passo 3: Preencher o campo "Endereço".*
6. *Verificação: Confirmar se o campo aceita apenas caracteres válidos.*
7. *Passo 4: Preencher o campo "Telefone".*
8. *Verificação: Garantir que o campo aceita apenas números e o único caractere especial é o +.*
9. *Passo 5: Preencher o campo "Senha".*
10. *Verificação: Garantir que o campo aceita entre 4 e 8 dígitos.*
11. *Passo 6: Clicar em enviar.*
12. *Verificação: Confirmar que a conta foi criada com sucesso e que o sistema exibe a mensagem de confirmação.*

5.7 DIRETRIZ 7: PREVENÇÃO DE VERIFICAÇÕES DESLOCADAS

5.7.1 Motivação

As diretrizes para prevenção de *smells* relacionados a verificações deslocadas visam garantir que as verificações devem ocorrer após as ações serem definidas, sejam atômicas e estejam alinhadas com o comportamento esperado do sistema e não estejam inseridas em outras etapas do teste.

5.7.2 Recomendações

1. Realizar Verificações imediatamente após a ação correspondente. Garantir que cada ação tenha sua verificação logo em seguida, para assegurar que o comportamento esperado foi obtido.

Exemplo:

1. Ação: Altere o status do pedido para "Processando"
2. Ação: Clique no botão "Enviar".
3. **Ação: Confirme que a mensagem de confirmação "Formulário enviado com sucesso" foi exibida.**

4. *Verificação:* Certifique-se de que o status do pedido foi atualizado corretamente.

Como resolver:

1. *Ação: Altere o status do pedido para "Processando"*

   Verificação: Certifique-se de que o status do pedido foi atualizado para Processando.

2. *Ação: Clique no botão "Enviar".*

   Verificação: Confirme que a mensagem de confirmação "Formulário enviado com sucesso" foi exibida.

Com o objetivo de permitir a leitura e aplicação de maneira independente, há um resumo deste capítulo no ANEXO A.

6 AVALIAÇÃO DAS DIRETRIZES

Neste capítulo discutimos as etapas do processo de avaliação e validação das diretrizes criadas no capítulo anterior. Para isso, foi criado um formulário seguindo o mesmo padrão dos anteriores, do capítulo 4 deste trabalho.

6.1 Caracterização

O questionário foi conduzido de forma completamente anônima e estruturado em três seções: (i) consentimento informado, (ii) dados demográficos e (iii) questões específicas. Na primeira seção, os participantes foram informados sobre os objetivos do estudo, o pesquisador responsável e as opções de consentimento para participação. Na segunda seção, foram coletados dados sobre a ocupação principal dos participantes, categorizada como Academia ou Indústria, com o intuito de analisar a experiência profissional e identificar possíveis divergências entre esses setores. Adicionalmente, foi solicitado aos participantes que indicassem seu tempo de experiência com testes de software utilizando uma escala Likert, variando de 0 (menos de um ano) a 10 (dez ou mais anos). Também foi perguntado sobre a profissão específica, com opções como Analista de Teste de Software, Engenheiro de Teste de Software, Analista de Garantia de Qualidade ou Outros, com um campo para respostas discursivas. Ainda nesta seção, foi solicitado que os participantes indicassem o país de origem.

Na terceira seção, composta por sete perguntas, os participantes foram convidados a expressar seu grau de concordância utilizando uma escala *Likert* de seis pontos, com opções que variavam de “Concordo plenamente” a “Discordo plenamente”, incluindo “Indiferente” e um campo para respostas discursivas. Além disso, foi disponibilizado um espaço para comentários adicionais, caso os participantes desejassem compartilhar suas impressões.

As perguntas específicas com os seus respectivos exemplos foram as seguintes:

Questão 01: Você concorda que os exemplos da Diretriz 1 estão seguindo as recomendações?

Questão 02: Você concorda que os exemplos da Diretriz 2 estão seguindo as recomendações?

Questão 03: Você concorda que os exemplos da Diretriz 3 estão seguindo as recomendações?

Questão 04: Você concorda que os exemplos da Diretriz 4 estão seguindo as recomendações?

Questão 05: Você concorda que os exemplos da Diretriz 5 estão seguindo as recomendações?

Questão 06: Você concorda que os exemplos da Diretriz 6 estão seguindo as recomendações?

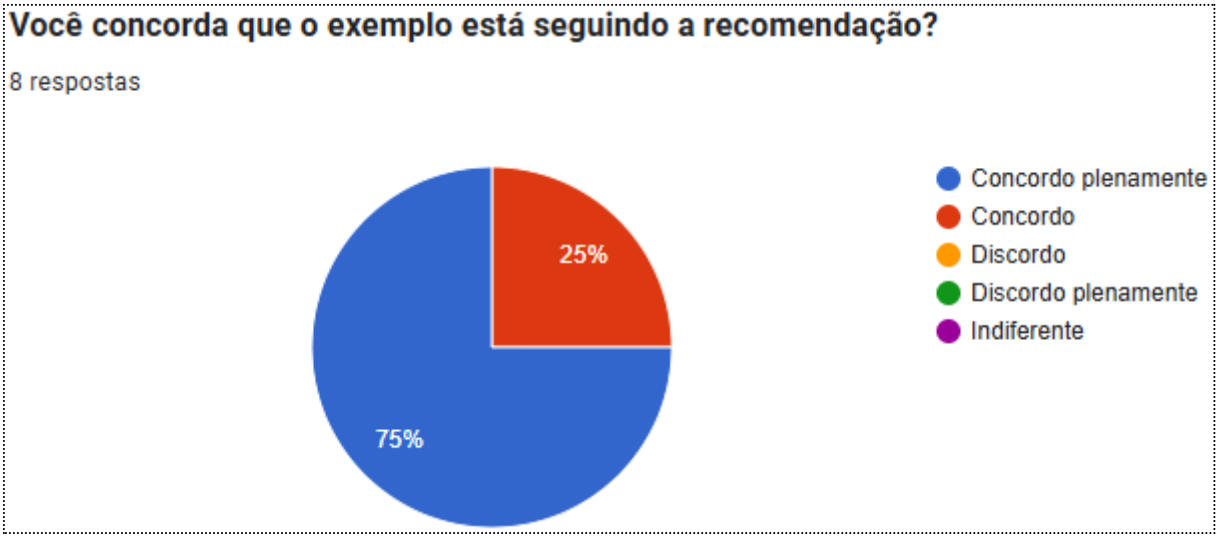
Questão 07: Você concorda que os exemplos da Diretriz 7 estão seguindo as recomendações?

6.2 Resultados

Neste formulário tivemos 08 respostas de profissionais atuantes na área, divididos entre engenheiros de testes de software (75%) e líderes técnicos (25%) da mesma empresa referência em telefonia móvel. Todas as respostas podem ser conferidas no repositório público (Souza, 2025). Na primeira parte sobre dados demográficos, todos os participantes foram da indústria. Com relação ao tempo de experiência na área de testes de software, 12% tem até 4 anos, 25% até 5 anos, 25% até 6 anos, enquanto que a maior parte, 37,5% possuem até 10 anos de experiência.

Na primeira questão, Figura 23, 100% dos participantes concordaram com o exemplo e a solução proposta, divididos entre concordo (25%) e concordo plenamente (75,%). Um deles destacou que: “*Se alguém for fazer o teste após certo tempo pode não encontrar a versão do app que o teste pede, poderia colocar "Youtube **a partir da** versão 3.2 em diante*". A sugestão é salutar desde que a funcionalidade a ser testada esteja desenvolvida na versão sugerida no teste.

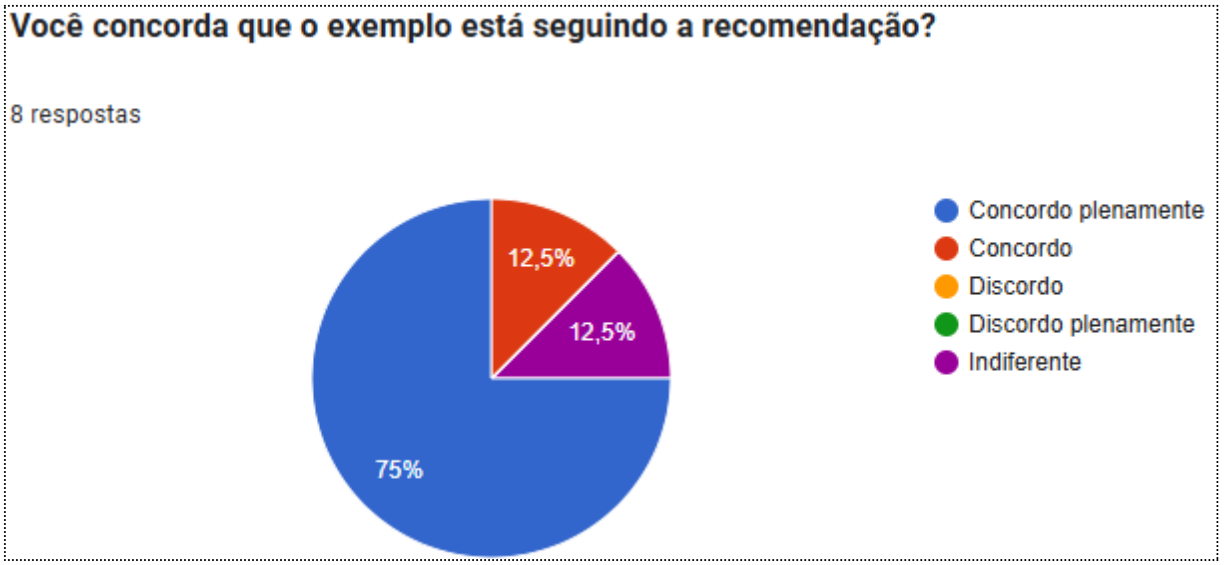
Figura 23: Gráfico referente às respostas do questionário 03 - Questão 01



Fonte: Autor (2025)

Na segunda questão, 87,5% dos participantes concordaram com o exemplo e a solução proposta, enquanto que 12,5% se mostraram indiferentes, de acordo com a Figura 24 abaixo. Um deles acrescentou que: "*Poderia deixar o texto ainda mais claro, ex: Abrir o Google Play Store para atualizar os serviços de GPS".* Confirmando a importância de descrever o significado das siglas na escrita do caso de teste.

Figura 24: Gráfico referente às respostas do questionário 03 - Questão 02.



Fonte: Autor (2025)

Na terceira questão houve concordância de 100% dos participantes (12,5% concordaram e 87,5% concordaram plenamente), Figura 25 abaixo. Um deles destacou que: "*Concordo, mas geralmente quando você está fazendo um batch de*

testes muito grande não tem tempo para ler os pré requisitos, talvez inserir parte deles de forma simples no próprio teste seja uma ideia". Essa configuração pode ser válida a depender do sistema usado para gerenciar os planos de testes. O ideal é que as pré condições do teste estejam inicialmente descritas em cada caso de teste.

Além do comentário anterior, houve outro bastante relevante: *"Esse é um ponto importante que contribui bastante pro entendimento e aparência geral do caso de teste. Também é muito comum, durante o design, que pré-condições sejam escritas como ações (steps) no test case. É algo que tento evitar e corrigir quando detecto em algum test case já criado. Adicionar pré-condições como steps de um case pode acabar aumentando a validação para além do escopo pretendido pelo teste. Um exemplo seria adicionar "Peça permissão para acessar relatórios" como um step do teste quando essa funcionalidade está fora do escopo e não é o que o teste visa validar (que seria o acesso em si, não a definição de permissões)."*

Figura 25: Gráfico referente às respostas do questionário 03 - Questão 03..

Fonte: Autor (2025)

Na quarta questão, também com 100% de concordância, Figura 26 abaixo, um dos participantes trouxe a seguinte reflexão: "*Não tem muito haver com o teste, mas uma reclamação é quando mudam os nomes dessas configurações e não atualizam o teste. Quem já fazia antes infere mas quem nunca fez sempre vem dizendo que achou um erro porque ta com outro nome ou não consegue fazer o teste*". Isso aplica-se à deficiência em relação às refatorações no tempo que a mudança, seja ela qual for, ocorre no software. No contexto profissional dos

participantes, a cada mudança do sistema operacional *Android,* que ocorre anualmente, mas também há inserção de funcionalidades durante todo o ano, é comum as configurações mudarem de caminho, tornando o teste obsoleto em pouco tempo reafirmando a alta demanda investida nas manutenções dos testes.

Figura 26: Gráfico referente às respostas do questionário 03 - Questão 04.

Fonte: Autor (2025)

Na quinta questão, todos os participantes concordaram, Figura 27, e não houveram comentários.

Figura 27: Gráfico referente às respostas do questionário 03 - Questão 05.

Fonte: Autor (2025)

Na penúltima questão, 100% dos participantes concordaram com a recomendação, Figura 28 abaixo, e um dos participantes pontuou que: "*Isso pode deixar o teste muito extenso mas a verificação geralmente é bater o olho, então devia mesmo ter*". Pode ser intuitivo para quem já está acostumado com os testes e a cultura da empresa, mas é necessário documentar todas as informações relevantes para a execução correta do teste.

Figura 28: Gráfico referente às respostas do questionário 03 - Questão 06.

Fonte: Autor (2025)

Na sétima e última questão, todos os participantes concordaram plenamente, não havendo comentários adicionais.

Os resultados obtidos a partir do formulário indicam uma concordância unânime entre os participantes sobre a importância e a necessidade da criação de diretrizes específicas para a escrita de testes de software. Esse consenso reforça a relevância de estabelecer padrões claros que promovam a qualidade, a legibilidade e a manutenção dos testes, contribuindo diretamente para o aumento da confiabilidade do software desenvolvido. Além disso, a unanimidade observada sugere que há uma demanda consolidada entre os profissionais da área, evidenciando que iniciativas nesse sentido podem atender a uma lacuna importante nas práticas atuais. Esses dados fornecem uma base para a elaboração de diretrizes que atendam às necessidades identificadas e fomentem melhores práticas no desenvolvimento de testes.

7 CONCLUSÕES E TRABALHOS FUTUROS

7.1 PRINCIPAIS CONTRIBUIÇÕES

As principais contribuições deste trabalho consistem em identificar pontos de melhorias na escrita dos testes em linguagem natural, sobretudo, da suíte de experiências do usuário, executada pelo time de regressão. Por se tratar de regressão, o alto custo, o tempo e a atenção precisam ser considerados como fatores essenciais para uma boa execução. Durante o ciclo de vida de desenvolvimento do software, não entra no planejamento muitos ciclos de testes de regressão, e com isso, quanto melhor a definição dos testes, o entendimento e a segurança na execução, mais satisfatório serão os resultados.

7.2 LIMITAÇÕES

Grande parte da literatura quando trata-se de *Smells*, direciona as pesquisas para códigos escritos de maneira confusa, buscando formas de melhorar o entendimento e utilização do código. A busca por melhorias em testes, sejam eles automatizados ou escritos em linguagem natural, é recente. O artigo mais antigo encontrado foi do Hauptman em 2013. Ou seja, há pouco mais de uma década que houve interesse em tentar melhorar os testes de software por essa perspectiva, sendo que esta atividade é executada na engenharia de software desde 1947.

7.3 TRABALHOS FUTUROS

Ao considerar os resultados dos dois cenários analisados, há a oportunidade de explorar ainda mais cada um deles. Nesse sentido, no contexto do software da urna eletrônica espera-se, como trabalhos futuros, a ampliação do tempo de análise disponível e o número de pesquisadores e especialistas envolvidos, bem como as técnicas e ferramentas já utilizadas, contribuindo para o desenvolvimento de soluções ainda inéditas, aplicadas ao contexto do ambiente da urna.

Com a continuidade do estudo da especificação dos testes manuais pode-se buscar uma padronização em sua descrição que viabilize a criação de uma ferramenta que automatize sua análise. O desenvolvimento de uma ferramenta de análise automática poderá otimizar o trabalho manual feito inicialmente e contribuir para a melhoria de sua qualidade, reduzindo a quantidade de testes manuais a partir de uma proposta de arquitetura para execução de testes automatizados na própria urna, bem como tornando mais eficiente a especificação de testes manuais.

No contexto de testes de software para dispositivos móveis ao observar a expressiva quantidade de casos de teste na suíte analisada, que é apenas uma parte dos testes de regressão, um dos principais pontos a serem explorados em atividades futuras seria ampliar a análise exploratória, abrangendo todos os casos de teste da suíte de experiências. Essa tarefa é trabalhosa, pois a maioria dos testes possui mais de três passos, além da necessidade de preparar o ambiente adequado para a execução dos testes. Outra ação importante seria organizar os testes por categoria, de acordo com funcionalidades compatíveis e testáveis. Esse aspecto traz um desafio adicional, pois as funcionalidades a serem testadas variam conforme a compatibilidade do dispositivo.

Com a identificação dos testes a serem refatorados, o próximo passo seria tornar a suíte mais coesa, alinhando-a com a cobertura desejada e garantindo uma melhor compreensão dos testes.

De maneira geral, os trabalhos futuros estão relacionados à expansão do conhecimento sobre *test smells* em linguagem natural. A partir destes, o roteiro deste trabalho pode ser repetido de forma a embasar a atualização do ferramental tecnológico existente, bem como das diretrizes para sua prevenção e correção.

Anexo A - Resumo das Diretrizes para prevenção de Smells em Testes Manuais de Software

Diretrizes para prevenção de *Smells* em Testes Manuais de Software				
Nº	Nome	Motivação	Recomendações	Exemplos
1	Prevenção de ambiguidades	As ambiguidades em testes manuais de software deixam espaço para interpretação por quem executa o teste (Hauptmann, 2013). Caso uma ambiguidade esteja colocada numa ação, esta pode causar dificuldades ao testador quanto ao que realizar no sistema sob teste (SUT). Caso a ambiguidade esteja presente numa verificação, o testador poderá ter dificuldades quanto ao que deve ser verificado na resposta do SUT.	1. Não usar artigos e pronomes indefinidos. Evitar termos vagos como “aproximadamente”, “talvez”, “possivelmente” ou qualquer outra expressão que possa ser interpretada de diferentes maneiras.	**“Abra qualquer aplicação e suspenda o computador”** Como resolver: Definir qual a aplicação (incluindo o número da versão) deve ser aberta. **“Em ‘Gravar alterações’, verifique se tudo está correto e selecione SIM”.** Como resolver: Definir o que é esperado como correto. **“Verifique se todos os apps estão dentro da pasta [nome da pasta]”.** Como resolver: Listar quais aplicativos são esperados dentro da pasta [nome da pasta]. **"Insira um valor alto no campo 'Quantidade' e verifique o resultado."** Como resolver: "Insira o valor '5000' no campo 'Quantidade' e verifique se o sistema exibe a mensagem 'Limite máximo excedido'." **"Certifique-se de que a temperatura está dentro de uma faixa aceitável."** Como resolver: "Certifique-se de que a temperatura está entre 20°C e 25°C durante a execução do teste." **"Certifique-se de que o arquivo seja salvo."** Como resolver: "O arquivo 'documento.docx' deve ser salvo no diretório padrão 'C:/Documentos' com a data e hora atual visíveis no nome do arquivo." **“Verifique que o botão ‘Enviar’ funciona corretamente.”** Como resolver: "O sistema deve exibir a mensagem 'Dados enviados com sucesso' e redirecionar o usuário para a página 'Resumo do Pedido'." **"Confirme que o item foi adicionado ao carrinho."** Como resolver: "O item aparece no resumo do carrinho com as informações de nome, quantidade e preço." **"Verifique se a mensagem aparece na página."** Como resolver: "Verifique se a mensagem 'Cadastro realizado com sucesso' aparece no topo da página 'Registro de Usuário'."
			2. Substitua advérbios e adjetivos por critérios quantificáveis.	Certifique-se de que o processo de carregamento é rápido. Como resolver: "O processo de carregamento do relatório é concluído em até 5 segundos." "O sistema deve apresentar a melhor performance durante o teste." Como resolver: "O sistema deve processar 1000 requisições simultâneas com um tempo de resposta inferior a 200ms para cada requisição."

2	Prevenção de termos específicos do domínio	A ausência de definição de termos específicos do domínio ou siglas não definidas pode causar insegurança na execução do caso de teste.	1. Explicar termos específicos do domínio. Definir siglas, valores, tamanhos e/ou qualquer informação referente ao domínio do conhecimento necessário para a execução do teste, tal como os resultados esperados para cada ação	**Exemplo:** **Pré-condição: Conecte o dispositivo à internet** **Ação: Acesse configurações** **Verificação: O menu de configurações foi acessado com sucesso** **Ação: Atualize os serviços de GPS** **Verificação: Os serviços foram atualizados** Como resolver: Pré-condição: Conecte o dispositivo a internet Ação: Acesse configurações Verificação: O menu de configurações foi acessado com sucesso Ação: Atualize os serviços de GPS (Google Play Store) Verificação: Os serviços foram atualizados
3	Prevenção de pré-condições deslocadas	As diretrizes para a prevenção de smells relacionados à pré-condições visam garantir que as pré-condições respeitem a estrutura canônica de um teste para suportar a execução do teste.	1. Definir as Pré-condições no início do caso de teste antes de qualquer ação ser executada. Isso garante que o ambiente e os estados necessários estejam prontos antes do início do teste.	**Ação: "Acesse o relatório de vendas e selecione o período desejado."** Como resolver: Pré-condição 01: Certifique-se de que o usuário está autenticado e tem permissão para acessar relatórios. Pré-condição 02: Certifique-se de que o usuário tem permissão para acessar relatórios. Ação: Acesse o relatório de vendas e selecione o período desejado. **"Realize o login no sistema para acessar o painel administrativo".** Como resolver: Pré-condição: Certifique-se de que o usuário possui uma conta válida e está registrado como administrador. Ação: Realize o login no sistema para acessar o painel administrativo. **Passo 01: "Verifique se o sistema está conectado à internet antes de enviar o formulário."** **Passo 02: "Ao validar a resposta, garanta que o usuário esteja autenticado."** Como resolver: Pré-condições agrupadas no início: O sistema deve estar conectado à internet. O usuário deve estar autenticado. Ação: Preencha o formulário e envie-o para validação.
4	Prevenção de ações agrupadas	As diretrizes para prevenção de smells em ações agrupadas de casos de teste buscam evitar a falta de cobertura de ações não verificadas que podem ocultar problemas no sistema.	1. Evitar passos com múltiplas ações.	**"Navegue até 'Relatórios', escolha 'Relatório Mensal', clique em 'Filtrar', selecione o mês desejado, aplique o filtro e exporte o relatório."** Como resolver: 1. Acesse a aba 'Relatórios'. 2. Escolha a opção 'Relatório Mensal'. 3. Clique no botão 'Filtrar'. 4. Selecione o mês desejado na lista. 5. Clique em 'Aplicar'. 6. Clique no botão 'Exportar Relatório'.
5	Prevenção de ações deslocadas	As diretrizes para prevenção de smells de ações mal colocadas visam garantir que as ações respeitem a estrutura canônica de um teste, para simplificar tanto o fluxo de execução do teste quanto futuras manutenções.	1. Pré-condições antes de ações. Assegurar que todas as pré-condições sejam satisfeitas antes de executar uma ação. Garantir que o ambiente e o estado inicial estejam prontos	**"Realize uma transferência bancária e envie o comprovante"** Como resolver: Pré-condição: Verifique se há saldo suficiente na conta para realizar a transferência. Ação 1: Execute a transferência bancária para o destinatário. Verificação 1: A transferência foi realizada Ação 2: Envie o comprovante para o email cliente@cliente.com.

			antes de realizar a ação.	Verificação 2: Email enviado com o comprovante.
6	Prevenção de ações não verificadas	As diretrizes para prevenção de smells relacionados a ações não verificadas visam garantir que todas as ações executadas tenham uma verificação correspondente, assegurando que os testes validem o comportamento esperado do sistema.	1. Verificar Todas as Ações. Cada ação executada no teste deve ser acompanhada por uma verificação e ter definido o resultado esperado.	**Ação: Altere o status do pedido para "Processando"** **Ação: Clique no botão "Enviar".** **Verificação: Certifique-se de que o status do pedido foi atualizado para "Processando".** Como resolver: Ação: Altere o status do pedido para "Processando" Verificação: Certifique-se de que o status do pedido foi atualizado para "Processando". Ação: Clique no botão "Enviar". Verificação: Verifique a mensagem de confirmação: "O pedido foi enviado corretamente".
			2. Evitar múltiplas ações.	**Preencher um formulário com várias informações do usuário (nome, e-mail, endereço, telefone, senha) e enviar o cadastro.** Como resolver: Passo 1: Preencher o campo "Nome". Verificação: Confirmar se o campo aceita apenas caracteres válidos. Passo 2: Preencher o campo "E-mail". Verificação: Garantir que o campo rejeita formatos inválidos e aceita endereços apenas que possuam @. Passo 3: Preencher o campo "Endereço". Verificação: Confirmar se o campo aceita apenas caracteres válidos. Passo 4: Preencher o campo "Telefone". Verificação: Garantir que o campo aceita apenas números e o único caractere especial é o +. Passo 5: Preencher o campo "Senha". Verificação: Garantir que o campo aceita entre 4 e 8 dígitos. Passo 6: Clicar em enviar. Verificação: Confirmar que a conta foi criada com sucesso e que o sistema exibe a mensagem de confirmação.
7	Prevenção de verificações deslocadas	As diretrizes para prevenção de smells relacionados a verificações deslocadas visam garantir que as verificações devem ocorrer após as ações serem definidas, sejam atômicas e estejam alinhadas com o comportamento esperado do sistema e não estejam inseridas em outras etapas do teste.	1. Realizar Verificações imediatamente após a ação correspondente. Garantir que cada ação tenha sua verificação logo em seguida, para assegurar que o comportamento esperado foi obtido.	**Ação: Altere o status do pedido para "Processando"** **Ação: Clique no botão "Enviar".** **Ação: Confirme que a mensagem de confirmação "Formulário enviado com sucesso" foi exibida.** **Verificação: Certifique-se de que o status do pedido foi atualizado corretamente.** Como resolver: Ação: Altere o status do pedido para "Processando" Verificação: Certifique-se de que o status do pedido foi atualizado para Processando. Ação: Clique no botão "Enviar". Verificação: Confirme que a mensagem de confirmação "Formulário enviado com sucesso" foi exibida.

Referências

ABORADAN, Anas; LANDING, Josef. **Finding Bad Smells in natural language Test Specifications Using NALABS**. 2022.

ARANDA, Manoel et al. A Catalog of Transformations to Remove Smells From Natural Language Tests. **arXiv preprint arXiv:2404.16992**, 2024.

AZEEM, Muhammad Ilyas et al. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. **Information and Software Technology**, v. 108, p. 115-138, 2019.

B. Hauptmann, M. Junker, S. Eder, L. Heinemann, R. Vaas, and P. Braun, "**Hunting for smells in natural language tests**," in 35th International Conference on Software Engineering, ser. ICSE. New York, NY, USA: IEEE, 2013, pp. 1217–1220.

BARROS, Víctor Pedroso Ambiel; MATOS, Simone N. **Um método para a refatoração de software baseado em frameworks de domínio**. 2015. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

BECK, Kent; FOWLER, Martin; BECK, Grandma. Bad smells in code. **Refactoring: Improving the design of existing code**, v. 1, n. 1999, p. 75-88, 1999.

BERNARDO, Paulo Cheque. **Padrões de testes automatizados**. 2011. Tese de Doutorado. Universidade de São Paulo.

BURNETTE, Ed. **Hello, Android introducing Google's mobile development platform: 2nd**. Android, 2009.

DAMASCENO, Humberto; BEZERRA, Carla. **Ensino da Prática de Refatoração de test smells**. In: Anais do XXXI Workshop sobre Educação em Computação. SBC, 2023. p. 293-304.

ENGSTRÖM, E.; RUNESON, P.; SKOGLUND, M. **A systematic review on regression test selection techniques. Information and Software Technology**, Elsevier, v. 52, n. 1, p. 14–30, 2010

FOWLER, Martin. **Refatoração: Aperfeiçoando o design de códigos existentes**. Novatec Editora, 2ª edição, 2020

G. Bavota, A. De Lucia, M. Di Penta, R. Oliveto, and F. Palomba. **An experimental investigation on the innate relationship between quality and refactoring**. Journal of Systems and Software, 107:1–14, 2015.

GALIN, Daniel. **Software quality assurance: from theory to implementation**. Pearson education, 2004.

GAROUSI, Vahid; KÜÇÜK, Barış. Smells in software test code: A survey of knowledge in industry and academia. **Journal of systems and software**, v. 138, p. 52-81, 2018.

HANNIBAL, M.; MONTANI, I.; VAN LANDEGHEM, S. Boyd A. spaCy: Industrial-strength Natural Language Processing in Python. 2020.

HAZIN, Raphael et al. Técnicas e Padrões para Otimizar a Interpretação de Códigos Fontes Aplicados à Fábrica de Software, 2017.

J. E. Bartlett II, J. W. Kotrlik, and C. C. Higgins, "Organizational research: Determining appropriate sample size in survey research," Information technology, learning, and performance journal, vol. 19, no. 1, pp. 43−50, 2001.

LAM, Wing et al. iDFlakies: A framework for detecting and partially classifying flaky tests. In: **2019 12th ieee conference on software testing, validation and verification (icst)**. IEEE, 2019. p. 312-322.

LOPES, Lucas Moreira et al. Teste de validação de software para smartphones android. 2022.

LOPES, Gustavo Augusto Calazans et al. Detecção de smells em testes automatizados em diferentes linguagens de programação. 2023.

MACIEL, Ana Carla Fernandes; VALLS, Carmem; SAVOINE, Márcia Maria. **Análise da qualidade de software utilizando as normas 12207, 15504, ISO 9000-3 e os modelos CMM/CMMI e MPS**. BR. Revista Cientifica do ITPAC, v. 4, p. 1-13, 2011.

NAIK, Kshirasagar; TRIPATHY, Priyadarshi. **Software testing and quality assurance: theory and practice**. John Wiley & Sons, 2011.

NOGUEIRA, Pedro Magnus Pedroso; DA COMPUTAÇÃO, Ciência; MATOS, Drª Simone Nasser. Mapeamento Sistemático de Processos de Refatoração de Software.

OLIVEIRA, Everton Rennê Barros de. **Relação entre refatorações e code smells na evolução de projetos de software e seu reflexo em medidas de software**. 2020. Dissertação de Mestrado. Universidade Federal de Pernambuco.

OPDYKE, William F. **Refactoring object-oriented frameworks**. University of Illinois at Urbana-Champaign, 1992.

PALOMBA, Fabio; ZAIDMAN, Andy. Notice of retraction: Does refactoring of test smells induce fixing flaky tests?. In: **2017 IEEE international conference on software maintenance and evolution (ICSME)**. IEEE, 2017. p. 1-12.

PEREIRA, Lucio Camilo Oliva; DA SILVA, Michel Lourenço. **Android para desenvolvedores**. Brasport, 2009.

PERUMA, Anthony et al. On the distribution of test smells in open source android applications: An exploratory study. 2019.

PRESSMAN, R. S.. **Engenharia de Software**. Tradução: Carlos Santos. 2. ed. São Paulo: Makron Books, 1995.

SANTANA, Railana dos Santos. RAIDE: uma abordagem semi-automatizada para identificação e refatoração de test smells. 2021.

SANTOS, André Luís de Medeiros; MIRANDA, Breno; BARROS, Roberto Souto Maior de; SOARES, Elvys; SOUZA, Emerson Paulo Soares de; FREITAS, Davi Simões. **Relatório sobre o Projeto Piloto de Inspeção do Código-Fonte da Urna Eletrônica na UFPE**. 2022. Disponível em: https://www.tse.jus.br/++theme++justica_eleitoral/pdfjs/web/viewer.html?file=https://www.tse.jus.br/co===municacao/arquivos/relatorio-ufpe/@@download/file/Relatorio_Piloto_UFPE_TSE_CodigoFonte.pdf. Acesso em: 01 out. 2024.

SPADINI, Davide et al. On the relation of test smells to software code quality. In: **2018 IEEE international conference on software maintenance and evolution (ICSME)**. IEEE, 2018. p. 1-12.

STAPP, Lucjan; ROMAN, Adam; PILAETEN, Michaël. **ISTQB® Certified Tester Foundation Level: A Self-Study Guide Syllabus V4. 0**. Springer, 2024.

SIQUEIRA, Vinicius José de. **History-based prioritization in the context of manual testing: a study in a real industrial setting**. 2022. Dissertação de Mestrado. Universidade Federal de Pernambuco.

SOARES, Elvys et al. Refactoring test smells: A perspective from open-source developers. In: Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing. 2020. p. 50-59.

SOARES, Elvys et al. Manual Tests Do Smell! Cataloging and Identifying Natural Language test smells. In: **2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)**. IEEE, 2023. p. 1-11.

Souza, Emerson P S (2025). Answers to the questionnaires applied during the survey. figshare. Dataset. https://doi.org/10.6084/m9.figshare.28319015

LAM, W. et al. idflakies: **A framework for detecting and partially classifying flaky tests**. In: IEEE. 2019 12th ieee conference on software testing, validation and verification (icst). [S.l.], 2019. p. 312–322.

LUO, Q. et al. **An empirical analysis of flaky tests**. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering. [S.l.: s.n.], 2014. p. 643–653.

VAN DEURSEN, Arie et al. Refactoring test code. In: **Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)**. Citeseer, 2001. p. 92-95.

VIANA, Virginia Maria Araújo. Um método para seleção de testes de regressão para automação. 2006. Dissertação de Mestrado. Universidade Federal de Pernambuco.

A. van Deursen, L. Moonen, A. Bergh, and G. Kok. **Refactoring test code**. In Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP), pages 92–95, 2001.