INSTITUTO FEDERAL DE ALAGOAS

CAMPUS ARAPIRACA

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FELIPE DA SILVA SANTOS

ROBSON ALVES GOMINHO

PROSPECÇÃO E DETECÇÃO DE *SMELLS* EM CASOS DE TESTE ESCRITOS NO PADRÃO BDD COM A LINGUAGEM GHERKIN

ARAPIRACA, AL

2025

FELIPE DA SILVA SANTOS
ROBSON ALVES GOMINHO

PROSPECÇÃO E DETECÇÃO DE *SMELLS* EM CASOS DE TESTE ESCRITOS NO PADRÃO BDD COM A LINGUAGEM GHERKIN

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Sistemas de Informação do Instituto Federal de Alagoas, Campus Arapiraca, como requisito parcial para obtenção de grau de Bacharel em Sistemas de Informação.

Orientador: Dr. Társis Marinho de Souza
Coorientador: Dr. Elvys Alves Soares

ARAPIRACA, AL
2025

Dados Internacionais de Catalogação na Publicação
Instituto Federal de Alagoas
***Campus* Arapiraca**

005
S237p Santos, Felipe da Silva.

Prospecção e detecção de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin [recurso eletrônico] / Felipe da Silva Santos, Robson Alves Gominho. – Dados eletrônicos (1 arquivo : 9 MB). – 2025.

Sistema requerido: Adobe Acrobat Reader.
Modo de acesso: Internet.
Orientação: Prof. Dr. Társis Marinho de Souza.
Co-orientador: Prof. Dr. Elvys Alves Soares.
Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Instituto Federal de Alagoas, *Campus* Arapiraca, Arapiraca, 2025.

1. Test smells. 2. BDD. 3. Gherkin. 4. Qualidade de software I. Gominho, Robson Alves. II. Título.

Luciete Barbosa da Silva | Bibliotecária – CRB-4/1739

FELIPE DA SILVA SANTOS
ROBSON ALVES GOMINHO

PROSPECÇÃO E DETECÇÃO DE *SMELLS* EM CASOS DE TESTE ESCRITOS NO PADRÃO BDD COM A LINGUAGEM GHERKIN

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Sistemas de Informação do Instituto Federal de Alagoas, Campus Arapiraca, como requisito parcial para obtenção de grau de Bacharel em Sistemas de Informação.

Aprovado em: 16/04/2025.

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
TARSIS MARINHO DE SOUZA
Data: 05/05/2025 17:58:35-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Társis Marinho de Souza (Orientador)
Instituto Federal de Alagoas – *Campus* Arapiraca

Documento assinado digitalmente
gov.br
ELVYS ALVES SOARES
Data: 05/05/2025 20:01:23-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Elvys Alves Soares (Coorientador)
Instituto Federal de Alagoas – *Campus* Maceió

Documento assinado digitalmente
gov.br
LEONARDO FERNANDES MENDONCA DE OLIVEI
Data: 06/05/2025 15:43:13-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Leonardo Fernandes Mendonça de Oliveira
Instituto Federal de Alagoas – *Campus* Maceió

Documento assinado digitalmente
gov.br
FLAVIO MOTA MEDEIROS
Data: 07/05/2025 07:12:39-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Flavio Mota Medeiros
Instituto Federal de Alagoas – *Campus* Maceió

AGRADECIMENTOS

Agradeço, primeiramente, à minha família — minha mãe, meu pai e meus dois irmãos — por todo o apoio, amor e incentivo ao longo da minha trajetória. Sem vocês, nada disso seria possível.

Ao professor Dr. Társis Marinho, minha profunda gratidão por ter sido meu professor durante oito anos. Agradeço pelos ensinamentos, pelas oportunidades e por acreditar no meu potencial desde o início. Sua orientação foi essencial para o meu crescimento acadêmico e profissional.

Agradeço ao professor Dr. Elvys Soares pelos valiosos conhecimentos compartilhados na área de qualidade de software e, principalmente, pela paciência em cada momento de aprendizado.

Minha imensa gratidão ao Robson, que foi a pessoa que mais me ajudou nesses quatro anos de faculdade. Sua ajuda constante fez toda a diferença nessa jornada.

Por fim, agradeço aos meus amigos pelo companheirismo e apoio incondicional durante todos esses anos.

FELIPE DA SILVA SANTOS

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por ter me guiado até aqui, e às minhas tias, Maria e Noemi, pelo apoio incondicional e carinho ao longo de toda minha trajetória acadêmica.

Minha profunda gratidão ao meu orientador, professor Dr. Társis Marinho, pela confiança depositada em mim, pelas oportunidades de crescimento profissional e, sobretudo, pela amizade construída ao longo do curso.

Estendo meus agradecimentos ao professor Dr. Elvys Soares, coorientador essencial para o desenvolvimento deste trabalho, que me auxiliou em todas as etapas de pesquisa.

Dedico este trabalho, que simboliza todo o meu esforço e dedicação, aos meus pais, que, mesmo à distância, sempre me incentivaram. Um agradecimento especial ao meu irmão, Kelven, cuja admiração me motiva a seguir em frente e mostrar a minha melhor versão.

Faço uma menção honrosa aos meus muitos amigos de curso, que estiveram ao meu lado nessa caminhada, desafiando-me a evoluir constantemente e contribuindo para o meu crescimento pessoal e profissional.

ROBSON ALVES GOMINHO

RESUMO

Os *test smells* são indícios de problemas estruturais que comprometem a qualidade de testes de *software*. Eles têm sido estudados em diferentes contextos, mas sua presença em testes no padrão *Behavior-Driven Development* (BDD) ainda é pouco investigada. Dado a crescente adoção do BDD na indústria, especialmente com a linguagem Gherkin, torna-se importante compreender se esses *smells* também ocorrem nessa tecnologia. O presente trabalho tem como objetivo identificar e categorizar *test smells* em testes escritos no padrão BDD, contribuindo para a melhoria da qualidade e manutenção dos testes. Para isso, foi realizada uma revisão sistemática para identificar *smells* no contexto de BDD, seguida por uma análise manual dos repositórios públicos de 7 projetos no *GitHub* que utilizam Gherkin, onde a partir dessas observações, foram propostos 12 tipos de *smells* efetivamente identificados na base de dados utilizada, dos quais passaram por um estudo de validação com 22 profissionais que trabalham com testes Gherkin, posteriormente sendo desenvolvida uma ferramenta automatizada para identificação desses *smells* propostos. Utilizou-se a ferramenta para analisar os repositórios selecionados e foram calculadas as suas métricas de *precision*, *recall* e *f-measure* nas detecções de *smells*. Os resultados indicaram a presença de um grande número de *smells* nos testes analisados, corroborando a hipótese de que esses problemas existem no contexto do BDD. Além disso, a ferramenta criada se mostrou eficiente na identificação de novos *smells*, ampliando o alcance da análise. Conclui-se que há uma lacuna significativa no estudo desses indícios em BDD e que esforços adicionais são necessários para investigar mais profundamente esse tema, visto que mesmo com uma base de dados limitada, foi possível encontrar uma quantidade expressiva de *smells*.

Palavras-chave: *Test smells;* BDD*;* Gherkin*;* Cucumber*;* qualidade de software*.*

ABSTRACT

Test smells are signs of structural problems that compromise the quality of software tests. They have been studied in different contexts, but their presence in tests in the Behavior-Driven Development (BDD) pattern is still little investigated. Given the growing use of BDD in industry, especially with the Gherkin language, it becomes important to understand whether these smells also occur in this technology. This work aims to identify and categorize test smells in tests written in the BDD pattern, contributing to the improvement of test quality and maintenance. To this end, a systematic review was carried out to identify smells in the context of BDD, followed by a manual analysis of the public repositories of 7 projects on GitHub that use Gherkin. From these observations, 12 types of smells effectively identified in the database used were proposed. 12 new smells were then validated with 22 professionals who work with Gherkin tests. An automated tool was subsequently developed to identify these proposed smells. The tool was used to analyze the selected repositories and their precision, recall and f-measure metrics were calculated in the smell detections. The results indicated the presence of a large number of smells in the analyzed tests, corroborating the hypothesis that these problems exist in the context of BDD. In addition, the created tool proved to be efficient in identifying new smells, expanding the scope of the analysis. It is concluded that there is a significant gap in the study of these signs in BDD and that additional efforts are needed to investigate this topic in more depth, since even with a limited database, it was possible to find a significant number of smells.

Keywords: Test smells; BDD; Gherkin; Cucumber; software quality.

LISTA DE QUADROS E TABELAS

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIAÇÕES E SIGLAS

API	Application Programming Interface
BDD	Behavior-Driven Development
CLI	Command-line Interface
CSV	Comma-Separated Values
FN	False Negative
FP	False Positive
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MLM	Multivocal Literature Mapping
MLR	Multivocal Literature Review
OSS	Open Source Software
QA	Quality Assurance
SGBD	Sistema de Gerenciamento de Banco de Dados
SLR	Systematic Literature Review
SQL	Structured Query Language
TCLE	Termo de Consentimento Livre e Esclarecido
TDD	Test-Driven Development
TN	True Negative
TP	True Positive

SUMÁRIO

1 INTRODUÇÃO

No contexto do *Behavior-Driven Development* (BDD), os testes são implementados para melhorar a qualidade do produto, permitindo uma linha de desenvolvimento segura e comunicativa entre toda a equipe do projeto, graças à definição sólida de requisitos, validação constante das funcionalidades da ferramenta, que garantem a sua integridade, redução dos custos de retrabalho e por apontar problemas antes da escrita de código do *software*. Testes auxiliam a equipe na compreensão do código de forma mais ágil e facilitam a construção do produto final. “Nesse sentido, o código de teste é utilizado como uma rede de segurança para evoluir e evitar que regressões ocorram” (Marabesi; García-Holgado; García-Peñalvo, 2024, p. 1, tradução nossa).

Além disso, a presença de uma suíte de testes bem elaborada contribui diretamente para a confiabilidade e manutenção do sistema. Testes funcionam como uma espécie de documentação viva, descrevendo comportamentos esperados do sistema e auxiliando novos membros da equipe a entenderem rapidamente as regras de negócio. Em projetos com ciclos de desenvolvimento contínuo, os testes se tornam essenciais para sustentar a qualidade ao longo do tempo e acelerar as entregas.

Entre as abordagens populares no mercado, destacam-se o *Test-Driven Development* (TDD) e o seu sucessor, BDD. De acordo com Marabesi, García-Holgado e García-Peñalvo (2024, p. 1) o TDD é uma metodologia que se sustenta na escrita de código de teste antes do código de produção, sendo fundamentado em três etapas principais:

- **Vermelha**: Escrita do código de teste, visando compreender seu comportamento;
- **Verde**: Implementação do teste com o menor número de alterações possíveis;
- **Azul**: Refatoração do teste, garantindo qualidade e evitando duplicação.

Por outro lado, o BDD propõe a definição de testes em linguagem natural antes mesmo da implementação do código. Essa abordagem busca promover uma melhor comunicação entre todos os envolvidos do projeto, como desenvolvedores, equipe de garantia de qualidade (QA) e *stakeholders*, ao descrever comportamentos

esperados do sistema de forma compreensível. Essa descrição, muitas vezes representada por meio da linguagem de marcação Gherkin, estrutura os testes em cenários compostos por passos como Dado que, Quando e Então. Usualmente, o BDD é adotado durante o desenvolvimento de histórias de usuário, nas quais diferentes papéis de usuário executam ações com objetivos específicos. Quando bem aplicado, o BDD atua como ponte entre requisitos de neget alócio e implementação técnica, facilitando a validação dos critérios de aceitação e garantindo que o *software* atenda às expectativas do usuário final (Farooq *et al.*, 2023). O BDD possui três palavras-chaves no seu desenvolvimento, que podem ser descritas como (Farooq *et al.*, 2023):

- ***Given***: Define o contexto inicial do evento (não obrigatório);
- ***When***: Contexto atual do evento, uma ação que será executada;
- ***Then***: Demonstra o resultado esperado da ação.

Embora o TDD e o BDD facilitem o trabalho colaborativo entre equipes de desenvolvimento, QA e pessoas não-técnicas ou de negócio, essas metodologias ágeis não estão isentas de desafios. A documentação de testes em linguagem natural normalmente resulta em casos de teste incompreensíveis, ambíguos, duplicados e de difícil manutenção, graças aos problemas de ortografia, tradução, diferentes estilos de descrição aplicados para procedimentos semelhantes e até mesmo abuso de abreviações que podem confundir novos testadores (Juhnke; Nikic; Tichy, 2021). Nesse contexto, emerge o conceito de *test smell*, uma característica recorrente em testes que indica um possível problema em sua estrutura, clareza ou manutenção, ainda que o teste em si não esteja, necessariamente, errado. Os *test smells* servem como sinais de alerta sobre a qualidade dos testes, podendo comprometer a sua legibilidade, confiabilidade e capacidade de evoluir junto ao código.

Os *test smells* são um problema significativo para a área de QA. Apesar de serem amplamente discutidos em meios informais da literatura, ainda há uma falta de padronização formal sobre o tema, especialmente no contexto do BDD. No entanto, esse cenário vem evoluindo. Em um estudo recente, Soares (2023), identificou 8 novos *smells* em casos de teste manuais escritos em linguagem natural,

o que evidencia que, para essa metodologia, ainda há muito a ser aprofundado na literatura acadêmica.

Diante do exposto, visando contribuir para a área de qualidade de *software*, o presente trabalho visa prospectar uma solução de detecção de *smells* em casos de teste escritos no padrão BDD com a linguagem Gherkin. Para isso, foi realizado um processo de levantamento de requisitos, onde a solução foi avaliada por meio de um formulário entre profissionais da área para garantir a relevância das propostas encontradas.

1.1 QUESTÕES DE PESQUISA

Levando essa problemática em consideração, esta pesquisa visa responder às seguintes questões:

- **QP1**: *"Quais smells podem ser encontrados em casos de teste escritos no padrão BDD com a linguagem Gherkin?"*;
- **QP2**: *"Quais são as formas de detecção de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?"*;
- **QP3**: *"Qual a frequência dos smells encontrados em casos de teste escritos no padrão BDD com a linguagem Gherkin?"*;
- **QP4**: *“É possível detectar de forma automatizada smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?”;*
- **QP5**: *“O quão eficiente é a detecção automatizada de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?”.*

1.2 OBJETIVOS

Como principal objetivo, busca-se **"Prospectar e detectar *smells* em casos de teste escritos no padrão BDD com a linguagem Gherkin"**, e para alcançá-lo, os seguintes objetivos específicos foram traçados:

- Conceituar casos de teste, *test smells*, TDD, BDD, Gherkin e Cucumber;
- Prospectar casos de teste escritos no padrão BDD com a linguagem Gherkin;
- Detectar *smells* em casos de teste escritos no padrão BDD com a linguagem Gherkin nos projetos selecionados;
- Categorizar os *smells* encontrados de acordo com as suas características;

- Documentar regras que caracterizam cada um dos *smells* encontrados;
- Classificar os *smells* encontrados de acordo com o número de ocorrências;
- Criar suporte ferramental para a detecção dos *smells* propostos.

Este é um estudo de caso que utiliza abordagem exploratória de natureza aplicada. A relevância desta pesquisa está associada à necessidade de garantir que os testes mantenham sua qualidade e utilidade ao longo do tempo, oferecendo uma análise sistemática dos problemas estruturais mais comuns e propondo soluções para a sua detecção e mitigação, com o auxílio de uma ferramenta criada.

1.3 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado em cinco seções, descritas da seguinte forma: na próxima seção, será apresentada a fundamentação teórica deste trabalho, contendo definições relevantes sobre metodologias, linguagens e *test smells*. Em seguida, uma discussão metodológica do processo de análise de um catálogo de *test smells* existentes, de revisão sistemática da literatura formal e informal, mineração de dados de 7 repositórios públicos no GitHub, análises amostrais que resultaram na descoberta de 12 *smells* presentes em casos de teste escritos na linguagem Gherkin, do formulário de pesquisa e a construção da ferramenta de detecção automática dos *smells* de teste propostos. Posteriormente, são descritos os resultados obtidos, onde serão apresentados o apuramento da amostragem de validação da ferramenta e do formulário, seguidos pelas considerações finais.

2 JUSTIFICATIVA

Discutir sobre a prospecção e detecção de *smells* em casos de teste escritos no padrão BDD com a linguagem Gherkin justifica-se pela ausência de padronização de trabalhos acadêmicos e profissionais sobre o tema, apesar da crescente adoção do BDD no desenvolvimento de *software*. Assim, é possível notar que a ausência de estudos aprofundados sobre os *smells* presentes em casos de teste pode impactar direta ou indiretamente empresas e equipes de desenvolvimento, através de problemas na documentação, comunicação ineficaz entre *stakeholders* e desenvolvedores, retrabalho devido a ambiguidades e inconsistências, aumento de *bugs* não detectados precocemente e, consequentemente, maiores custos e atrasos nos projetos de *software*.

A escolha do tema "Prospecção e detecção de *smells* em casos de teste escritos no padrão BDD com a linguagem Gherkin" justifica-se porque, apesar da relevância crescente do BDD como metodologia ágil que promove a colaboração entre desenvolvedores, testadores e *stakeholders*, ainda existe uma notável lacuna na literatura e nas práticas industriais sobre a qualidade dos casos de teste escritos utilizando Gherkin. Profissionais e organizações da área de Engenharia de *Software* poderão beneficiar-se significativamente deste estudo, com base em melhorias na documentação dos testes, prevenção de *bugs* e redução do retrabalho. Assim, o presente trabalho partiu da necessidade de entender como identificar e corrigir *smells* pode melhorar a qualidade dos testes, otimizar o tempo e recursos empregados no desenvolvimento e facilitar o processo de comunicação e transferência de conhecimento entre equipes e *stakeholders*.

Portanto, fica evidente a importância do estudo proposto, considerando o seu potencial para impactar positivamente o processo de desenvolvimento de *software* ao contribuir para a redução de custos operacionais, prevenção de retrabalhos desnecessários e garantia de uma maior qualidade dos produtos entregues, resultando em uma melhor experiência para os usuários finais.

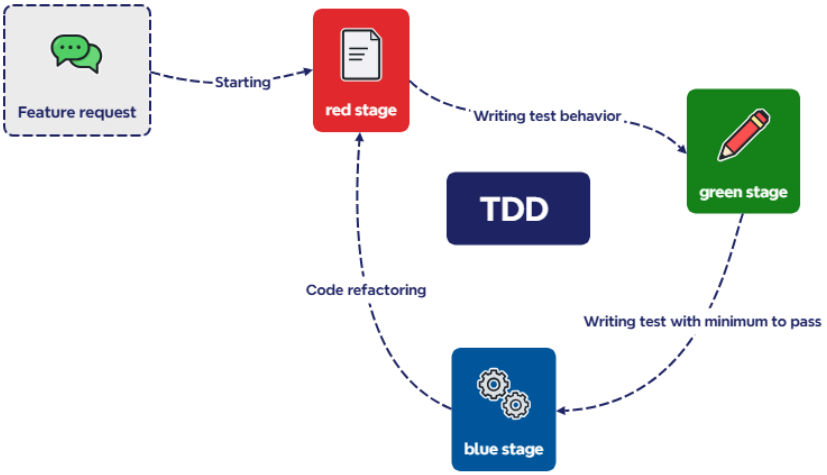
3 FUNDAMENTAÇÃO TEÓRICA

Este trabalho está fundamentado em artigos científicos, documentos oficiais, teses e dissertações que proporcionam um aprofundamento significativo a respeito do tema. A visão científica de autores e especialistas da área contribui para definir, com base sólida, cada subtópico relevante para o desenvolvimento desta pesquisa.

3.1 TEST-DRIVEN DEVELOPMENT (TDD)

O TDD é uma prática que foi popularizada pela *eXtreme Programming*, na intenção de agilizar o processo de desenvolvimento de *software*. Essa abordagem prioriza a testabilidade do código para dar suporte à dinâmica dos negócios. Seguindo as suas etapas de desenvolvimento (vermelho, verde e azul), a promessa seria melhorar a qualidade do código e mitigar *code smells*. (Marabesi; García-Holgado; García-Peñalvo, 2024, p. 1). A Figura 1 ilustra o ciclo de vida do TDD.

Figura 1 - Ciclo de vida da metodologia TDD



Fonte: Os autores.

Além disso, o TDD é um assunto relevante entre a comunidade ágil, uma vez que sua aplicação demonstra melhorias na qualidade do *software* ao longo das últimas décadas. Além disso, essa técnica pode ser combinada com outras

tecnologias, como o próprio BDD, para aprimorar a clareza e a comunicação dos requisitos de *software*. Por ser muito ampla, o TDD ainda aborda de forma mínima práticas de teste de unidade e de aceitação (Marabesi; García-Holgado; García-Peñalvo, 2024, p. 5-7).

3.2 BEHAVIOR-DRIVEN DEVELOPMENT (BDD)

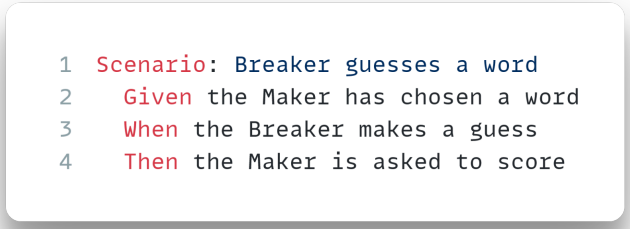
O BDD é uma metodologia ágil que promove a colaboração entre as equipes técnicas e de negócios, fornecendo uma linguagem comum para a especificação de requisitos e cenários de teste. De acordo com o trabalho de Farooq *et al.* (2023, p. 2), o BDD é operado em um nível mais alto do que o TDD, sendo considerado um refinamento dessa metodologia.

É utilizado quando o objetivo é desenvolver testes antes de implementar código-fonte, facilitando o desenvolvimento do *software* quando bem definido. Para tornar esse processo mais acessível, utiliza-se a linguagem de marcação Gherkin, que permite a escrita de testes automatizados em linguagem natural. Esses testes são frequentemente executados com a ferramenta de suporte Cucumber*,* desenvolvida por Aslak Hellesøy, a qual permite a execução desses cenários de teste em linguagem natural, abrindo também um leque de opções mais dinâmicas para a metodologia, como o uso de *Hooks* e mais palavras reservadas (Cucumber, 2024).

3.2.1 Cucumber

Para Cucumber (2024), o Cucumber é uma ferramenta que lê especificações de executáveis escritos em texto simples e verifica se o *software* atende a essas regras. As especificações consistem em múltiplos *examples* ou *scenarios* dentro de um arquivo *feature*, conforme ilustrado na Figura 2.

Figura 2 - Exemplo de cenário BDD para Cucumber



```
Scenario: Breaker guesses a word
  Given the Maker has chosen a word
  When the Breaker makes a guess
  Then the Maker is asked to score
```

Fonte: Adaptado de Cucumber (2024).

Cada cenário é uma sequência de etapas para o Cucumber validar, gerando um relatório que indica o sucesso ou falha da execução dos testes (Cucumber, 2024).

3.2.2 Gherkin

De acordo com a documentação oficial do Cucumber (2024), o Gherkin é um conjunto de regras gramaticais que torna o teste estruturado e simples o suficiente para que o Cucumber compreenda, como desenvolvido no cenário da Figura 2. Sua sintaxe padronizada permite preparar testes automatizados, documentar comportamentos esperados do sistema e realizar especificações de execução para o Cucumber. Além disso, essa gramática oferece um grande suporte linguístico, com mais de 70 idiomas disponíveis, para garantir que a equipe possa utilizar palavras-chave no seu idioma nativo. Os seus documentos são armazenados em arquivos de extensão característica "*.feature*".

3.2.2.1 Feature

No contexto do Cucumber, uma *feature* representa um conjunto de funcionalidades descritas em linguagem natural para documentar requisitos do sistema. Conforme descrito na documentação oficial do Cucumber (2024), um arquivo de feature contém cenários (*scenarios*) que detalham o comportamento esperado do *software*.

3.2.2.2 Scenarios ou Examples

Os *scenarios* (ou *examples*) no Cucumber são exemplos de comportamentos esperados para uma funcionalidade, definidos dentro de um arquivo *feature*. Cada cenário contém uma sequência de *steps* que descrevem um fluxo de interação do usuário com o sistema (Cucumber, 2024).

3.2.2.3 Scenario Outline

O *Scenario Outline* no Cucumber é uma estrutura que permite a reutilização de um cenário com diferentes conjuntos de dados, substituindo variáveis pelos valores definidos em uma tabela de *Examples*. Isso facilita a escrita de testes mais concisos e abrangentes (Cucumber, 2024).

3.2.2.4 Steps

Os *steps* são os blocos fundamentais de um cenário no Cucumber. Eles descrevem as ações que serão executadas no sistema durante a execução do teste, utilizando palavras-chave como *Given*, *When* e *Then* (Cucumber, 2024).

3.2.2.5 Background

O *background* é uma seção opcional em um arquivo *feature* que define um conjunto de passos comuns a todos os cenários do arquivo. Isso evita repetição de código e melhora a legibilidade dos testes (Cucumber, 2024).

3.2.2.6 Rules

As *rules* (regras) permitem agrupar cenários que compartilham um contexto comum e definir comportamentos esperados dentro de determinadas condições de negócio. Segundo a documentação oficial do Cucumber (2024), as regras ajudam a estruturar melhor os requisitos do sistema.

3.2.2.7 Tags

As *tags* no Cucumber são usadas para categorizar e filtrar a execução de cenários. Elas permitem organizar os testes e selecionar quais serão executados, facilitando a automação e o gerenciamento dos casos de teste (Cucumber, 2024).

3.3 TEST SMELL

À medida que os testes crescem em complexidade e volume, surgem desafios relacionados à manutenção e qualidade, como os *test smells* ou *smells* de teste, que, por sua vez, são indicações de potenciais problemas no design e implementação de testes de *software* automatizados, que podem comprometer a clareza, reuso e manutenção, impactando diretamente na eficácia da abordagem BDD (Van Deursen *et al.*, 2001, p. 92). Assim como um *code smell*, o *smell* de teste não significa que há um problema existente, mas um indício de dificuldade adicional que pode estar na manutenibilidade (como duplicação de código), na falta de cobertura (como a ausência ou problemas de verificação) ou em resultados não confiáveis, como um comportamento de execução não determinístico (Soares *et al.*, 2023, p. 1).

3.4 ANTI-PATTERN

Os *anti-patterns* (ou antipadrões) são soluções comuns a problemas recorrentes que, ao contrário dos *design patterns*, resultam em código de baixa qualidade, comprometendo a manutenibilidade, a escalabilidade e a compreensão do *software*. Conforme Brown *et al.* (1998, p. 2), um *anti-pattern* ocorre quando uma solução aparentemente lógica e usual conduz a consequências indesejáveis no desenvolvimento de *software*.

3.5 STAKEHOLDERS

O termo *stakeholder* refere-se a qualquer pessoa, grupo ou entidade que tenha interesse direto ou indireto em um projeto, podendo influenciar ou ser influenciado pelos seus resultados. De acordo com Freeman (1984, p. 46), *stakeholders* incluem clientes, desenvolvedores, gestores e investidores, entre outros, sendo fundamentais para a definição de requisitos e tomadas de decisão no desenvolvimento de *software*.

3.6 GITHUB

O GitHub é uma plataforma de hospedagem de código-fonte e controle de versão baseada no Git, que permite colaboração remota entre desenvolvedores, integração contínua e gestão de repositórios. Segundo Chacon e Straub (2023, p.

166-170), o GitHub fornece ferramentas para revisão de código, automação de fluxos de trabalho e segurança de *software*, sendo amplamente adotado no desenvolvimento ágil.

4 METODOLOGIA

O presente estudo consiste em uma pesquisa aplicada de caráter exploratório e descritivo, que visa identificar e analisar quais *test smells* podem ser encontrados em casos de teste escritos no padrão BDD utilizando a linguagem Gherkin. Inicialmente, foi realizado um estudo detalhado baseado no catálogo de *test smells* proposto por Soares (2023) como ponto de partida. Para complementar e expandir essa base conceitual, foi conduzida uma revisão sistemática da literatura formal e informal, iniciando pela definição do protocolo de pesquisa, incluindo a seleção de palavras-chave principais e seus sinônimos, seguida pela construção de uma string de busca apropriada e escolha criteriosa das bases acadêmicas adequadas. Posteriormente, a aplicação da string de busca será realizada nessas bases, obtendo-se uma coleta inicial dos resultados encontrados. Os resultados serão submetidos a um processo rigoroso de filtragem, resultando na seleção final dos artigos relevantes. A partir dessa seleção, serão extraídas e sintetizadas informações essenciais para responder às questões de pesquisa.

A próxima etapa envolve a coleta de casos reais de teste escritos em Gherkin disponíveis em projetos *open-source* hospedados no GitHub. Para isso, serão estabelecidos critérios específicos para a seleção dos repositórios, além da adaptação do método proposto por Kfouri (2019) para extração automatizada dos casos de teste. Essa extração será acompanhada de um processo organizado de armazenamento e validação dos dados coletados para garantir a qualidade e relevância das informações obtidas.

Uma análise manual minuciosa será realizada em uma amostra representativa dos casos de teste coletados para identificar e categorizar os *test smells* encontrados. Com base nessa análise, será desenvolvido um catálogo atualizado e específico de smells aplicáveis aos casos de testes escritos em Gherkin.

Para garantir a relevância e precisão deste catálogo, será realizada uma validação qualitativa com profissionais experientes da área de qualidade de *software*, por meio da aplicação de um questionário estruturado. Esses especialistas

fornecerão insights adicionais por meio de uma avaliação sistemática dos itens catalogados, contribuindo para a robustez e precisão dos resultados obtidos.

Com o objetivo de automatizar a identificação dos *test smells* catalogados, serão desenvolvidos *scripts* específicos para essa tarefa. Uma amostra independente de testes será separada da ferramenta desenvolvida para possibilitar uma validação rigorosa e objetiva, utilizando métricas quantitativas como precisão, *recall* e *f-measure*. Após validada, a ferramenta será aplicada em todos os repositórios coletados para determinar a frequência dos *test smells* identificados.

Por fim, foi desenvolvida uma plataforma *web* que permitirá a identificação automática dos *test smells* em arquivos *feature*, contando também com uma *dashboard* visual interativa para apresentação clara dos resultados, além de um catálogo interativo que facilitará o acesso detalhado às informações sobre cada *smell* detectado.

Os resultados finais serão apresentados de forma qualiquantitativa, integrando dados secundários provenientes da literatura acadêmica e dados primários coletados diretamente através da análise manual, automatizada e validação com profissionais especializados.

4.1 ANÁLISE DE TEST SMELLS CATALOGADOS

Com o objetivo de estabelecer uma base conceitual sólida e abrangente para a identificação e classificação dos *test smells* que possam estar presentes em casos de teste escritos no padrão BDD utilizando a linguagem Gherkin, foi realizada uma análise aprofundada do catálogo de *test smells* proposto por Soares (2023). Este catálogo representa uma contribuição inédita à literatura, consolidando informações de 127 estudos primários e listando detalhadamente 480 *test smells* distintos.

O processo de análise se iniciou com a revisão criteriosa do catálogo, identificando e compreendendo cada tipo de *test smell*, suas características, contextos de ocorrência e possíveis impactos nos casos de teste. Além disso, foram examinados especificamente os critérios adotados por Soares (2023) para a categorização e agrupamento dos *test smells*.

A análise também incluiu a identificação daqueles *smells* que apresentam maior relevância ou frequência no contexto da escrita de testes em linguagem

Gherkin, priorizando aqueles diretamente aplicáveis ou mais comuns ao cenário de BDD. Com base nessa seleção, foi possível estabelecer uma base sólida para o levantamento e detecção sistemática dos *test smells* nos casos de testes BDD a serem investigados neste trabalho.

Ao final desta etapa, foi obtida uma visão clara e detalhada dos principais *test smells* relevantes ao contexto da pesquisa, permitindo uma aplicação direta e eficiente na fase subsequente de prospecção e detecção dos mesmos em casos de testes reais escritos no padrão BDD com a linguagem Gherkin.

5 REVISÃO MULTIVOCAL DE LITERATURA

O objetivo desta revisão é analisar a literatura para identificar e caracterizar problemas de design e escrita de casos de teste utilizando o padrão BDD, com ênfase em *test smells*. Para tanto, foram considerados tanto estudos formais (artigos revisados por pares) quanto informais (literatura cinzenta, como *blogs*, relatórios técnicos e vídeos). A questão de pesquisa central que orientou o estudo foi: quais *smells* podem ser encontrados em casos de teste escritos no padrão BDD na literatura?

A Revisão Multivocal de Literatura (MLR) adotada neste trabalho diferencia-se das revisões de literatura sistemáticas tradicionais (SLR) por incluir fontes informais, permitindo uma compreensão mais abrangente do estado da prática na indústria, além do estado da arte na academia (Garousi; Felderer; Mäntylä, 2019).

5.1 PROTOCOLO DE PESQUISA

O protocolo de pesquisa foi estruturado em torno de cinco componentes principais: (1) vocabulário adotado, (2) questão de pesquisa, (3) bases de dados consultadas, (4) estratégia de busca e (5) critérios de inclusão e exclusão.

5.2 VOCABULÁRIO

Os estudos foram classificados em duas categorias e ambas as categorias foram consideradas como estudos primários para esta revisão:

- Formais: Artigos revisados por pares publicados em periódicos científicos e anais de conferências;
- Informais: Conteúdos acadêmicos não revisados por pares (monografias, dissertações, teses) e literatura cinzenta (*blogs*, relatórios técnicos, vídeos, páginas da *web*).

5.3 BASES DE DADOS E STRING DE BUSCA

As bases de dados formais utilizadas foram o Google Scholar[1] e o Springer[2], enquanto as informais foram o Google e o Bing. A string de busca foi construída com base na população (BDD) e na intervenção (*test smells* e *anti-patterns*): (”*test**

[1]https://scholar.google.com.br/?hl=pt
[2]https://link.springer.com/

*smell** ” *OR* ”*anti pattern**”) *AND* (”bdd” *OR* ”*behavior driven development*”). O símbolo ”*” foi utilizado para abranger variações lexicais, como pluralizações. A busca foi realizada sem filtros adicionais, considerando todo o texto dos estudos primários.

5.4 CRITÉRIOS DE INCLUSÃO E EXCLUSÃO

Os critérios de inclusão estabelecem as propriedades que um estudo primário deve possuir para ser selecionado, enquanto os critérios de exclusão determinam sua inelegibilidade após a seleção. As seguintes condições foram estabelecidas como critérios de inclusão para a seleção dos estudos primários:

- Estudos redigidos em inglês ou português;
- Estudos acadêmicos revisados por pares;
- Conteúdos acadêmicos não revisados por pares e literatura cinza.

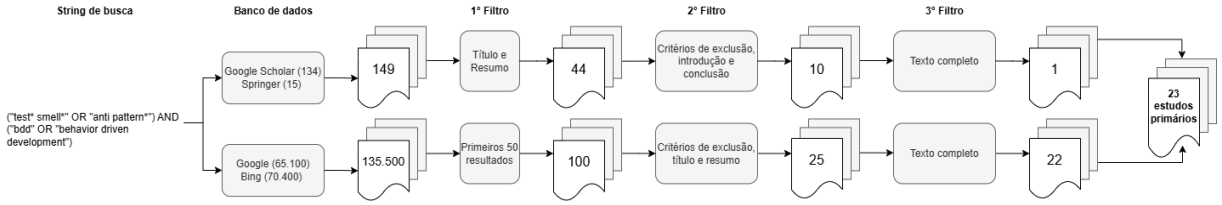
Os seguintes critérios de exclusão foram estabelecidos para filtrar conteúdos que não atendem aos objetivos do estudo, evitando trabalhos irrelevantes, incompletos ou de difícil acesso:

- Estudos cujo conteúdo não seja relevante ou não contribua para os objetivos da pesquisa;
- Artigos da literatura formal que não tenham sido oficialmente publicados;
- Artigos que exigem pagamento para acesso, priorizando aqueles de acesso aberto;
- Artigos com conteúdo incompleto ou indisponível na íntegra.

5.5 SELEÇÃO DOS ESTUDOS PRIMÁRIOS

A Figura 3 apresenta as etapas do processo de seleção dos estudos primários. A string de busca foi utilizada como entrada em pesquisas de bancos formais e informais. Realizada entre os dias 02 de Dezembro de 2024 e 03 de Dezembro de 2024, foram recuperados 149 resultados em bases de dados formais de estudo (Google Scholar e Springer) e aproximadamente 135.500 presentes na *web* em bases informais (Google e Bing). Esses estudos foram submetidos a um processo de filtragem até a seleção final.

Figura 3 - Seleção dos estudos primários



Fonte: Os autores.

A seleção consistiu em dois filtros nas bases de dados formais. O primeiro filtro, consiste na leitura do título e resumo dos artigos buscados, levando em consideração os critérios de inclusão e exclusão. Dos 149 estudos selecionados, apenas 34 passaram pela primeira etapa de filtragem e 10 permaneceram pendentes de análise para a segunda etapa, visto a difícil identificação do tema apenas pelo resumo. No segundo filtro, o número de trabalhos formais foi reduzido para 10 resultados e os materiais pendentes foram todos descartados por não se enquadrar nos critérios definidos. Por fim, no terceiro filtro, todo o conteúdo do material foi analisado, resultando em apenas 1 resultado formal válido para os estudos primários.

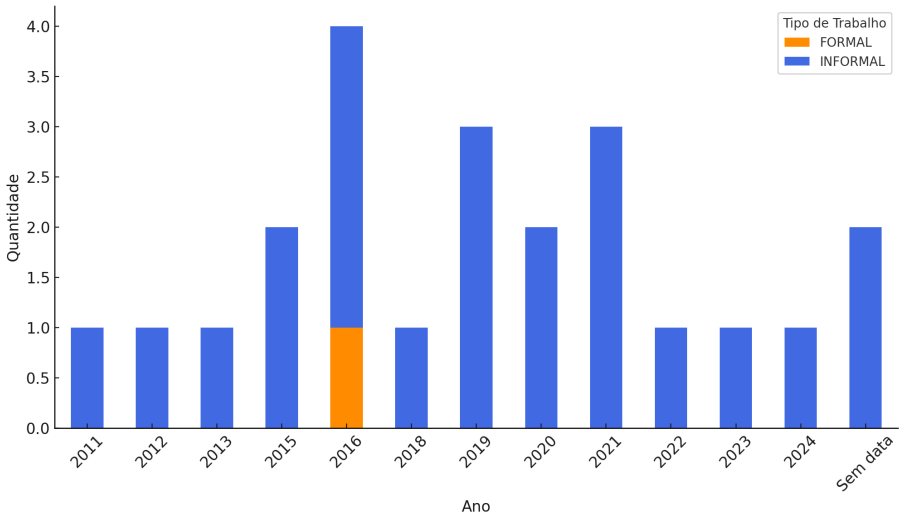
A quantidade de informações recuperadas pelos mecanismos de busca nas bases de dados informais (135.500 resultados) estava muito além da capacidade de análise da equipe de pesquisa. Portanto, seguindo a abordagem de estudos prévios que usaram de abordagem MLR (Kulesovs, 2015; Tom; Aurum; Vidgen, 2013), o primeiro filtro visa reduzir as fontes de pesquisa para uma quantidade de análise viável de 50 resultados, totalizando 100 fontes provenientes da *web*. Cabe ressaltar que esses mecanismos de busca classificam com base na relevância do trabalho, logo os resultados ordenados ao final da busca tendem a ter menor prestígio para esta pesquisa (Kulesovs, 2015; Tom; Aurum; Vidgen, 2013). Os critérios de exclusão, leitura do conteúdo da página e remoção de duplicatas do segundo filtro, proporcionaram um total de 25 fontes informais distintas que atendiam aos critérios de seleção. Analisando o conteúdo completo como proposto no terceiro filtro, os conteúdos relevantes no meio informal totalizaram 22 fontes.

No final do processo, foram selecionados 23 estudos (1 formal e 22 informais), que foram distribuídos em um gráfico de colunas para melhor visualização. Durante as pesquisas, 2 trabalhos informais não apresentaram ano de publicação, e foram devidamente separados em uma coluna própria. Com base nas

informações ressaltadas, pode-se observar na Figura 4 a distribuição dos estudos primários por ano e tipo. A pesquisa não foi limitada por um período específico, visto o número reduzido de resultados.

Percebe-se a presença majoritária dos trabalhos informais dos últimos 9 anos que ainda se mantêm relevantes entre a comunidade, o que torna questionável a ausência de estudos recentes desde esse período e a necessidade de atualização dessas informações. Enquanto que para o único resultado formal de 2016, o questionamento toma uma proporção ainda maior, visto a dificuldade de encontrar respostas confiáveis sobre o tema, revelando um oceano azul inexplorado pelos pesquisadores. Portanto, pesquisas formais a respeito de *smells* no contexto do BDD precisam ser investigadas a fim de contribuir para o avanço da literatura na área de garantia de qualidade e melhorar os padrões de desenvolvimento de casos de teste para essa metodologia.

Figura 4 - Distribuição dos estudos primários por ano



Fonte: Os autores.

5.6 EXTRAÇÃO DOS DADOS

Os dados individuais sobre *test smells* foram extraídos de cada estudo primário. No conjunto de dados fornecido, em relação às publicações, listamos o título, o *link*, o tipo (ou seja, artigo, livro, artigo acadêmico, página da *web*, tese/dissertação, artigo técnico/manuscrito e vídeo de palestra/apresentação) e se o estudo primário utiliza as expressões ”*test smell*” ou ”*anti pattern*”. Em relação aos *test smells* em cada estudo primário, listamos o nome, a definição, as categorias, um

exemplo e uma proposta de solução. No total, foram extraídas 108 ocorrências de 33 *test smells* a partir de 23 estudos primários selecionados.

Notamos que os termos ”test smells” e ”anti-patterns” são utilizados para os mesmos conceitos. A Tabela 1 apresenta a distribuição de ambas as terminologias nos estudos primários selecionados e o número total de estudos primários considerados neste estudo.

Tabela 1 - Terminologia utilizada pelas fontes selecionadas

Terminologia	Informal	Formal	Total
Anti-pattern	21	0	21 (91,3%)
Test Smell	0	1	1 (4,35%)
Nenhuma	1	0	1 (4,35%)
Total	**22 (95,7%)**	**1 (4,3%)**	**23 (100%)**

Fonte: Os autores.

Outro ponto crucial na extração dos dados é a diferenciação entre *test smells* e *code smells*, que limita ainda mais os resultados encontrados pela pesquisa, uma vez que em BDD, não há explicitamente o uso de código, mas sim de linguagem natural para o desenvolvimento dos casos de teste (Rompaey *et al.*, 2007). Muitas publicações sobre *code smells* referem-se a eles conjuntamente como *smell* ou *anti-pattern*, pois ambos ”refletem problemas de design e/ou implementação no código-fonte do *software* que podem ter um impacto negativo na qualidade do *software*” (Tahir *et al.*, 2020). Esse mesmo raciocínio foi utilizado neste trabalho ao mapear estudos com ambas as terminologias (*test smells* e *anti-patterns*), como também realizado por Garousi e Küçük (Garousi; Küçük, 2018) em seu estudo *Multivocal Literature Mapping* (MLM). No entanto, a Tabela 1 mostra que há uma distinção muito bem definida pelo tipo do material analisado, pois em estudos informais encontramos menções apenas ao termo “anti-pattern”, enquanto nos estudos formais apenas o termo “test smell” é citado. Além disso, apenas um estudo informal não utilizou nenhuma das denominações, dado que se tratava de um guia de boas práticas na escrita de casos de teste utilizando o padrão BDD.

Quanto à quantidade de *test smells* descritos por cada fonte, as principais fontes selecionadas foram a postagem de *blog* feito por Theo England (England, 2016), com 13 *test smells*, a postagem de *blog* feito por Thomas Sundberg (Sundberg, 2016), com 12, e a postagem de *blog* de Federico Toledo *et al.* (Toledo *et al.*, 2019), com 7 definições de *test smells*. Nosso conjunto de fontes selecionadas possui uma média de 4,91 *test smells* descritos por fonte, uma mediana de 5 e uma moda de 6.

Em nosso conjunto de dados, optamos por escolher para análise o padrão BDD de escrita de casos de teste por ser amplamente usado em testes manuais de linguagem natural, o que proporcionou uma filtragem do número de resultados relacionados aos tipos de *test smells*, uma vez que com base nos resultados apresentados pela pesquisa, é evidente que ainda se trata de um conteúdo recente na literatura. Visando evitar um número ainda menor de resultados, nenhuma tecnologia foi descrita como prioritária.

5.7 RESULTADOS

A Tabela 2 apresenta os 5 *test smells* mais populares após a atividade de extração dos dados. O total de ocorrências dos *test smells* listados é 59, representando aproximadamente 54,6% do total de ocorrências neste estudo. A lista completa de *test smells* e suas ocorrências pode ser encontrada no **APÊNDICE A**.

É importante observar que, entre os *test smells* analisados, os mais frequentes incluem *UI-Coupled Steps* (23 ocorrências), *Misconception: “BDD is for Testing”* (10 ocorrências) e *Overly Detailed Steps* (10 ocorrências). Outros *test smells* notáveis, como *Lack of Scenario Collaboration* (9 ocorrências) e *Multi-Rule Testing* (7 ocorrências), também aparecem com frequência significativa.

Tabela 2 - Lista dos 5 test smells mais populares

Test Smell	Ocorrências
UI-Coupled Steps	23
Misconception: “BDD is for Testing”	10
Overly Detailed Steps	10
Lack of Scenario Collaboration	9
Multi-Rule Testing	7

Fonte: Os autores.

5.8 AMEAÇAS À VALIDADE DA REVISÃO

Como ameaças à validade interna, foi-se utilizado a definição de *test smells* ou *anti-patterns* — ou a indagação de um estudo que o faça — como critério de seleção para o conjunto de dados primários. Outros critérios de seleção poderiam limitar ou ampliar ainda mais os resultados de modo desnecessário, como o caso do uso da definição “poor quality”, que ocasionou o retorno de trabalhos fora do escopo de tema da pesquisa. Para minimizar essa ameaça, foi conduzida uma análise preliminar que buscava verificar quais os termos mais usados pelos estudos pretendidos para descrição. Além disso, todos os resultados provenientes do Google Scholar e Springer foram devidamente analisados.

Quanto às ameaças à validade da conclusão, no que diz respeito aos nomes alternativos adicionados aqueles *smells* que possuíam apenas descrição ou que foram nomeados de modo informal, há a possibilidade de batismo incorreto. Essa ameaça foi abordada utilizando-se de padrões percebidos em outros estudos da pesquisa para a classificação recorrente de um mesmo *smell* de características compatíveis para completude do dado faltante em discussões entre os avaliadores em reuniões, para melhor nomeação de acordo com a descrição para aqueles *smells* de única ocorrência.

Como ameaças à validade do constructo, a seção 5.6 menciona a presença majoritária de *anti-pattern* como definição para os smells selecionados. Embora possa haver diferença entre tais conceitos na literatura, não foram considerados nos resultados, e a fim de evitar a perda de ocorrências de *smells* ou a incapacidade de realização da pesquisa pela ausência de retorno dos dados, todos os resultados

obtidos foram incluídos. Portanto, como alguns *smells* listados neste trabalho podem ser, na verdade, antipadrões, espera-se que estudos futuros possam estabelecer melhores divisões de contexto entre essas definições, levando em consideração os dados fornecidos por esta revisão como ponto de partida.

Em relação às ameaças à confiabilidade, que dizem respeito à consistência e replicabilidade dos achados, cabe destacar que:

1. Os algoritmos das bases de dados informais alteram os resultados das primeiras 50 aparições à medida que novos resultados são indexados e ponderados às suas bases de dados;
2. Por fim, em relação às ameaças externas à validade, analisando 50 resultados por pesquisa informal não é suficiente para atingir a generalização para os dados acima de 1 milhão. Essa ameaça foi minimizada usando recomendações da literatura que detalham a classificação de algoritmos utilizados pelos motores de busca e informam que a probabilidade de cliques úteis após os 50 resultados é baixa (Kulesovs, 2015; Tom; Aurum; Vidgen, 2013).

6 PROSPECÇÃO E COLETA DE CASOS DE TESTE

Dada a ausência de uma lista publicamente mantida de projetos em código aberto que utilizem Gherkin, optou-se por utilizar o GitHub como fonte principal de coleta de dados. O GitHub é uma plataforma amplamente adotada por desenvolvedores e atualmente contém cerca de 8,5 mil repositórios públicos que utilizam a linguagem Gherkin.

Entretanto, devido à grande quantidade de projetos disponíveis, tornou-se essencial uma filtragem criteriosa para garantir a seleção de repositórios relevantes e evitar ruídos nos dados coletados.

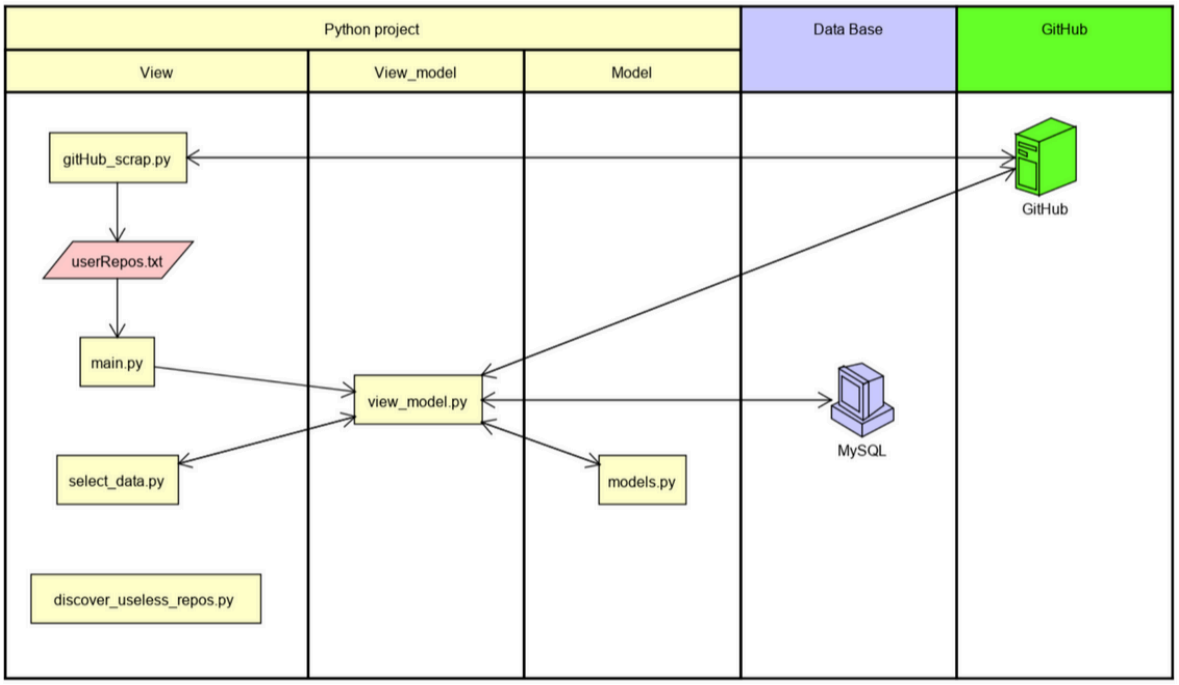
A estratégia de seleção adotada envolveu a aplicação de filtros baseados na linguagem do projeto e sua popularidade, medida pelo número de estrelas (*stars*). Essa pesquisa foi realizada em 15 de Dezembro de 2024[3], resultando na identificação de 7 projetos que atendiam aos critérios estabelecidos. A decisão de limitar a busca a projetos com pelo menos 500 estrelas teve como objetivo garantir que apenas repositórios com impacto significativo na comunidade fossem analisados.

A mineração e extração dos dados desses repositórios foram realizadas utilizando uma adaptação do minerador[4] proposto por Kfouri (2019). Esse minerador foi ajustado para retomar suas operações normais para os cenários descritos em Gherkin. O processo de mineração levou aproximadamente um dia para ser concluído, considerando todas as etapas descritas no fluxo do projeto (apresentado na Figura 5).

Os repositórios selecionados passaram, então, por uma análise detalhada, com o objetivo de identificar padrões e anomalias estruturais relacionadas a *test smells* em cenários BDD escritos com Gherkin.

[3]usando (language:Gherkin stars:>=500) no GitHub
[4]https://github.com/ggpsgeorge/github-repo-scrapping-clustering

Figura 5 - Esquema do projeto de mineração de repositórios BDD



Fonte: Kfouri (2019).

6.1 COLETA E SELEÇÃO DE REPOSITÓRIOS

O fluxo do projeto inicia-se com a execução do script “github_scrap.py”, presente na camada de “View”, um arquivo “[...] que procura no GitHub por nomes de repositórios usando uma tag, no caso BDD, como pesquisa e ordenando por número de estrelas” (Kfouri, 2019, p. 18).

Quando iniciado, o *script* solicita uma entrada do usuário referente à página inicial. Em seguida, o programa executa um *loop* que realiza a busca em 9 páginas, respeitando um intervalo de 120 minutos, conforme as limitações impostas pela *Application Programming Interface* (API) do GitHub (Kfouri, 2019, p.18). O Código-Fonte 1 apresenta o código adaptado do arquivo.

Código-Fonte 1 - Composição do arquivo “github_scrap.py”

```
import requests
import time

class GitHub():

  def __init__(self, url):
    self.url = url
```

```
# Variable that defines the min and max pages in the url
# to scrap (API only accepts 9 pages per minute)

inipage = int(input("Ini: "))
finpage = inipage + 9

while finpage < 11:

  f = open("usersReposp" + str(inipage) + "_" + str(finpage-1) + ".txt", "w")

  url =
"https://github.com/search?q=stars%3A%3E%3D500+language%3AGherkin&type=repositories&ref=a
dvsearch&p="

  for j in range(inipage, finpage):
    response = requests.get(url + str(j))
    data = response.json()

    if 'results' in data['payload']:
      all_results = data['payload']['results']
      print(f'Pág. {j}: {len(all_results)} resultados encontrados')

      for result in all_results:
        repo_url = f'/{result['repo']['repository']['owner_login']}/{result['repo']['repository']['name']}'
        f.write(repo_url + "\n")
    else:
      print(f'Nenhum resultado encontrado na página {j}')

  f.close()

  print("Ini: %d Fin: %d\n" % (inipage, finpage))

  time.sleep(120)

  inipage = finpage
  finpage = inipage + 9
```

Fonte: Adaptado de Kfouri (2019).

Algumas adaptações foram realizadas em relação ao modelo proposto por Kfouri (2019), visando otimizar o processo de coleta de dados pela redução do número de iterações, pois todos os resultados estão concentrados em uma única página e um ajuste na URL de busca para aplicar os critérios de filtragem mencionados anteriormente.

“Para cada iteração, o programa cria um arquivo ‘usersRepospX_Y.txt’, onde X é a página inicial e Y é a página final. Como exemplo, o primeiro arquivo criado será chamado ‘userReposp1_9.txt’” (Kfouri, 2019, p. 19).

A escrita dos resultados no arquivo de texto foi realizada em formato JavaScript *Object Notation*[5] (JSON), substituindo a abordagem anterior que utilizava a biblioteca BeautifulSoup para extração dos dados. As informações do arquivo “userReposp.txt” serão utilizadas posteriormente para complementar o caminho das requisições à API. O Quadro 1 apresenta as informações dos projetos capturados neste estudo.

Quadro 1 - Resultados da coleta dos nomes dos repositórios

Nome do proprietário	Nome do repositório
sdkman	sdkman-cli
thoughtbot	factory_bot_rails
inukshuk	jekyll-scholar
opencypher	openCypher
keygen-sh	keygen-api
serverlessworkflow	specification
scality	Zenko

Fonte: Os autores.

Por meio do arquivo “github_scrap.py”, foi possível capturar um total de 7 repositórios públicos que utilizavam a linguagem Gherkin e atendiam aos critérios de 500 estrelas de popularidade.

A coleta dos dados dos repositórios é realizada pelo *script* “main.py”, que executa a extração e o armazenamento das informações obtidas via requisições à

[5]https://www.json.org/json-en.html

API do GitHub. A primeira etapa do processo consiste na adição do *token* de autenticação, que pode ser facilmente gerado pelo GitHub.

> Esse token é necessário, pois o GitHub normalmente só permite, no máximo, 60 requisições por hora do mesmo endereço de IP, quantidade insuficiente para as nossas necessidades. Com o token, a quantidade máxima de requisições aumenta para 5000 por hora, atendendo às nossas necessidades (Kfouri, 2019, p. 20)

O programa “main.py” é responsável por ler os dados extraídos anteriormente pelo “github_scrap.py”, que foram armazenados no arquivo “userReposp.txt”. O programa percorre cada linha do arquivo e, para cada repositório listado, realiza uma requisição *Hypertext Transfer Protocol* (HTTP) à API do GitHub.

A resposta da API é um JSON contendo as informações detalhadas do repositório, que são extraídas e armazenadas no banco de dados, populado com as respectivas tabelas “Repository”, “Feature”, “Scenario” e “Step” (Kfouri, 2019, p. 20-21). A Código-Fonte 2 apresenta a estrutura adaptada do código presente no arquivo “main.py”.

Código-Fonte 2 - Composição do arquivo “main.py” do minerador

```
from view_model import ViewModel
import time


# 1 - Initialising things ================================================

token = [GITHUB_TOKEN]

view_repository = ViewModel(token)

# 2 - opening document with repositories to search ========================

path_api = "https://api.github.com/repos"

archives = []
for i in range(1, 11, 9):
  archive = f'usersReposp{i}_{i+8}.txt'
  archives.append(archive)

for archive in archives:
```

```
start_time = time.time()
f = open(archive, "r")
users = f.read().splitlines()
f.close()

# 3 - searching repositories and saving them on objects ======================

print("Starting downloading repositories from " + archive)

paths = []
for user in users:
  paths.append(path_api + user)

problem_paths = []
for path in paths:
  try:
    print("Downloading repository from path: " + path)
    repository = view_repository.getRepositoryFromPath(path)
    print("Saving repository " + path + " on BD")
    if repository.features != None:
      view_repository.saveRepositoryOnDB(repository)
    time.sleep(120)
  except:
    problem_paths.append(path)
    print("There was a problem with the repository from path: " + path)

print(".....%d seconds....." % (time.time() - start_time))

print("These repositories had problems:")

for path in problem_paths:
  print(path)
```

Fonte: Adaptado de Kfouri (2019).

Foi realizada uma pequena otimização na segunda etapa do programa, visando reduzir o tempo de processamento e melhorar a eficiência da coleta. Inicialmente, o programa era executado várias vezes para cada arquivo de texto gerado. Em testes anteriores, observou-se a necessidade de poupar tempo nesse

processo, implementando uma lista com os índices de todos os arquivos de texto e, a partir dela, popular a lista com os repositórios coletados.

Na terceira e última etapa, há um tratamento de erro durante a coleta e armazenamento dos dados. Caso haja algum problema com um repositório, o seu nome é registrado no terminal para permitir a recuperação manual posteriormente.

Entretanto, mesmo com a primeira filtragem, ainda é possível que existam repositórios com a *tag* BDD que não possuem nenhum arquivo de extensão ".feature", tornando-os irrelevantes para a pesquisa (Kfouri, 2019, p. 21).

Ainda no estudo de Kfouri (2019, p. 21), é importante ressaltar que a classe "view_repository", pertencente ao arquivo "view_model.py", possui os métodos de coleta e armazenamento dos repositórios no banco de dados, que são respectivamente "getRepositoryFromPath(path)" e "saveRepositoryOnDB(repository)". A Tabela 3 apresenta a quantidade de registros salvos no banco de dados.

Tabela 3 - Resultados da coleta dos dados dos repositórios

Entidade	Registros
Repository	7
Feature	372
Scenario	4040
Step	33451

Fonte: Os autores.

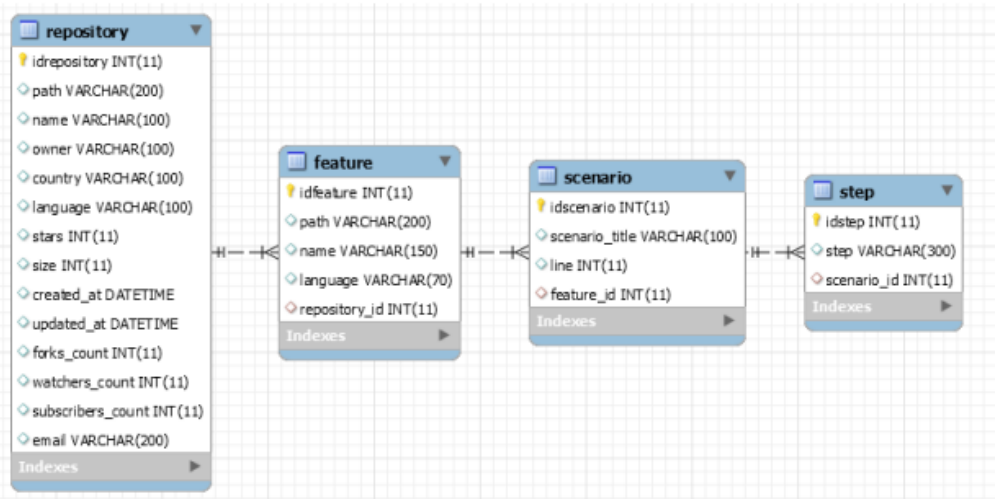
Embora a pesquisa tenha se baseado em apenas 7 repositórios, o número de *features*, *scenarios* e *steps* registrados é expressivamente alto. Esse volume reforça a percepção de que os projetos analisados utilizam o BDD de forma consistente, aplicando testes escritos em Gherkin de maneira extensiva para especificação e validação de suas funcionalidades.

No entanto, para garantir que a presença do BDD seja efetiva e não apenas um resquício de processos passados, foi implementado um critério de seleção baseado no *delta* de atualização das features, conforme descrito no tópico 6.3.

6.2 ARMAZENAMENTO E ESTRUTURAÇÃO DOS DADOS

Para armazenar os dados coletados, Kfouri (2019) utilizou o Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL, cuja estrutura é representada pelo modelo relacional ilustrado na Figura 6.

Figura 6 - Modelo Entidade-Relacionamento do minerador



Fonte: Kfouri (2019).

O arquivo “view_model.py”, localizado na camada “View_model” da arquitetura proposta por Kfouri (2019) (Figura 5), manteve a sua estrutura original. Entretanto, algumas alterações foram necessárias no método de conexão com o banco de dados, pois neste estudo foi optado o uso de um *container* Docker para manter o banco de dados. O Código-Fonte 3 apresenta a adaptação do método de conexão com o banco de dados do arquivo “view_model.py”.

Código-Fonte 3 - Método de conexão do arquivo “view_model.py”

```
def __init__(self, token):
  self.token = token
  self.features = []
  self.num_files = 0
  self.num_func = 0
  try:
    self.engine =
create_engine('mysql://root:[DATABASE_PASSWORD]@localhost:[PORT]/[DATABASE_NAME]',
echo=False)
    Session = sessionmaker(bind=self.engine)
    session = Session()
```

```
  session.query(1).from_statement(text('SELECT 1')).all()
  session.close()
  print("Connected to Data Base")
except Exception as e:
  print(f"Not connected to Data Base: {e}")
```

Fonte: Adaptado de Kfouri (2019).

As principais modificações incluem ajustes na URL do método “create_engine()”, que agora requer senha estabelecida para o banco, porta do *container* e nome do banco de dados em execução.

Além disso, o tratamento de exceções, agora é exibido diretamente no terminal, permitindo ao desenvolvedor diagnosticar o problema mais rapidamente.

Para replicar o modelo relacional apresentado na Figura 6 e garantir a compatibilidade com o Código-Fonte 3, o Código-Fonte 4 apresenta o comando utilizado para criar o *container* em um ambiente Linux.

Código-Fonte 4 - Comando para criação do *container*

```
docker run -d \
  --name [CONTAINER_NAME] \
  -e MYSQL_ROOT_PASSWORD=[DATABASE_PASSWORD] \
  -e MYSQL_DATABASE=[DATABASE_NAME] \
  -p [CONTAINER_PORT]:3306 \
  mysql:latest
```

Fonte: Os autores.

O uso do Docker permite maior flexibilidade, garantindo que o banco de dados seja executado em um ambiente isolado e controlado, facilitando a replicação do estudo em diferentes máquinas e cenários.

6.3 CRITÉRIOS DE SELEÇÃO DE PROJETOS

Segundo Kfouri (2019, p. 25), uma etapa essencial dessa prospecção é a verificação da consistência na utilização do BDD nos repositórios identificados.

Embora muitos projetos no GitHub possuam a *tag* BDD, nem todos utilizam efetivamente essa abordagem. Alguns repositórios podem conter a *tag*, mas não

possuem arquivos *feature* registrados ou não apresentam atualizações regulares nas etapas do processo BDD.

Em seu trabalho, o Kfouri (2019) utilizou um *Open Source Software* (OSS)[6] que também será empregado neste trabalho. Esse *software* é responsável por calcular a diferença entre a data da última versão do código-fonte do repositório e a última atualização do diretório de *features*. Essa diferença entre as datas é denominada *delta* e é medida em dias.

O objetivo desse critério de seleção é determinar se o projeto realmente utiliza BDD ou se apenas iniciou esse processo, mas abandonou sua implementação ao longo do tempo. Para isso, definiu-se um *delta* limite de 365 dias, ou seja, se um repositório tiver um delta superior a 365 dias, ele será descartado da análise, pois indica que o desenvolvimento das *features* BDD não está mais ativo. Os resultados desse critério de seleção podem ser consultados na Tabela 4.

Tabela 4 - Resultados do critério de seleção dos dados

Nome do repositório	*Delta*
sdkman-cli	0
factory_bot_rails	0
jekyll-scholar	0
openCypher	-49
keygen-api	-22
specification	-1466
Zenko	-2005

Fonte: Os autores.

Valores positivos próximos ou iguais a zero (por exemplo, em sdkman-cli, factory_bot_rails e jekyll-scholar) indicam que o código-fonte está sendo atualizado junto ou pouco antes dos testes BDD, sugerindo que os testes acompanham a evolução do projeto.

Valores negativos (por exemplo, em openCypher e em Zenko) indicam que os arquivos de testes BDD estão sendo atualizados mais recentemente do que o

[6]https://github.com/RafaelFazzolino/test_pydriller

código-fonte. Isso sugere que a metodologia BDD continua ativa no projeto, mesmo quando o código principal não recebe novas implementações.

Essa observação reforça a hipótese de que os repositórios utilizam o BDD de maneira contínua e independente das atualizações no código-fonte, indicando um possível uso de testes como documentação viva ou validação recorrente do sistema.

6.4 VALIDAÇÃO DOS DADOS

Para garantir a integridade e a consistência dos dados armazenados, foi estabelecida uma consulta de validação com o objetivo de identificar e eliminar possíveis registros inválidos que passaram pelas etapas anteriores de filtragem. O Código-Fonte 5 apresenta a consulta SQL (*Structured Query Language*) utilizada para essa validação.

Código-Fonte 5 - Consulta de validação dos dados

```
SELECT
  project_population,
  project_stars,
  project_language,
  project_features,
  project_scenarios,
  project_steps,
  SUM(project_features) OVER () AS total_features,
  SUM(project_scenarios) OVER () AS total_scenarios,
  SUM(project_steps) OVER () AS total_steps
FROM (
  SELECT
    CONCAT(r.owner, '_', r.name) AS project_population,
    r.stars AS project_stars,
    r.language AS project_language,
    COUNT(DISTINCT f.idfeature) AS project_features,
    COUNT(DISTINCT s.idscenario) AS project_scenarios,
    COUNT(step.idstep) AS project_steps
  FROM
    repository r
  JOIN feature f ON r.idrepository = f.repository_id
  JOIN scenario s ON f.idfeature = s.feature_id
  JOIN step ON s.idscenario = step.scenario_id
```

```
  GROUP BY project_population, project_stars, project_language
) AS subquery;
```

Fonte: Os autores.

A consulta interliga as tabelas do banco de dados por meio de *joins*, garantindo que todos os dados armazenados possuam referências válidas. O critério de validação segue a seguinte lógica:

1. Um projeto é considerado válido se possuir pelo menos uma *feature* cadastrada;
2. Uma *feature* é considerada válida se estiver vinculada a um projeto válido e possuir pelo menos um *scenario* cadastrado;
3. Um *scenario* é considerado válido se estiver vinculado a uma *feature* válida e possuir pelo menos um *step* cadastrado;
4. Um *step* é considerado válido se estiver vinculado a um *scenario* válido.

Essa abordagem permite detectar registros órfãos ou dados inconsistentes, garantindo que apenas projetos estruturados corretamente sejam considerados na análise.

Esse último processo de filtragem resultou na eliminação de 29 *features* e 2 *scenarios* do repositório OpenCypher, que descumpriam os critérios de validação. As *features* descartadas eram arquivos vazios, contendo apenas título, sem qualquer conteúdo relevante (exemplificado no Código-Fonte 6). Enquanto os *scenarios* inválidos continham todo o seu conteúdo comentado, impossibilitando sua execução e validação (conforme o Código-Fonte 7). A Tabela 5 apresenta o quantitativo final de registros disponíveis no banco de dados após essa filtragem.

Código-Fonte 6 - Exemplo de *feature* descartada

```
#
# Copyright (c) 2015-2024 "Neo Technology,"
# Network Engine for Objects in Lund AB [http://neotechnology.com]
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
```

```
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Attribution Notice under the terms of the Apache License 2.0
#
# This work was created by the collective efforts of the openCypher community.
# Without limiting the terms of Section 6, any Derivative Work that is not
# approved by the public consensus process of the openCypher Implementers Group
# should not be described as “Cypher” (and Cypher® is a registered trademark of
# Neo4j Inc.) or as "openCypher". Extensions by implementers or prototypes or
# proposals for change that have been documented or implemented should only be
# described as "implementation extensions to Cypher" or as "proposed changes to
# Cypher that are not yet approved by the openCypher community".
#

#encoding: utf-8

Feature: Aggregation4 - Avg
```

Fonte: Dados da pesquisa.

Código-Fonte 7 - Exemplo de *scenario* descartado

```
### Error detail not necessarily recognizable
#  @NegativeTest
#  Scenario: [15] Fail oning an hexadecimal literal containing a invalid symbol character
#    Given any graph
#    When executing query:
#      """
#      RETURN 0x1A2b3c4#5E6f7 AS literal
#      """
#    Then a SyntaxError should be raised at compile time: InvalidNumberLiteral
```

Fonte: Dados da pesquisa.

Tabela 5 - Quantitativo final de registros disponíveis para amostra

Entidades	Total	Válidos	Descartes	Percentual de perda
Repository	7	7	0	0%
Feature	372	343	29	7,80%
Scenario	4040	4038	2	0,05%
Step	33451	33451	0	0%

Fonte: Os autores.

A começar pelas *features*, houve uma perda de 7,80% dos registros totais. As informações presentes no Código-Fonte 6 sugerem que os arquivos ainda não possuem implementação e estão aguardando casos de teste. Quanto ao impacto mínimo nos *scenarios*, com apenas 2 registros eliminados que representam 0,05% do total, é notório que pouquíssimos arquivos possuíam resíduos de casos de teste que não deveriam estar mais presentes no projeto.

6.5 ANÁLISE DOS DADOS DESCARTADOS

A filtragem não comprometeu significativamente a representatividade dos dados, confirmando que as etapas anteriores realizadas no trabalho de Kfouri (2019) e pelos autores deste trabalho, foram bem definidas.

Todavia, para verificar se a remoção das instâncias inválidas era justificável, seguiu-se a criação de um projeto com base na documentação oficial Cucumber (2024), onde houve a filtragem de *features* defeituosas com as mesmas características das descartadas no estudo, seguida pela execução dos testes com o auxílio do arquivo "RunCucumberTest.java", conforme representado no Código-Fonte 8.

Código-Fonte 8 - Composição do arquivo "RunCucumberTest.java"

```
package hellocucumber;

import org.junit.platform.suite.api.ConfigurationParameter;
import org.junit.platform.suite.api.IncludeEngines;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.platform.suite.api.Suite;
```

```
import static io.cucumber.junit.platform.engine.Constants.*;

@Suite
@IncludeEngines("cucumber")
@SelectPackages("hellocucumber")
@ConfigurationParameter(key = PLUGIN_PROPERTY_NAME, value = "pretty")
@ConfigurationParameter(key = FEATURES_PROPERTY_NAME, value =
"src/test/resources/hellocucumber/test.feature")
public class RunCucumberTest {
}
```

Fonte: Os autores.

A chave “FEATURES_PROPERTY_NAME”, foi utilizada para filtrar e executar apenas o arquivo defeituoso. O resultado pode ser observado no Código-Fonte 9.

Código-Fonte 9 - Resultado de execução forçada de *feature* vazia

```
org.junit.platform.suite.engine.NoTestsDiscoveredException: Suite [hellocucumber.RunCucumberTest]
did not discover any tests

        at java.base/java.util.stream.ForEachOps$ForEachOp$OfRef.accept(ForEachOps.java:184)
        at java.base/java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:212)
        at java.base/java.util.Iterator.forEachRemaining(Iterator.java:133)
        at java.base/java.util.Spliterators$IteratorSpliterator.forEachRemaining(Spliterators.java:1939)
        at java.base/java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:556)
        at java.base/java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:546)
        at
java.base/java.util.stream.ForEachOps$ForEachOp.evaluateSequential(ForEachOps.java:151)
        at
java.base/java.util.stream.ForEachOps$ForEachOp$OfRef.evaluateSequential(ForEachOps.java:174)
        at java.base/java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:265)
        at java.base/java.util.stream.ReferencePipeline.forEach(ReferencePipeline.java:611)
```

Fonte: Os autores.

O erro retornado pelo Cucumber confirma que a ferramenta não executa features vazias, pois não há testes presentes. A mesma validação foi realizada removendo a *tag* de filtragem, onde o Cucumber ignorou completamente os arquivos

inválidos e processou apenas os que possuíam cenários. Esse mesmo teste foi aplicado para os cenários de teste comentados, resultando no mesmo comportamento.

7 ANÁLISE PRELIMINAR E CATEGORIZAÇÃO DOS TEST SMELLS

Esta etapa segue um processo baseado em amostragem estatística, seleção aleatória de casos de teste e categorização detalhada dos *smells* identificados, diretamente para a resposta da questão de pesquisa QP1: *"Quais smells podem ser encontrados em casos de teste escritos no padrão BDD com a linguagem Gherkin?"*.

7.1 AMOSTRAGEM E SELEÇÃO DOS CASOS DE TESTE

A determinação do tamanho da amostra foi realizada com base na fórmula para dados categóricos de Cochran (Ii; Kotrlik; Higgins, 2001). Os parâmetros utilizados foram uma taxa de confiabilidade de 80% (i.e., *z-score* 1.28) e margem de erro de 5% considerando todos os *scenarios* presentes na base de dados do estudo. O que resultou em uma população de 158 *scenarios* que seriam utilizados para validação manual dos dados. Para escolha dos casos de teste a serem analisados, foi utilizada uma consulta que seleciona randomicamente os cenários armazenados na base de dados. O Código-Fonte 10 apresenta a estrutura da consulta randômica.

Código-Fonte 10 - Consulta randômica dos cenários de teste

```
SELECT SUBSTRING(f.path, 89) AS feature_reference,
    ROW_NUMBER() OVER (PARTITION BY f.idfeature ORDER BY s.line) AS scenario_position,
    s.scenario_title,
    count(step.scenario_id) AS number_steps
FROM scenario s
JOIN feature f on s.feature_id = f.idfeature
JOIN step on s.idscenario = step.scenario_id
GROUP BY feature_reference, step.scenario_id
ORDER BY RAND()
LIMIT 158;
```

Fonte: Os autores.

A consulta retorna o nome da *feature* de referência, posição, título e contagem de passos do cenário de teste sorteado. A Tabela 6 detalha a distribuição das amostras de *scenarios* por projeto.

Tabela 6 - Distribuição das amostras de *scenarios* por projeto

Nome do repositório	Amostra	Total
sdkman-cli	6 (3,80%)	158 (3,91%)
factory_bot_rails	1 (0,63%)	25 (0,62%)
jekyll-scholar	7 (4,43%)	145 (3,59%)
openCypher	59 (37,34%)	1640 (40,61%)
keygen-api	82 (51,90%)	2004 (49,63%)
specification	2 (1,27%)	21 (0,53%)
Zenko	1 (0,63%)	45 (1,11%)
Total	**158 (100%)**	**4038 (100%)**

Fonte: Os autores.

Projetos maiores, como keygen-api e openCypher, representam 90,24% do total de cenários. Isso sugere que esses repositórios possuem um uso mais intensivo de testes BDD e consequentemente uma maior participação na amostra, representando quase 90% dela. Enquanto projetos menores, como factory_bot_rails e specification, têm uma participação mínima na amostra, o que significa a utilização do BDD de forma mais restrita ou menos documentada.

Além disso, como restrição adicional, foi considerada a validação dos dados descrita no Código-Fonte 5, garantindo que apenas cenários válidos fossem incluídos na amostra. Durante a análise, observou-se a necessidade de não somente investigar o *scenario* sorteado, mas também os recursos presentes no arquivo como um todo de forma geral, que pudessem interagir com o cenário da amostra. A intenção desta verificação é auxiliar no processo de prospecção e detecção dos *smells*. A distribuição dos cenários sorteados para análise preliminar, pode ser consultada no **APÊNDICE B**.

7.2 IDENTIFICAÇÃO E CATEGORIZAÇÃO DE TEST SMELLS

Os *smells* identificados foram classificados de acordo com as características estruturais da linguagem Gherkin, usando critérios da composição da *feature* (por exemplo, títulos, *tags*, *steps* etc.). A categorização teve como objetivo facilitar o diagnóstico e a identificação de padrões, permitindo uma validação mais ágil e

precisa por parte dos pesquisadores. Cada categoria definida contém um nome apropriado e descrição detalhada que ajuda na identificação do *smell*. A amostra propôs um total de 12 *smells* distintos identificados em mais de um cenário, que serão detalhados a seguir.

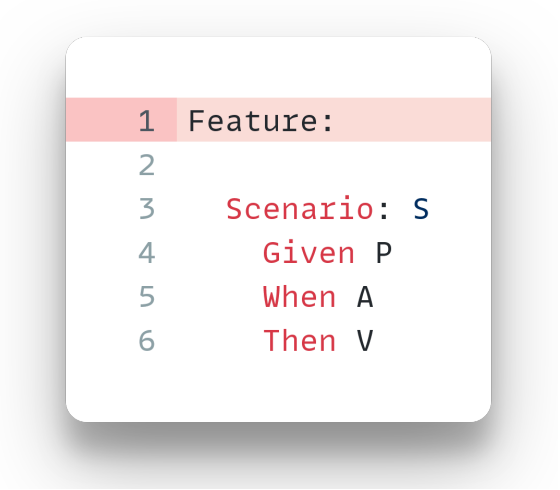
7.2.1 Title Smells

Possuem como característica principal a incoerência no título da *feature* ou do cenário de teste, tornando-os vagos, ambíguos ou pouco descritivos. O que compromete a compreensão dos testes e dificulta a manutenção. Esta foi uma das categorias que mais obteve variações de ocorrência, tendo um total de 3 *smells* identificados. Os defeitos se restringem principalmente a *features* e *scenarios*, dos quais são os únicos capazes de possuir títulos.

7.2.1.1 Untitled Feature

Acontece quando uma funcionalidade não possui nenhum título anexado para identificação do seu propósito. A Figura 7 ilustra a regra de identificação deste *smell*, seguida pelo Código-Fonte 11 com um exemplo de *Untitled Feature* encontrado na amostra.

Figura 7 - Regra para *Untitled Feature*



Fonte: Os autores.

Dado que:

1) S é um título de cenário
2) P é uma pré-condição

3) A é uma ação

4) V é uma verificação

Percebe-se que apesar de possuir uma declaração obrigatória, a *feature* não é necessariamente nomeada, o que pode causar ambiguidade**,** quando o objetivo da funcionalidade fica pouco claro; rastreabilidade reduzida**,** caracterizada pela dificuldade de associação entre cenários e requisitos; desalinhamento e comunicação falha**,** pelo risco de divergência entre intenção e implementação, prejudicando o entendimento entre equipes.

Código-Fonte 11 - Demonstração de *Untitled Feature*

```
Feature:
  In order to not have to manually configure Factory Bot as the Rails testing fixture replacement by
using the --fixture-replacement=factory_bot option
  I would like the Factory Bot Rails gem to configure Factory Bot as the fixture replacement.
```

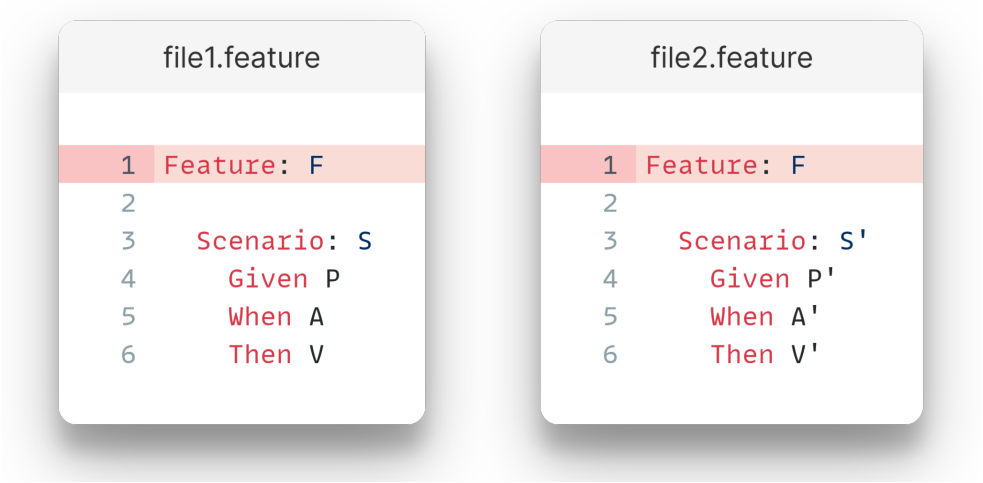
Fonte: Dados da pesquisa.

Ainda que a funcionalidade contenha informações descritivas, o título não consta em sua composição. Em testes realizados, percebeu-se que para o Cucumber não há problema de execução. Nesse caso, a ferramenta não depende do nome da *feature* para apresentar os resultados de execução dos testes. Quando há a necessidade de se atribuir um título, a ferramenta preenche essa lacuna internamente com um nome genérico “Feature”, para apresentar ao testador os cenários que passaram naquele arquivo. De maneira geral, as consequências deste *smell* são exclusivamente voltadas aos testadores humanos.

7.2.1.2 Duplicate Feature Title

Também foram encontradas *features* distintas que possuíam os mesmos títulos na amostra de *scenarios*. A Figura 8 demonstra a regra de identificação do *smell*.

Figura 8 - Regra para *Duplicate Feature Title*



Fonte: Os autores.

Dado que:

1) F é um título de *feature*

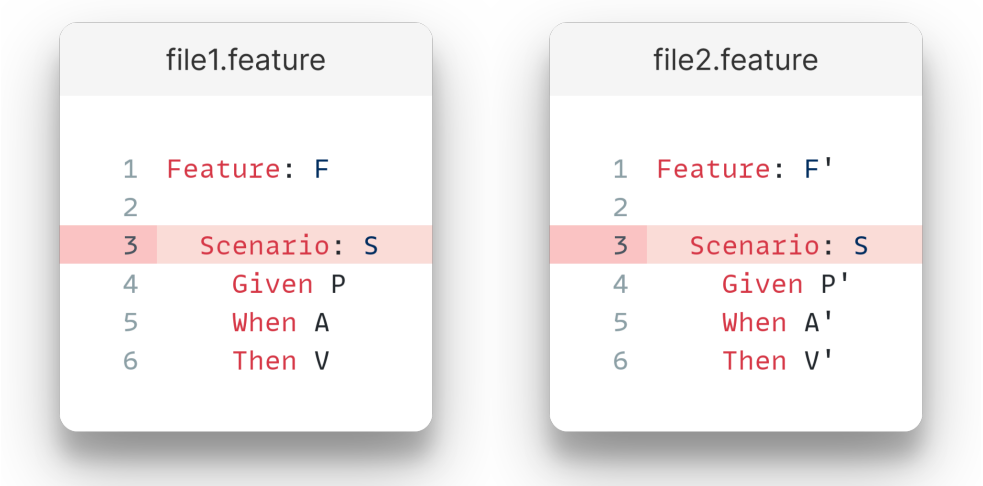
As consequências desse *smell* também afetam os testadores pela confusão de escopo**,** dificulta a distinção entre funcionalidades diferentes; rastreabilidade prejudicada, complicando a identificação de qual funcionalidade está sendo avaliada; interpretação incorreta**,** que aumenta o risco de erros de análise dos resultados dos testes.

Assim como o *Untitled Feature*, o Cucumber não enxerga problemas de execução para este caso, sendo indiferente para a ferramenta a apresentação dos títulos sem quaisquer diferenciações que ajudem o testador a identificá-los separadamente, cabendo apenas à análise da sua composição.

7.2.1.3 Duplicate Scenario Title

Partindo para uma abordagem semelhante, mas no contexto dos *scenarios* dentro e fora da mesma funcionalidade, a amostra demonstrou que também há registros desse tipo com os mesmos títulos. A Figura 9 detalha a regra de identificação do *Duplicate Scenario Title*.

Figura 9 - Regra para *Duplicate Scenario Title*



Fonte: Os autores.

As consequências desse *smell* envolvem confusão nos resultados**,** que dificulta o entendimento de qual cenário foi executado ou falhou; inconsistência, dificultando entender qual é o comportamento esperado; rastreabilidade comprometida**,** que dificulta ligar cenários aos requisitos.

Para essa situação, há confusão da ferramenta para apresentação dos dados, pois é impossível executar os casos de teste isoladamente sem o uso de filtros extremamente rigorosos, que só funcionam quando os títulos se repetem fora da mesma *feature*. Este *smell* gera problemas não só para os testadores, mas também dificulta a execução dos testes por parte da ferramenta, afetando o seu desempenho.

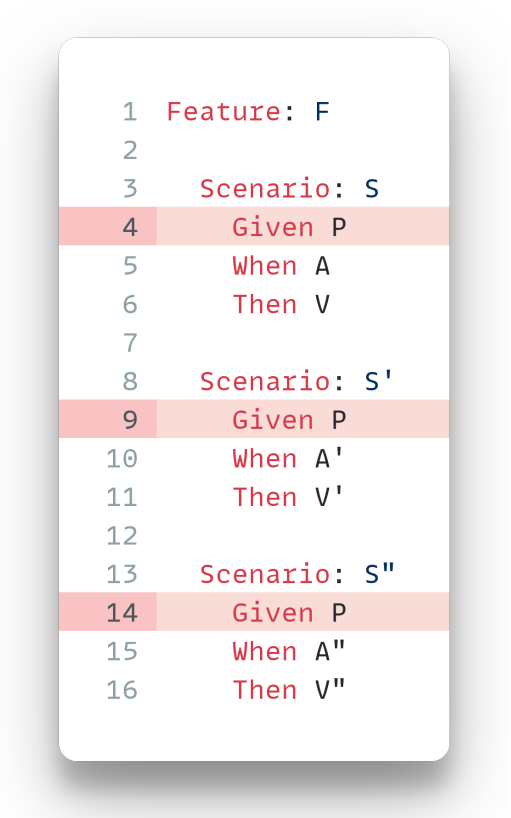
7.2.2 Background Smells

Marcados pelo uso inadequado da estrutura *Background*, como um excesso de informações irrelevantes, dependência de dados que variam entre cenários ou ausência em casos de utilização apropriada. Esta categoria teve 1 único registro de *smell* que se mostrou bastante recorrente nos cenários da amostra. Por se tratar de um componente utilizado apenas uma vez em cada *feature*, suas variações são limitadas, dependendo em sua maioria de contexto.

7.2.2.1 Absence of Background

Definido quando uma *feature* com múltiplos casos de teste contém pré-condições comuns repetidas desnecessariamente. O que poderia ser simplificado com o uso da estrutura *Background*. O uso exacerbado das mesmas pré-condições na maioria dos testes também indica um *Background* mal implementado em uma funcionalidade ou presença de *scenarios* fora do contexto da *feature*. A Figura 10 apresenta as regras para identificação deste *smell*.

Figura 10 - Regra para *Absence of Background*



```
1  Feature: F
2
3    Scenario: S
4      Given P
5      When A
6      Then V
7
8    Scenario: S'
9      Given P
10     When A'
11     Then V'
12
13   Scenario: S"
14     Given P
15     When A"
16     Then V"
```

Fonte: Os autores.

Este processo corresponde a redundâncias de código, aumentando de forma desnecessária o tamanho do arquivo; há também redução de clareza, visto que passos repetitivos desviam o foco da lógica de teste; dificuldade de manutenção, pois há a necessidade de se atualizar todos os cenários em virtude da ocorrência de atualizações.

A ausência de um *Background* não resulta em interferência para a execução dos testes pelo Cucumber.

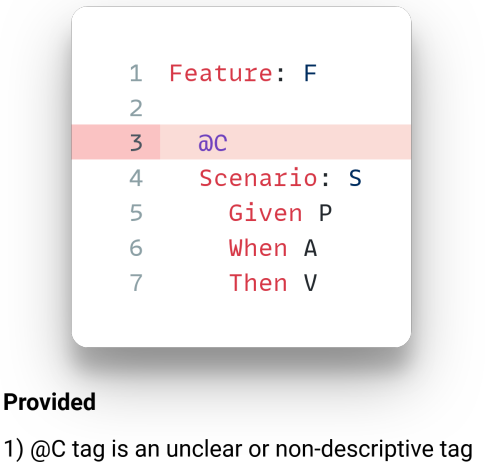
7.2.3 Tag Smells

Definidos quando se há uso incompreensível, irrelevante ou excessivo de *tags*. Dificulta o entendimento do propósito da *feature* ou *scenario* para os desenvolvedores, além da implementação incorreta de *hooks*. As etiquetas de *scenarios* representam 3 instâncias do total de *smells* encontrados neste estudo. Por se tratar de um componente simples, 2 dos casos envolvem situações de contexto que podem confundir os desenvolvedores durante a escrita dos testes ou implementação de ganchos de execução, e portanto, não puderam ser validados na ferramenta proposta como solução no capítulo 8.

7.2.3.1 Cryptic Tag

Uma *Cryptic Tag* ocorre quando uma *tag* usada em um cenário ou funcionalidade não é clara o suficiente para comunicar o seu propósito, confundindo principalmente novos contribuidores. Os detalhes deste *smell* são apresentados na Figura 11.

Figura 11 - Regra para *Cryptic Tag*



Fonte: Os autores.

As principais consequências deste *smell* envolvem a dificuldade de compreensão da finalidade da *tag*; manutenção complexa, pelo difícil esclarecimento de onde a *tag* pode ser adicionada ou removida nos testes; erros de organização, sendo prejudiciais à categorização e execução dos testes. O Código-Fonte 12 fornece um exemplo de *Cryptic Tag* encontrado na amostra.

Código-Fonte 12 - Demonstração de *Cryptic Tag*

```
@ee
  Scenario: Shared admin generates a token for an isolated environment
```

Fonte: Dados da pesquisa.

Percebe-se que o propósito de uso da etiqueta “@ee” é indecifrável, confundindo seriamente novos testadores. Essa *tag* é apenas uma das infinitas nomenclaturas que definem este *smell*.

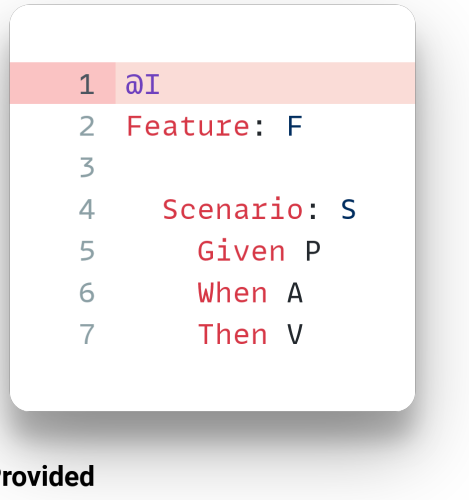
Como este *smell* tem um propósito contextual, não há necessidade de verificação pelo Cucumber, visto que sua ocorrência não interfere no seu fluxo de execução. Além disso, não seria possível validar sua presença de maneira eficiente, visto as limitações das expressões regulares implementadas na solução.

7.2.3.2 Dead Feature

Um outro *smell* contextual de caráter semelhante ao *Cryptic Tag* ocorre quando uma etiqueta é usada em um arquivo de funcionalidade que contém cenários desatualizados ou obsoletos no projeto. As etiquetas são recursos personalizáveis, que na maioria das vezes exige um contexto interno entre os desenvolvedores para evitar falsos resultados durante a execução de testes.

Este *smell* aponta questionamentos sobre a necessidade de se preservar casos de teste irrelevantes, que apenas resultam em filtros de execução e *hooks* desnecessários. A Figura 12 ilustra as regras de identificação deste *smell*.

Figura 12 - Regra para *Dead Feature*



Fonte: Os autores.

Os seus principais impactos são o acúmulo de código obsoleto, que torna o projeto mais pesado e difícil de se manter; confusão para os testadores, gerando dúvidas sobre a relevância dos cenários ao executar os testes; perda de foco, pela possibilidade de desvio de atenção e de funcionalidades ativas importantes. O Código-Fonte 13 apresenta um trecho de *feature* presente na amostra como exemplo.

Código-Fonte 13 - Demonstração de *Dead Feature*

```
@api/v1.0 @deprecated
Feature: Release artifact relationship
```

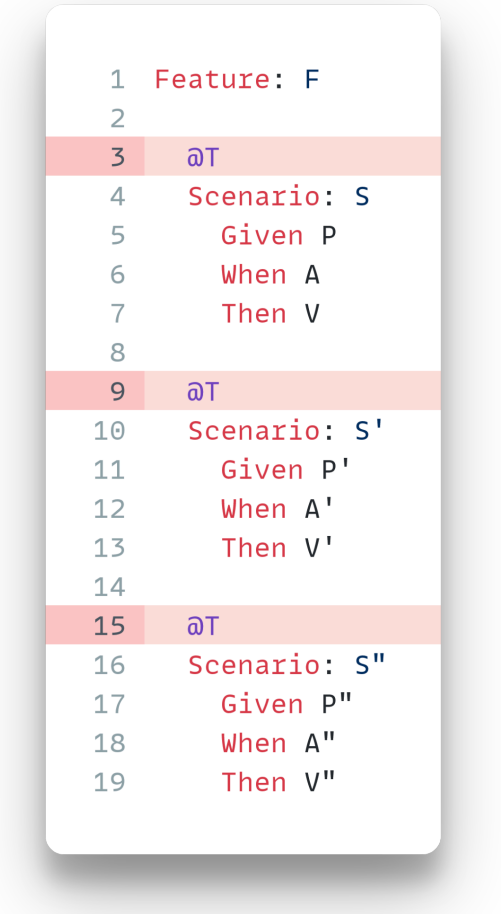
Fonte: Dados da pesquisa.

A funcionalidade possui testes de uma versão anterior de uma API, que não está mais em uso, portanto os desenvolvedores adicionaram a *tag* “@deprecated” como indicativo da sua obsolescência. Nesta situação também não há necessidade realizar validações de comportamento no Cucumber, por razões de contexto anteriormente citadas. Também não será considerada sua implementação na ferramenta de detecção automatizada de *smells*.

7.2.3.3 Vicious Tag

Uma *Vicious Tag* é a repetição da mesma etiqueta em todos os cenários da funcionalidade, gerando problemas de manutenção e duplicação de código. De acordo com Cucumber (2024), quando uma *tag* é adicionada ao título da funcionalidade, ela será automaticamente associada a todos os casos de teste presentes, o que permite uma implementação mais limpa e evita os problemas citados. A Figura 13 define as regras do *smell*.

Figura 13 - Regra para *Vicious Tag*



Fonte: Os autores.

Dado que:

1) @T é uma *tag*

Como principais efeitos, há a redundância de código, aumentando o tamanho e reduzindo a legibilidade do arquivo; manutenção trabalhosa, graças às mudanças na *tag* que exigem atualizações massivas nos cenários; e propensão a erros, por

haver um risco maior de inconsistências ao gerenciar um grande número de etiquetas manualmente.

Em estudos relativos ao comportamento do Cucumber, percebeu-se que quando *tags* são atribuídas ao *Background*, erros de execução são disparados. As *tags* também se comportam de maneira curiosa quando sobrepostas na palavra-chave *Rule*, que executa todos os cenários até a próxima regra da funcionalidade. O mesmo se aplica aos *Examples* de um *Scenario Outline*.

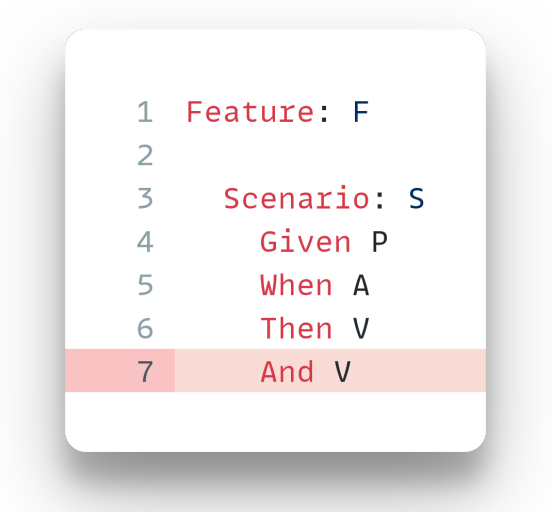
7.2.4 Step smells

Envolvem qualquer defeito na construção do fluxo de passos do cenário de teste. Esta categoria possui as mais variadas causas que apontam erros lógicos ou de sintaxe durante o desenvolvimento. Os *Step Smells* representam a maior porção das propostas dessa seção do estudo, com um total de 4 variações estruturais extremamente características que tornam o teste defeituoso.

7.2.4.1 Duplicate Step

Essa proposta diz respeito a duplicações duvidosas dos passos dos casos de teste (desconsiderando a *keywork* de acompanhamento). O que pode resultar em resultados incorretos, por erros de implementação dos desenvolvedores. As regras de identificação podem ser observadas a seguir (Figura 14).

Figura 14 - Regra para *Duplicate Step*



Fonte: Os autores.

Cabe lembrar que a situação não se limita apenas aos *scenarios*, mas também ao *Background*. As principais consequências deste *smell* são redundância de código e múltipla execução de um mesmo *step* desnecessariamente. O Código-Fonte 14 demonstra uma das ocorrências encontradas na amostra.

Código-Fonte 14 - Demonstração de *Duplicate Step*

```
Scenario: Admin quick validates a check-in license that is valid
  Given I am an admin of account "test1"
  And the current account is "test1"
  And the current account has 1 "webhook-endpoint"
  And the first "webhook-endpoint" has the following attributes:
    """
    {
      "subscriptions": ["*"]
    }
    """
  And the current account has 1 "policies"
  And all "policies" have the following attributes:
    """
    {
      "requireCheckIn": true,
      "checkInInterval": "day",
      "checkInIntervalCount": 1
    }
    """
  And the current account has 1 "license"
  And all "licenses" have the following attributes:
    """
    {
      "policyId": "$policies[0]",
      "lastCheckInAt": "$time.now"
    }
    """
  And I use an authentication token
  When I send a GET request to "/accounts/test1/licenses/$0/actions/validate"
  Then the response status should be "200"
  And the response should contain a valid signature header for "test1"
  And the response body should contain a "license"
  And the response body should be a "license" with a lastValidated within seconds of "$time.now.iso"
```

```
And the response body should contain meta which includes the following:
  """
  { "valid": true, "detail": "is valid", "code": "VALID" }
  """
And the response should contain a valid signature header for "test1"
And sidekiq should process 1 "touch-license" job
And the first "license" should have a lastValidatedAt within seconds of "$time.now.iso"
And sidekiq should have 0 "webhook" jobs
And sidekiq should have 0 "metric" jobs
And sidekiq should have 1 "request-log" job
```

Fonte: Dados da pesquisa.

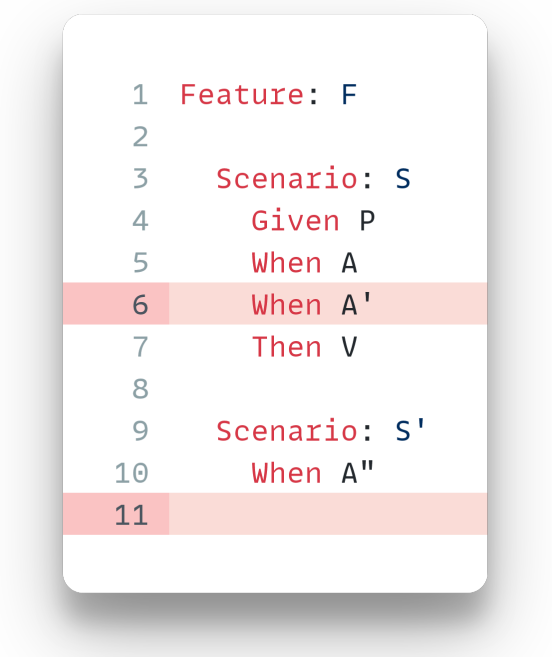
A repetição de passos também é um forte indicador de que mais de um teste está sendo realizado dentro do mesmo cenário, o que ocasiona em um caso de teste incorreto que anula um correto e vice-versa. Para o Código-Fonte 14, o *step* "*the response should contain a valid signature header for 'test1'"* gera desconfianças com relação a sua dupla aparição.

Para o Cucumber, um passo repetido é apenas mais um passo do teste que deverá ser executado. Além disso, os testes de verificação do estudo permitiram perceber que as *keywords Given, When* e *Then* não possuem nenhuma relevância durante a execução dos testes, pois cada descrição de *step* é uma função única que será executada pelo Cucumber, e justamente por isso, a duplicação de passos pode resultar em problemas, graças a reexecução de um passo que pode gerar resultados diferentes.

7.2.4.2 Malformed Test

O *Malformed Test* se trata do uso indevido das palavras reservadas *Given*, *When* ou *Then*, ao invés de se utilizar *And* ou *But* para dar continuidade ao raciocínio. A não inclusão de uma ação ou verificação essenciais para a estrutura de um cenário também são característicos deste *smell*. As suas regras de definição podem ser visualizadas na Figura 15.

Figura 15 - Regra para *Malformed Test*



Fonte: Os autores.

Os principais problemas ocasionados por esse *smell* envolvem quebra de legibilidade, pela dificuldade de entendimento da lógica do teste, devido à inconsistência na estrutura; e perda de clareza no objetivo, pela incompletude ou uso abusivo de *When* ou *Then*. O Código-Fonte 15 demonstra um caso de *Malformed Test* da amostra.

Código-Fonte 15 - Demonstração de *Malformed Test*

```
Scenario: Product retrieves the artifact for a release of their product (1 week TTL)
  Given the current account is "test1"
  And the current account has 1 "product"
  And the current account has 3 "releases" for the first "product"
  And the current account has 1 "artifact" for the first "release"
  Given I am a product of account "test1"
  And I use an authentication token
  And I use API version "1.0"
  When I send a GET request to "/accounts/test1/releases/$0/artifact?ttl=604800"
  Then the response status should be "303"
  And the response body should be an "artifact"
```

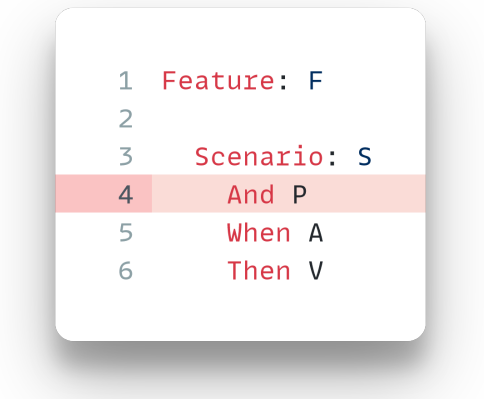
Fonte: Dados da pesquisa.

O Código-Fonte 15 mostra uma duplicação da *keyword Given*, situação essa que é indiferente para o Cucumber. Entretanto, em análises mais profundas, foi possível perceber que a execução de *scenarios outline* sem exemplos ou cenários com um ou nenhum passo, não são um problema para o Cucumber, especificamente o segundo caso, onde o resultado é compreendido como um sucesso, o que não deveria acontecer.

7.2.4.3 Starting with the Left Foot

Ocorre quando um cenário de teste começa com *And*, *But* ou *Then*. Essas situações confundem o testador e o forçam a conferir novamente o *Background*, caso exista. Além disso, começar com *Then* implica em um Background com *When* (o que não consta na documentação do Cucumber) ou um cenário sem ações. A Figura 16 apresenta a regra de definição.

Figura 16 - Regra para *Starting With The Left Foot*



Fonte: Os autores.

Os principais problemas gerados por esse *smell* são confusão no entendimento, pela dificuldade de identificar o contexto inicial do teste; inconsistência estrutural, pela quebra de boas práticas definidas pela documentação do Cucumber; e diagnóstico trabalhoso, pela exigência de revisões adicionais do *Background* para entender o cenário. O Código-Fonte 16 apresenta um exemplo deste *smell*.

Código-Fonte 16 - Demonstração de *Starting With The Left Foot*

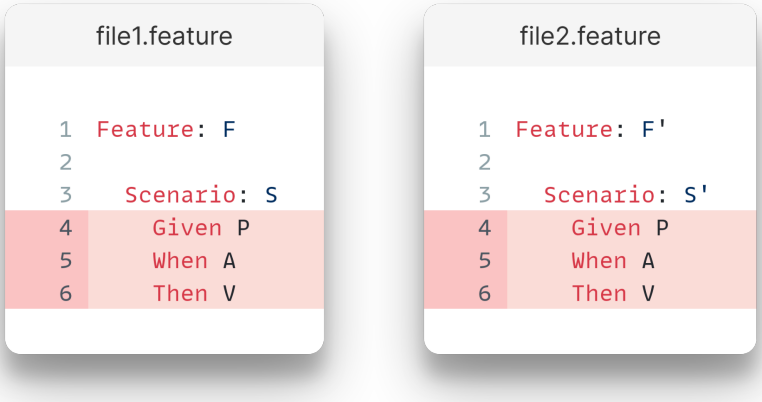
```
Scenario: A new candidate is available
        And the following candidates are currently available from remote API:
            | candidate |
            | activator |
            | groovy    |
            | kotlin    |
            | scala     |
        When I enter "sdk update"
        Then I see "Adding new candidates(s): kotlin"
        And the Candidates cache should contain "activator,groovy,kotlin,scala"
```

Fonte: Dados da pesquisa.

7.2.4.4 Duplicate Test Case

Ocorre quando o corpo do cenário se repete uma ou mais vezes dentro do mesmo arquivo *feature* ou entre arquivos distintos, independente do título e objetivo da funcionalidade. Se o conteúdo do teste é exatamente igual ao outro, não há necessidade da existência de outros cenários idênticos. A Figura 17 mostra a regra de identificação.

Figura 17 - Regra para *Duplicate Test Case*



Fonte: Os autores.

O *smell Duplicate Test Case* aponta redundâncias desnecessárias, aumentando o tamanho do arquivo sem agregar valor; ocasiona também uma manutenção ineficiente, pois alterações serão replicadas em cenários idênticos; e

gera resultados confusos, pois os relatórios podem conter duplicatas que dificultam a análise.

7.2.5 Table Smells

Apresentam uma estruturação enganosa das tabelas dentro do código de teste que podem ocasionar na obtenção de falsos positivos durante a execução do *scenario* ou até mesmo em um retorno não esperado dos dados inseridos como parâmetro. Apesar de representar apenas 1 dos *smells* descobertos, essa é uma categoria que possui alto grau de impacto em casos de teste durante a execução dos cenários e na sua compreensão, mas que por necessitar de contexto em seu caso de aparição, não foi possível adicioná-la à solução.

7.2.5.1 Incomplete Table

Uma *Incomplete Table* é uma tabela sem título, corpo ou quando há dispersão de chave e valor na horizontal (em linha). São características que norteiam a composição estrutural da tabela e podem causar confusão dos componentes pelo testador ou ferramenta. A Figura 18 representa a regra de apresentação do *smell*.

Figura 18 - Regra para *Incomplete Table*

```
1  Feature: F
2
3    Scenario: S
4      Given P
5
6        | Anna | 165 |
7      When A
8      Then V
9
10   Scenario: S'
11     Given P'
12     When A'
13       | name | height |
14
15     Then V'
16
17   Scenario: S"
18     Given P"
19     When A"
20     Then V"
21       | name   | Anna |
22       | height | 165  |
```

Fonte: Os autores.

O primeiro cenário representa uma tabela sem cabeçalho, enquanto o segundo se trata da ausência de valores que compõem o corpo da tabela. O terceiro, por sua vez, apresenta as informações em sentido de linha. Os problemas evidentes estão nos erros de automação, pelo processamento incorreto da tabela formatada; confusão na interpretação, graças a dificuldade do entendimento da estrutura dos dados; e a perda de clareza, que compromete a lógica do teste e a rastreabilidade das informações. O Código-Fonte 17 fornece uma demonstração desse *smell* na amostra.

Código-Fonte 17 - Demonstração de *Incomplete Table*

```
Scenario: [3] Use list lookup based on parameters when there is no type information
  Given any graph
  And parameters are:
    | expr | ['Apa'] |
    | idx  | 0       |
  When executing query:
    """
    WITH $expr AS expr, $idx AS idx
    RETURN expr[idx] AS value
    """
  Then the result should be, in any order:
    | value |
    | 'Apa' |
  And no side effects
```

Fonte: Dados da pesquisa.

Dentre todas as propostas, esta ofereceu maior contribuição analítica a respeito do poder de processamento do Cucumber. Quando uma tabela é executada somente com título, dois resultados distintos podem ser retornados pela ferramenta: um colchete vazio quando se é apresentado somente título, e um valor nulo para situações em que um campo vazio é adicionado. Esses casos podem mudar completamente os relatórios de teste, a depender da implementação.

Para situações em que a tabela não possui título, a ferramenta automaticamente interpreta a primeira linha como cabeçalho, enquanto a distribuição de chave e valor na horizontal, apontou que quanto maior a tabela, mais caótica se torna a apresentação dos dados, graças à completa confusão do Cucumber na identificação de chave-valor. Com relação à validação deste *smell* pela ferramenta, a sua extrema dificuldade de detecção por regex impossibilitou a sua detecção automatizada.

7.3 INTERPRETAÇÃO DA ANÁLISE PRELIMINAR

A análise manual dos 158 cenários de teste revelou 12 tipos distintos de *test smells*, classificados em cinco categorias principais. A Tabela 7 apresenta a

distribuição dos smells por categoria, destacando sua representatividade percentual no total de defeitos encontrados.

Tabela 7 - Distribuição das Categorias de *Test Smells* Identificados

Categoria	Quantidade
Title Smells	3 (25%)
Background Smells	1 (8,33%)
Tag Smells	3 (25%)
Step Smells	4 (33,33%)
Table Smells	1 (8,33%)
Total	**12 (100%)**

Fonte: Os autores.

A categoria mais frequente foi a de *Step Smells*, representando 33,33% dos defeitos identificados. Esse resultado sugere que problemas na formulação dos *steps* são os mais diversos em testes escritos no padrão BDD, impactando diretamente e de modo variado, a clareza, legibilidade e manutenção dos cenários de teste.

Os *Title Smells* e *Tag Smells*, com 25% cada, também apresentaram uma incidência significativa. A repetição de títulos de *features* e *scenarios*, bem como o uso inadequado ou confuso de *tags*, demonstram que esses aspectos estruturais são falhas comuns na escrita de testes BDD, podendo afetar a rastreabilidade e a organização dos testes dentro dos projetos de maneiras diferentes.

Já os *Background Smells* e *Table Smells*, com 8,33% cada, aparecem em menor variabilidade, indicando que problemas estruturais na organização do *Background* e na construção de tabelas ocorrem em menor diversificação, mas ainda podem comprometer a eficiência dos testes quando presentes.

Todos os *smells* encontrados foram registrados com um nome de batismo, descrição detalhada e informações relevantes dos resultados obtidos a partir de testes de comportamento. A finalidade desta etapa está fundamentada na necessidade de se ter definições robustas dos problemas identificados, antes da validação com especialistas e pesquisadores.

8 VALIDAÇÃO DOS *TEST SMELLS* IDENTIFICADOS

A validação dos *test smells* identificados foi conduzida por meio de um formulário estruturado para coletar percepções de profissionais da área de testes de *software*. O questionário foi elaborado com o propósito de avaliar a compreensão dos participantes sobre diferentes *test smells* presentes em cenários escritos no padrão BDD utilizando a linguagem Gherkin e o framework Cucumber. Além disso, buscou-se verificar a relevância desses *smells* na prática de testes automatizados e seu impacto na clareza, manutenção e rastreabilidade dos testes.

Os dados coletados permitiram uma análise quantitativa e qualitativa das percepções dos profissionais sobre esses problemas, fornecendo insumos para fundamentar a pesquisa. A seguir, serão detalhados os aspectos relacionados à construção do formulário, seus objetivos, público-alvo e resultados obtidos. O formulário na íntegra pode ser visto no **APÊNDICE C**.

8.1 CONSTRUÇÃO DO FORMULÁRIO

O formulário foi desenvolvido utilizando a ferramenta Google Forms, escolhida pela praticidade de criação, distribuição e coleta automática das respostas. O formulário foi estruturado de forma clara e objetiva, contendo questões que avaliam o entendimento dos profissionais sobre diferentes tipos de *smells* encontrados em casos de teste escritos utilizando o padrão BDD com a linguagem Gherkin. Cada pergunta descreve um *smell* específico acompanhado de exemplos, destacando claramente as implicações negativas desses problemas na qualidade e manutenção dos testes.

As questões foram organizadas em três seções principais: uma seção inicial com o Termo de Consentimento Livre e Esclarecido (TCLE), esclarecendo objetivos, riscos, benefícios e voluntariedade da participação; uma segunda seção sobre o perfil profissional dos respondentes (tempo de experiência e função), assim como informações demográficas; e uma terceira seção que apresenta descrições, exemplos e consequências identificadas para cada tipo de *smell* com opções para indicar o nível de concordância do participante sobre a relevância e impacto de cada *smell*.

Os participantes foram convidados de forma voluntária, recebendo o link de acesso ao formulário através de diversos canais de comunicação, tais como e-mail, Microsoft Teams e WhatsApp.

8.2 OBJETIVO

O objetivo principal do formulário foi validar a percepção dos profissionais de teste de *software* sobre a existência, relevância e impactos dos *smells* identificados em testes escritos utilizando o padrão BDD com Gherkin. A intenção era compreender como esses profissionais percebem os problemas relacionados a más práticas de design e implementação desses testes, além de obter insights sobre a importância atribuída por esses profissionais aos problemas apontados.

Essa avaliação contribui diretamente para o objetivo geral do estudo, que visa identificar e caracterizar os principais *smells* presentes em cenários escritos em BDD, ajudando a compreender como tais problemas afetam a qualidade, a clareza e a manutenção dos testes.

8.3 PÚBLICO-ALVO

O público-alvo do formulário foram profissionais da área de testes de *software* com experiência em BDD e especificamente com a linguagem Gherkin e o padrão BDD utilizando a ferramenta Cucumber. O público incluiu:

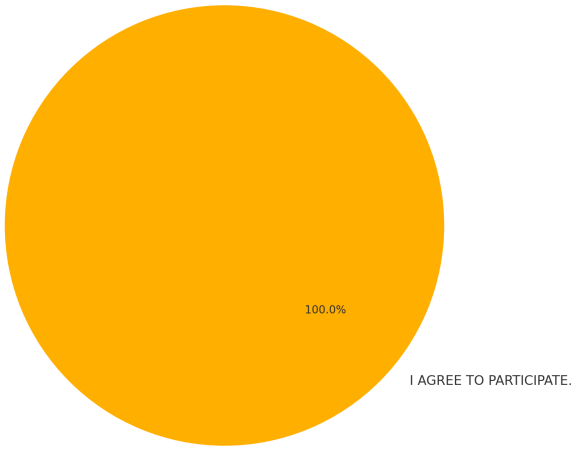
- Analistas de Teste de *Software*;
- Engenheiros de Teste de *Software*;
- Analistas de Garantia da Qualidade;
- Outros profissionais diretamente envolvidos com a criação, manutenção e execução de testes automatizados utilizando BDD.

Esses profissionais foram selecionados justamente pelo seu conhecimento técnico e experiência prática na área, possibilitando uma avaliação precisa e pertinente dos problemas identificados pelo estudo.

8.4 RESULTADOS

Um total de 22 pessoas responderam ao formulário, fornecendo informações sobre sua compreensão e percepção em relação aos *test smells* identificados em testes escritos no padrão BDD utilizando a linguagem Gherkin e o framework Cucumber. A totalidade dos respondentes manifestou concordância em participar da pesquisa após lerem o TCLE, o que indica que todos os participantes compreenderam plenamente os objetivos, riscos e benefícios relacionados à sua participação.

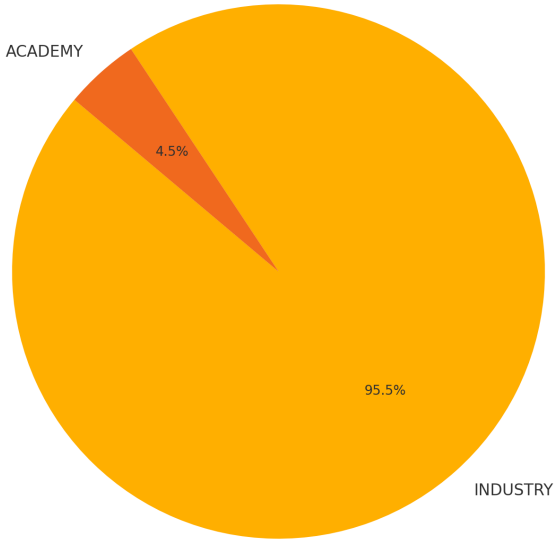
Gráfico 1 - Termo de consentimento livre e esclarecido



Fonte: Os autores.

O formulário também revelou que a maioria absoluta dos participantes (95,45%, correspondente a 21 indivíduos) atua na indústria, enquanto uma pequena parcela (4,55%, representando 1 indivíduo) atua na academia. Esse resultado destaca a predominância significativa de profissionais da indústria na pesquisa, proporcionando insights diretamente alinhados à prática profissional cotidiana de testes de *software*.

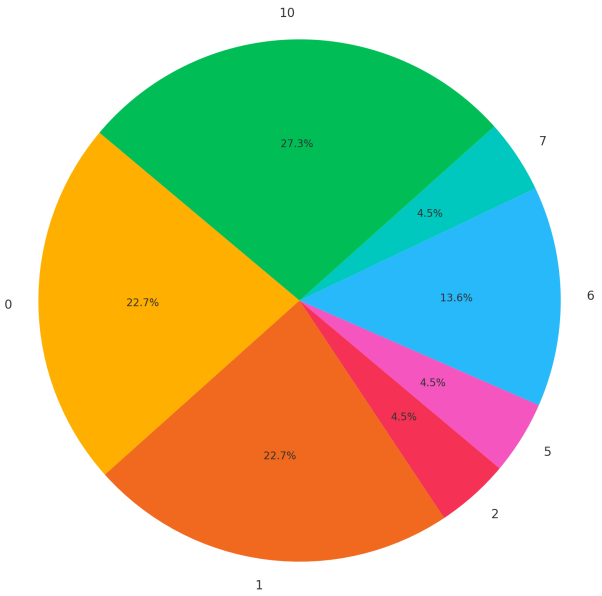
Gráfico 2 - Área de atuação



Fonte: Os autores.

Com relação à experiência com testes de *software*, a pesquisa revelou uma distribuição diversificada entre os participantes. Do total de 22 respondentes, a maioria declarou possuir 10 ou mais anos de experiência (27,27%, 6 indivíduos), indicando uma significativa experiência na área. Além disso, houve uma parcela considerável de respondentes com pouca experiência (menos de 1 ano e 1 ano), cada categoria correspondendo a 22,73% (5 indivíduos em cada). Observou-se ainda participantes com níveis intermediários de experiência: 13,64% (3 indivíduos) com 6 anos, e parcelas menores (4,55% cada, representando 1 indivíduo) nas categorias de 2, 5 e 7 anos. Esse cenário demonstra que o formulário alcançou um público heterogêneo quanto à experiência em testes de *software*, garantindo uma ampla variedade de perspectivas sobre os *test smells* abordados na pesquisa.

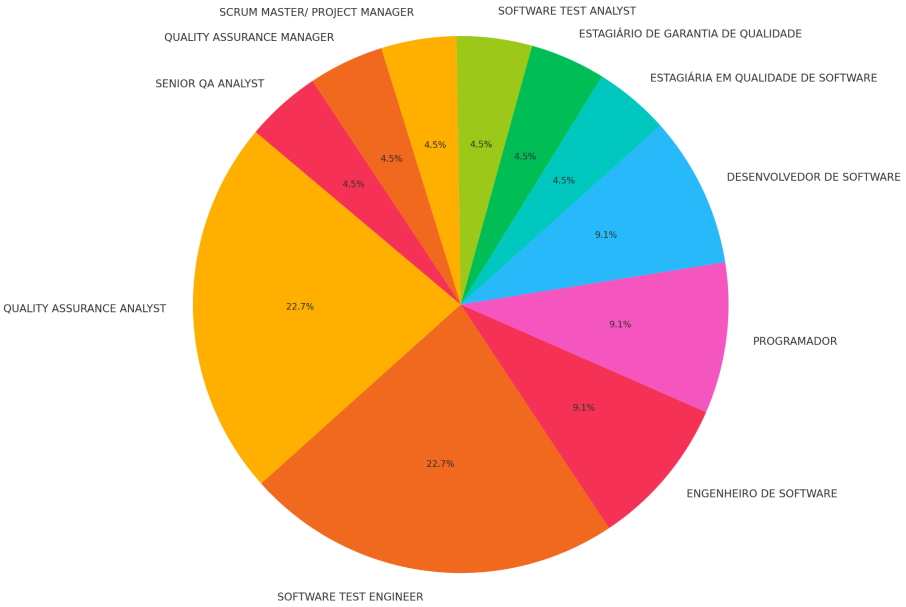
Gráfico 3 - Experiência com testes de software



Fonte: Os autores.

Os resultados obtidos na pergunta referente à ocupação dos participantes evidenciou uma diversidade significativa entre as ocupações dos 22 participantes da pesquisa. Destacaram-se duas funções principais: *Quality Assurance Analyst* e *Software Test Engineer*, ambas com 5 respondentes cada, representando individualmente 22,73% do total. Em seguida, houve a presença de profissionais identificados como Engenheiro de *Software*, Programador e Desenvolvedor de *Software*, cada uma dessas categorias com 2 respondentes, representando 9,09% individualmente. Por fim, outras ocupações como Estagiária em Qualidade de *Software*, Estagiário de Garantia de Qualidade, *Software Test Analyst*, *Scrum Master*/*Project Manager*, *Quality Assurance Manager* e *Senior QA Analyst* foram citadas por 1 respondente cada (4,55% por categoria). Esses resultados confirmam que a pesquisa conseguiu captar percepções provenientes de diversos perfis profissionais ligados diretamente às práticas de testes de *software*, contribuindo para a riqueza e abrangência dos dados obtidos.

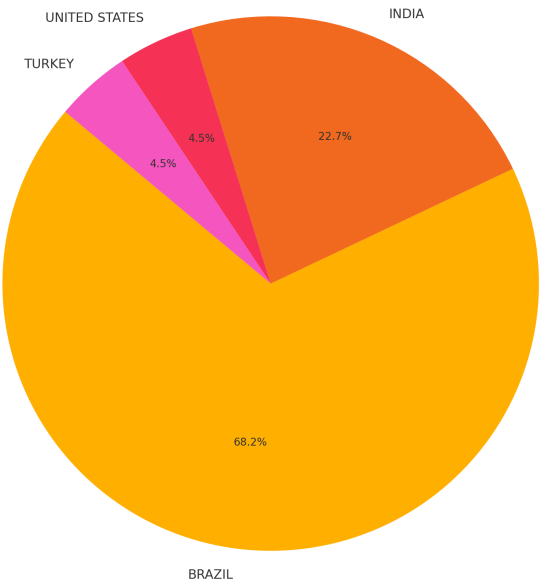
Gráfico 4 - Ocupações



Fonte: Os autores.

Com relação aos dados demográficos, houve uma predominância significativa de participantes residentes no Brasil, com 15 respondentes, representando 68,18% do total. Em seguida, a Índia apareceu como o segundo país mais frequente, com 5 participantes (22,73%). Outros países, como Estados Unidos e Turquia, tiveram participação menor, cada um com apenas 1 respondente (4,55% cada). Esses resultados indicam que o estudo alcançou majoritariamente profissionais brasileiros, com relevante participação internacional, o que pode contribuir para uma compreensão mais ampla e diversificada sobre a percepção dos *test smells* identificados na pesquisa.

Gráfico 5 - Demografia dos participantes



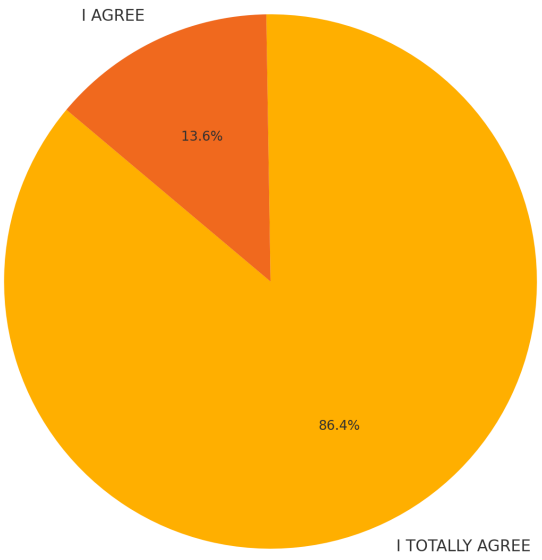
Fonte: Os autores.

A análise da questão referente ao smell denominado "Untitled Feature", caracterizado pela ausência de título em arquivos *feature*, mostrou que a grande maioria dos respondentes concorda fortemente com sua relevância. Dos 22 participantes, 86,36% (19 respondentes) afirmaram "concordar totalmente" com as consequências negativas desse *smell*, enquanto 13,64% (3 respondentes) indicaram apenas "concordar". Nenhum respondente discordou ou demonstrou neutralidade sobre o impacto desse problema, indicando forte consenso acerca da importância de atribuir títulos claros aos arquivos para evitar ambiguidades, manter a rastreabilidade e melhorar a comunicação entre equipes.

Os comentários fornecidos pelos respondentes sobre o *smell* "Untitled Feature" revelou importantes percepções qualitativas. Os participantes destacaram principalmente que a ausência de um título claro em arquivos *feature* prejudica significativamente a clareza, comunicação e rastreabilidade dos testes realizados. Eles enfatizaram que essa ausência pode causar confusão e dificultar a manutenção e o reaproveitamento de testes existentes em novos cenários, especialmente em contextos onde existem muitas funcionalidades. Além disso, foi ressaltado que títulos descritivos são essenciais para facilitar consultas futuras e garantir uma

melhor colaboração e entendimento entre as equipes envolvidas, contribuindo diretamente para a qualidade do *software* e eficiência do fluxo de trabalho.

Gráfico 6 - Validação do smell *Untitled Feature*



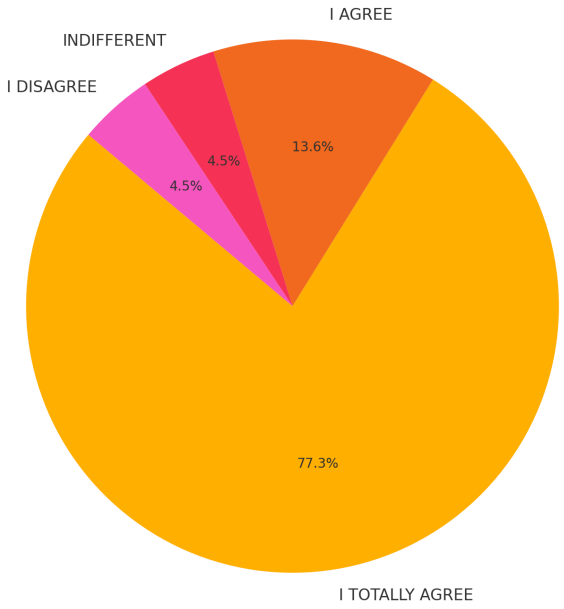
Fonte: Os autores.

A análise das respostas obtidas para a questão sobre o smell denominado "Duplicate Feature Title", caracterizado pela duplicação do título de uma funcionalidade em diferentes arquivos .feature, apresentou resultados predominantemente positivos em relação ao reconhecimento de sua importância. Dos 22 respondentes, 77,27% (17 participantes) afirmaram "concordar totalmente" com a relevância e consequências negativas desse smell, enquanto 13,64% (3 participantes) declararam apenas "concordar". Uma pequena parcela demonstrou opinião diferente, sendo 4,55% (1 participante) indiferente e outros 4,55% (1 participante) manifestando discordância. Esses resultados indicam que a maior parte dos profissionais reconhece claramente o potencial desse problema em gerar confusão de escopo, prejudicar a rastreabilidade e dificultar a interpretação correta dos resultados dos testes, embora exista uma pequena divergência de opiniões sobre o grau de relevância desse *smell* específico.

A análise dos comentários fornecidos pelos participantes referente à questão sobre o *smell* "Duplicate Feature Title" revelou pontos interessantes sobre a

percepção dos respondentes. A maioria destacou que títulos duplicados podem causar dificuldades na identificação precisa das funcionalidades, aumentando o tempo gasto nas tarefas e prejudicando a rastreabilidade dos testes. Os respondentes ressaltaram que essa duplicidade gera complexidade desnecessária, aumentando os riscos de erros de interpretação durante a análise dos testes e prejudicando a qualidade do código, além de afetar negativamente a comunicação e eficiência das equipes envolvidas. Entretanto, foi destacado também um ponto divergente: alguns participantes argumentaram que em certas situações práticas é necessário criar arquivos .feature com títulos semelhantes para agrupar testes específicos (como testes de regressão ou testes ponta-a-ponta), ou para facilitar a divisão de arquivos extensos, embora concordem que isso pode gerar confusão se não gerenciado adequadamente. Em síntese, a maioria concorda que títulos claros e únicos são preferíveis para evitar problemas futuros.

Gráfico 7 - Validação do smell *Duplicate Feature Title*



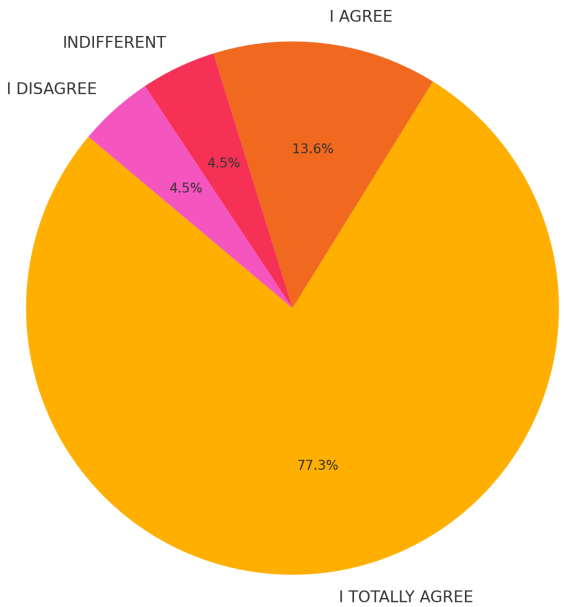
Fonte: Os autores.

A análise das respostas relacionadas à questão sobre o *smell* denominado "Duplicate Scenario Title", caracterizado pela duplicação de títulos de cenários em arquivos *feature*, revelou um elevado nível de concordância entre os participantes. Dos 22 respondentes, 77,27% (17 indivíduos) afirmaram "concordar totalmente" com as consequências negativas descritas, enquanto 13,64% (3 participantes) indicaram

apenas "concordar". Uma minoria (9,09%, ou 2 participantes) discordou da relevância deste *smell*. De forma geral, percebe-se que a maioria dos profissionais reconhece que esse problema gera confusão nos resultados dos testes, inconsistências na compreensão do comportamento esperado e prejuízos significativos na rastreabilidade dos cenários em relação aos requisitos.

A análise qualitativa dos comentários fornecidos pelos respondentes referente à questão sobre o *smell* "Duplicate Scenario Title", revelou percepções relevantes e algumas divergências. A maioria dos respondentes reforçou que a duplicação dos títulos dos cenários prejudica a clareza na identificação dos testes executados, aumentando a complexidade, comprometendo a qualidade dos testes e impactando negativamente a eficiência do processo de desenvolvimento e verificação do *software*. Entretanto, também foi mencionado que, em certos contextos específicos (como testes de sanidade, regressão ou ponta-a-ponta), pode ser aceitável reutilizar títulos similares, desde que os passos sejam distintos e claramente definidos. Esse ponto indica que, embora a maioria reconheça as consequências negativas desse *smell*, alguns profissionais identificam situações práticas específicas em que a duplicação de títulos pode ser justificável ou até necessária.

Gráfico 8 - Validação do smell *Duplicate Scenario Title*

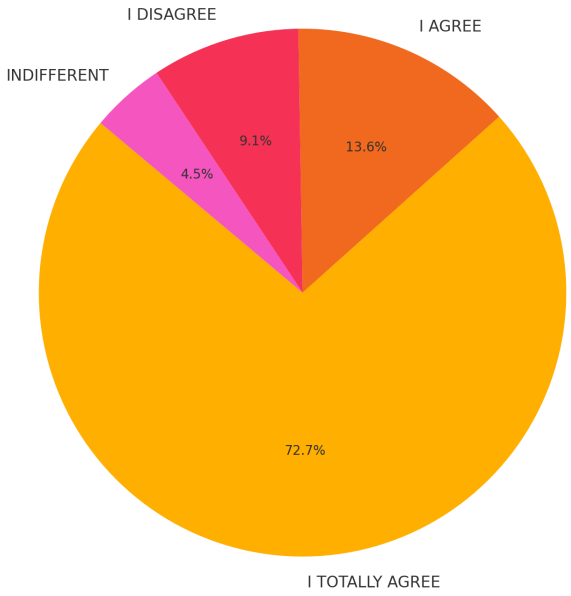


Fonte: Os autores.

A análise da questão sobre o *smell* denominado "Absence of Background", caracterizado pela repetição de precondições em múltiplos cenários sem utilizar o recurso *Background*, revelou resultados predominantemente concordantes entre os participantes. Dos 22 respondentes, 72,73% (16 indivíduos) afirmaram "concordar totalmente" com as consequências descritas para esse *smell*, enquanto 13,64% (3 indivíduos) indicaram apenas "concordar". Em contrapartida, 9,09% (2 respondentes) discordaram das consequências apontadas, e 4,55% (1 indivíduo) manifestou indiferença. Esses resultados demonstram que a grande maioria dos profissionais reconhece que a ausência do recurso *Background* gera redundância de código, reduz a clareza dos testes e aumenta a dificuldade de manutenção ao exigir múltiplas atualizações em cenários distintos. Contudo, observa-se uma pequena divergência quanto à relevância dessas consequências na prática diária.

A análise dos comentários dos participantes sobre a questão relacionada ao smell "Absence of Background" revelou percepções coerentes e convergentes. Os respondentes enfatizaram que a ausência do recurso *Background* para agrupar precondições repetidas frequentemente leva à redundância de código, tornando-o mais extenso e menos claro. Foi destacado ainda que a não utilização adequada do *Background* aumenta significativamente a dificuldade de manutenção dos testes, especialmente quando alterações nas precondições exigem atualizações em múltiplos cenários. Um participante mencionou que, apesar de parecer trivial inicialmente, a repetição contínua dessas precondições pode gerar problemas expressivos no médio e longo prazo. Outro comentário ressaltou que a prática correta do uso do *Background* melhora a organização, otimiza o processo de desenvolvimento e facilita a manutenção e clareza dos testes. Esses comentários reforçam a importância desse recurso na melhoria da qualidade e eficiência das práticas de testes baseadas em BDD.

Gráfico 9 - Validação do smell *Absence of Background*



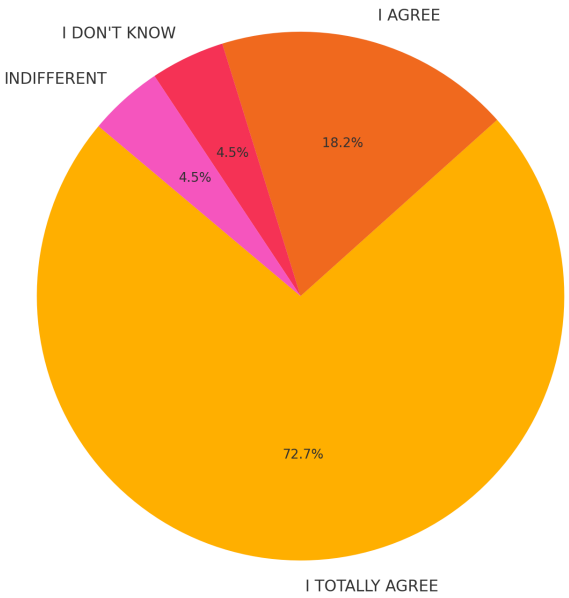
Fonte: Os autores.

A análise das respostas obtidas para a questão sobre o *smell* denominado "Cryptic Tag", caracterizado pelo uso de *tags* pouco claras ou que não comunicam adequadamente seu propósito, indicou que a grande maioria dos participantes concorda com a relevância desse problema. Dos 22 respondentes, 72,73% (16 indivíduos) afirmaram "concordar totalmente" com as consequências descritas, enquanto 18,18% (4 participantes) indicaram "concordar". Além disso, houve 4,55% (1 respondente) que declarou "não sei" e outros 4,55% (1 respondente) manifestaram-se "indiferentes". Esses resultados destacam que a maior parte dos profissionais reconhece claramente que o uso de *tags* criptografadas ou ambíguas causa dificuldades significativas na compreensão do objetivo dos testes, gera complexidade desnecessária na manutenção e compromete a organização e categorização dos cenários. A presença de respostas neutras e indecisas sugere também uma leve incerteza em relação a este *smell* específico entre uma pequena parcela dos respondentes.

A análise dos comentários fornecidos pelos participantes sobre o *smell* "Cryptic Tag" evidenciou uma concordância geral quanto à importância de clareza nas *tags* utilizadas em cenários e funcionalidades. Os respondentes destacaram que *tags* ambíguas ou pouco claras prejudicam diretamente a eficiência da manutenção,

organização e execução dos testes. Foi amplamente mencionado que cenários ou funcionalidades com *tags* inadequadas deveriam ser reescritos, garantindo clareza e facilidade de compreensão por todos os membros da equipe. Contudo, um comentário apontou uma perspectiva diferente, sugerindo que *tags* aparentemente "criptografadas" não seriam necessariamente um problema caso futuramente cenários adicionais relacionados à mesma funcionalidade sejam adicionados, justificando a existência momentânea da *tag* menos clara. Ainda assim, prevalece o consenso entre os participantes sobre a importância de se utilizar *tags* intuitivas e descritivas, visando a eficiência e eficácia das atividades de teste.

Gráfico 10 - Validação do smell *Cryptic Tag*



Fonte: Os autores.

A análise da questão referente ao *smell* denominado "Dead Feature", caracterizado pela presença de cenários desatualizados ou obsoletos em arquivos *feature*, mostrou um alto nível de concordância sobre suas implicações negativas. Dos 22 respondentes, 72,73% (16 participantes) indicaram "concordar totalmente" com as consequências descritas. Outros 13,64% (3 respondentes) manifestaram-se indecisos, indicando não ter certeza sobre o impacto desse *smell*. Além disso, 9,09% (2 participantes) concordaram parcialmente com a questão, enquanto 4,55% (1 participante) discordaram. Os resultados demonstram um consenso geral de que manter cenários obsoletos no projeto gera confusão entre testadores, acarreta

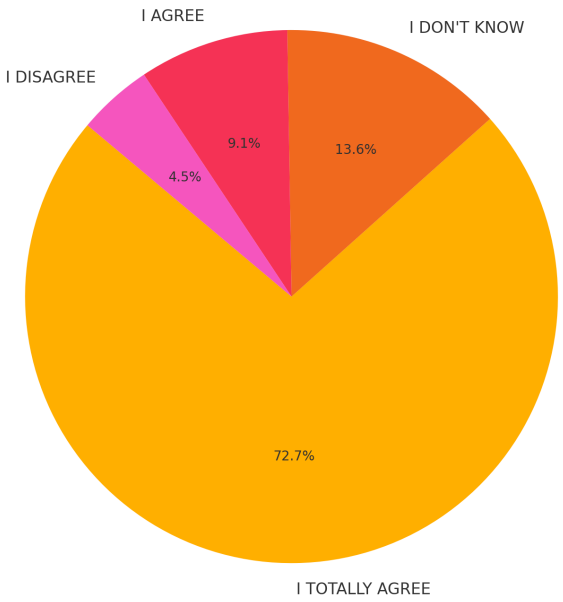
acúmulo desnecessário de código e desvia a atenção de funcionalidades ativas e relevantes, prejudicando assim a manutenção e qualidade do projeto como um todo.

A análise qualitativa dos comentários relacionados à questão sobre o *smell* "Dead Feature" revelou duas perspectivas distintas entre os respondentes. A maioria enfatizou que cenários obsoletos ou desatualizados aumentam a complexidade e distraem os testadores, prejudicando a manutenção e organização do projeto. Além disso, ressaltaram que mesmo marcados com *tags* como *@deprecated*, tais cenários continuam representando riscos à eficiência e qualidade, recomendando que sejam removidos para evitar confusões e distrações durante a execução dos testes.

Por outro lado, uma minoria destacou que, em algumas circunstâncias específicas, pode ser útil manter temporariamente cenários marcados claramente com tags padronizadas (ex.: *@deprecated* ou *@obsolete*), visando preservar o histórico ou facilitar futuras referências sobre funcionalidades antigas.

Ainda assim, prevalece entre a maioria dos participantes o entendimento de que cenários obsoletos devem ser preferencialmente eliminados para garantir a clareza e facilidade de manutenção do projeto de testes.

Gráfico 11 - Validação do smell *Dead Feature*

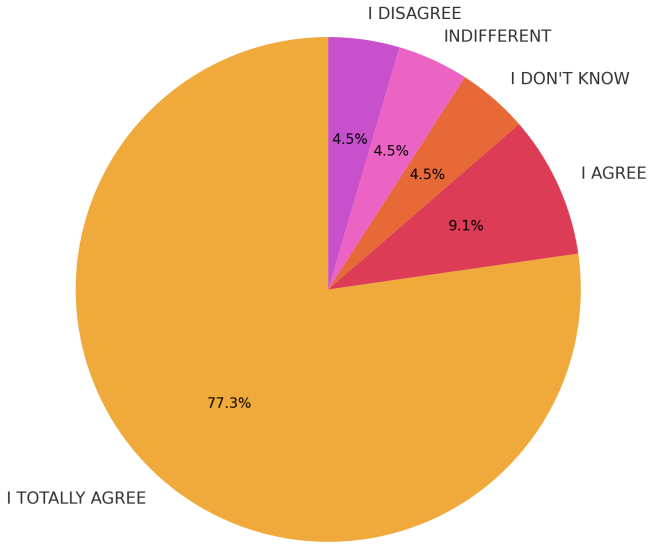


Fonte: Os autores.

A análise das respostas referentes ao *smell* "Vicious Tag", caracterizado pela repetição excessiva de uma *tag* em todos os cenários de um arquivo *feature*, demonstrou forte concordância sobre suas consequências negativas. Dos 22 respondentes, 77,27% (17 indivíduos) afirmaram "concordar totalmente", destacando preocupações sobre redundância no código, aumento desnecessário no tamanho dos arquivos e dificuldades na manutenção dos testes. Além disso, 9,09% (2 participantes) indicaram concordância moderada. Observou-se também respostas minoritárias distribuídas entre "não sei", "indiferente" e "discordo", com 4,55% (1 indivíduo) para cada uma dessas categorias. Esse resultado evidencia que, apesar do consenso majoritário quanto à relevância negativa deste *smell*, há uma pequena parcela com percepções mais neutras ou incertas sobre suas implicações práticas.

A análise qualitativa dos comentários relacionados à questão sobre o *smell* "Vicious Tag" revelou percepções amplamente concordantes, destacando preocupações quanto às consequências negativas da repetição excessiva de *tags* em arquivos *feature*. Os respondentes mencionaram que o uso indiscriminado e repetido de *tags* pode tornar os testes redundantes, dificultar a manutenção e aumentar a propensão a erros. Alguns participantes sugeriram explicitamente a importância de utilizar as *tags* de maneira mais seletiva, clara e organizada, evitando redundâncias desnecessárias para facilitar a manutenção e reduzir a complexidade. Também foi sugerido que, em vez de aplicar uma mesma *tag* repetidamente em todos os cenários, uma melhor prática seria colocar essa *tag* no nível da funcionalidade inteira, simplificando a organização e aumentando a legibilidade dos testes. Entretanto, alguns comentários sugeriram que em certos casos específicos, a repetição de *tags* pode ser necessária dependendo do contexto de teste, embora esses casos sejam considerados exceções.

Gráfico 12 - Validação do smell *Vicious Tag*



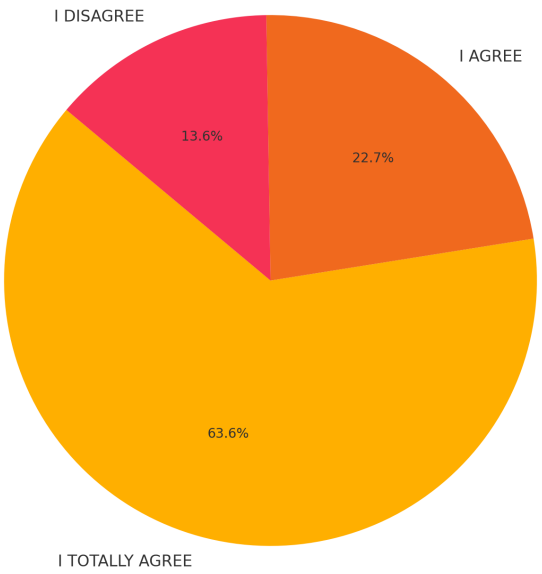
Fonte: Os autores.

A análise das respostas para o *test smell Duplicate Step*, caracterizado pela repetição de um mesmo passo dentro de um único cenário, revelou ampla concordância entre os participantes quanto às suas consequências negativas. Dos 22 respondentes, 63,64% (14 indivíduos) afirmaram “concordar totalmente” com os problemas associados, como redundância de código e execução desnecessária de etapas. Outros 22,73% (5 participantes) indicaram “concordar”, totalizando 86,37% de concordância. Em contrapartida, 13,64% (3 respondentes) discordaram da caracterização desse padrão como um *smell* prejudicial. Esses dados apontam para um consenso majoritário de que a duplicação de passos compromete a clareza, a eficiência e a manutenibilidade dos testes, embora uma pequena parcela de profissionais veja valor ou necessidade na repetição de passos em cenários específicos.

Os comentários dos respondentes sobre o *smell Duplicate Step* reforçaram a visão de que a repetição de passos dentro de um mesmo cenário deve ser evitada sempre que possível. Os participantes destacaram que a duplicação compromete a clareza e a eficiência dos testes, tornando-os mais difíceis de manter e propensos a erros ou inconsistências. Recomendaram a refatoração dos cenários para remover

redundâncias e o uso de etapas compartilhadas como alternativa mais sustentável. No entanto, surgiram contrapontos relevantes: alguns profissionais argumentaram que a duplicação pode ser necessária em casos específicos, como validações repetidas após diferentes ações, ou para confirmar que um estado do sistema permanece inalterado. Esses comentários evidenciam que, apesar do consenso predominante quanto aos impactos negativos do *Duplicate Step*, a aplicabilidade do *smell* pode depender do contexto funcional e dos objetivos de verificação de cada teste.

Gráfico 13 - Validação do smell *Duplicate Step*



Fonte: Os autores.

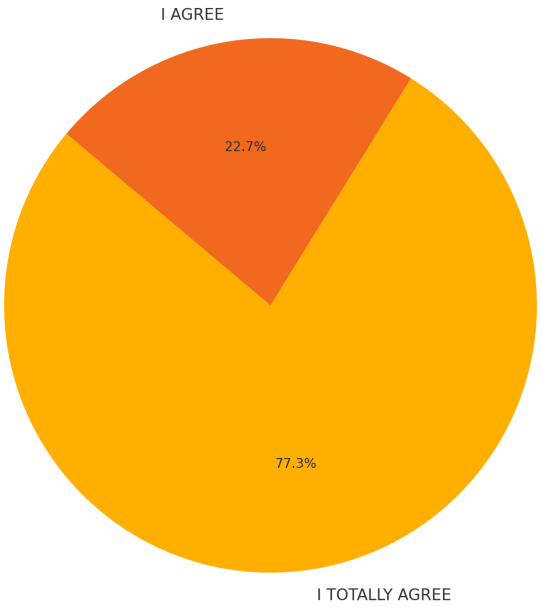
A análise das respostas obtidas para a questão sobre o *smell* "Malformed Test", caracterizado pelo uso inadequado das palavras reservadas *Given*, *When* e *Then* dentro de um cenário, demonstrou um alto nível de concordância entre os participantes. Dos 22 respondentes, 77,27% (17 indivíduos) afirmaram "concordar totalmente" com as consequências descritas, enquanto os 22,73% restantes (5 indivíduos) indicaram "concordar". Nenhum participante expressou discordância ou indiferença em relação ao problema.

Esse resultado evidencia um consenso significativo de que a má estruturação dos cenários de teste impacta negativamente a legibilidade, dificulta o entendimento da lógica do teste e pode gerar inconsistências na interpretação dos objetivos

testados. A ausência das palavras-chave essenciais ou seu uso inadequado pode comprometer a clareza dos testes, tornando-os ambíguos e difíceis de manter. Assim, fica evidente a necessidade de seguir boas práticas na escrita de cenários em BDD para garantir uma estrutura coesa e compreensível.

A análise dos comentários fornecidos pelos participantes referente à questão sobre o *smell* "Malformed Test", reforça a percepção de que cenários de teste mal estruturados comprometem significativamente a clareza e a legibilidade dos testes. Os respondentes enfatizaram que seguir corretamente as regras do Gherkin é essencial para garantir a correta interpretação e automação dos cenários. Além disso, foi destacado que testes malformados podem dificultar tanto a compreensão humana quanto a execução automatizada, tornando o processo de validação do comportamento do sistema menos eficiente. A manutenção de uma estrutura coesa e o uso adequado das palavras reservadas *Given*, *When* e *Then* são, portanto, fundamentais para assegurar que os cenários de teste sejam compreensíveis, completos e eficazes.

Gráfico 14 - Validação do smell *Malformed Test*



Fonte: Os autores.

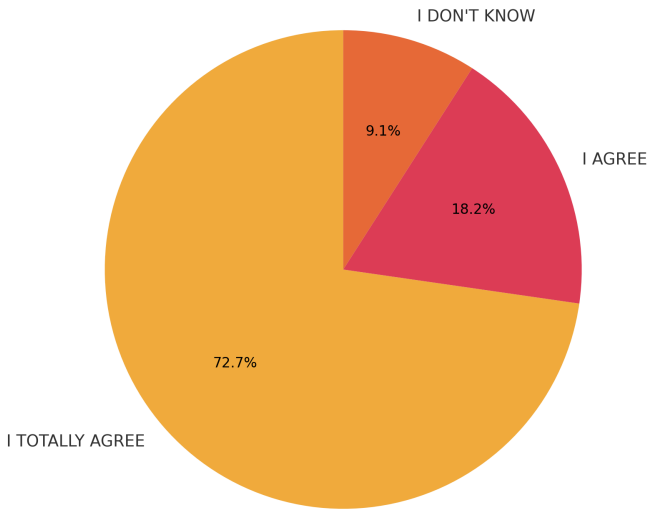
A análise das respostas obtidas para a questão sobre o *smell* "Starting With The Left Foot", caracterizado pelo uso inadequado de palavras reservadas no início

dos cenários de teste, revelou um alto nível de concordância entre os participantes. Dos 22 respondentes, 72,73% (16 indivíduos) afirmaram "concordar totalmente" com as consequências descritas, enquanto 18,18% (4 indivíduos) indicaram apenas "concordar". Além disso, 9,09% (2 indivíduos) declararam que não tinham certeza sobre o impacto desse *smell*, não havendo respostas de discordância.

Esses resultados demonstram que a maioria dos profissionais reconhece que iniciar um cenário com *And*, *But* ou *Then*, em vez das palavras-chave apropriadas *Given* ou *When*, pode dificultar a compreensão do contexto do teste, gerar inconsistências estruturais e aumentar o esforço necessário para diagnosticar e corrigir falhas. A presença de uma pequena parcela de participantes que não soube opinar pode indicar que esse *smell* é menos evidente em algumas práticas de escrita de testes, ou que sua relevância pode variar conforme o contexto de uso do BDD.

A análise dos comentários fornecidos pelos participantes sobre o *smell* "Starting With The Left Foot", reforça a importância de seguir as boas práticas estabelecidas pelo Cucumber. Os respondentes destacaram que iniciar um cenário com *And*, *But* ou *Then* em vez de *Given* ou *When* compromete a clareza e a estrutura dos testes, tornando-os mais difíceis de compreender e manter. Foi enfatizado que seguir as diretrizes recomendadas melhora a legibilidade, consistência e eficácia dos testes, além de facilitar a manutenção e depuração do código. Em síntese, a maior parte dos participantes concorda que a correta estruturação dos cenários é essencial para garantir um processo de teste bem organizado e alinhado às boas práticas do BDD.

Gráfico 15 - Validação do smell *Starting With The Left Foot*



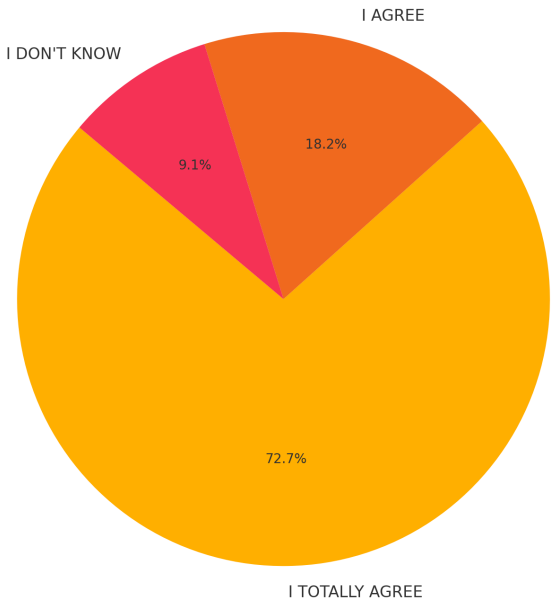
Fonte: Os autores.

A análise da questão referente ao *smell Duplicate Test Case*, caracterizado pela duplicação do conteúdo de um cenário de teste em um mesmo arquivo ou em arquivos distintos, demonstrou uma forte concordância quanto aos impactos negativos dessa prática. Dos 22 respondentes, 72,73% (16 participantes) afirmaram "concordar totalmente" com as consequências apresentadas, enquanto 18,18% (4 participantes) indicaram apenas "concordar". Em contrapartida, 9,09% (2 participantes) discordaram da caracterização do *smell*, representando uma minoria que relativiza sua criticidade dependendo do contexto de uso.

Os comentários qualitativos reforçaram a percepção majoritária de que cenários duplicados geram redundância desnecessária, dificultam a manutenção do conjunto de testes e tornam os relatórios confusos, especialmente quando contêm múltiplas execuções idênticas. Participantes defenderam que a eliminação de casos duplicados contribui para maior clareza, eficiência e melhor organização dos testes. Por outro lado, as respostas divergentes indicaram que, em alguns contextos específicos — como diferentes arquivos representando finalidades distintas — a repetição de testes pode ser justificada, desde que ocorra de forma intencional e controlada.

Esse panorama revela que, embora o *smell* seja amplamente reconhecido como problemático, sua severidade pode ser atenuada por decisões de design e organização do projeto de testes, destacando a importância do discernimento técnico ao aplicar práticas de reutilização ou separação de cenários.

Gráfico 16 - Validação do smell *Duplicate Test Case*



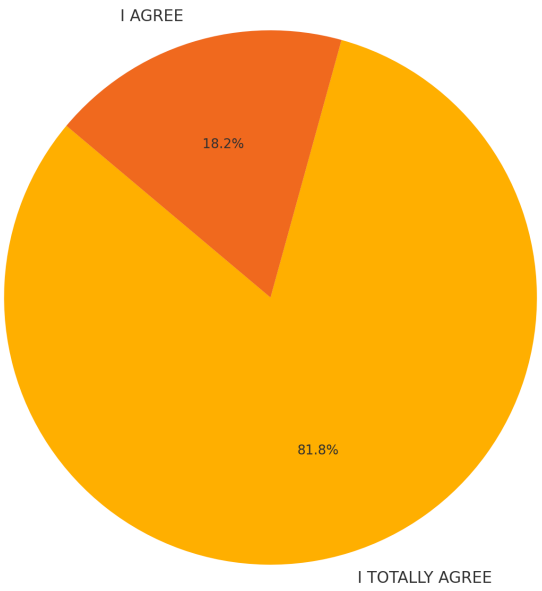
Fonte: Os autores.

A análise das respostas obtidas para a questão sobre o *smell* "Incomplete Table", caracterizado pela presença de tabelas incompletas, sem cabeçalho, sem corpo ou com valores mal formatados, revelou um consenso significativo entre os participantes. Dos 22 respondentes, 81,82% (18 indivíduos) afirmaram "concordar totalmente" com as consequências descritas, enquanto os 18,18% restantes (4 indivíduos) indicaram "concordar". Não houve respostas de discordância ou indiferença em relação ao problema.

Esses resultados demonstram que a maioria dos profissionais reconhece que tabelas mal estruturadas podem comprometer a automação dos testes, dificultar a interpretação dos dados e reduzir a clareza dos cenários, impactando diretamente a rastreabilidade das informações. A unanimidade entre os respondentes sugere que esse *smell* é amplamente reconhecido como um problema crítico que deve ser evitado para garantir a consistência e eficácia dos testes no padrão BDD.

A análise dos comentários fornecidos pelos participantes com relação ao *smell* "Incomplete Table", reforça a percepção de que tabelas mal formatadas impactam negativamente tanto a automação dos testes quanto a interpretação dos dados. Os respondentes enfatizaram que tabelas incompletas ou mal estruturadas podem gerar erros na automação e confundir os testadores, dificultando a compreensão e execução dos testes. Além disso, foi destacado que garantir a completude, a formatação correta e a presença de cabeçalhos claros é essencial para manter a clareza e rastreabilidade dos testes, bem como para facilitar a manutenção do código. Um dos participantes também ressaltou a importância do uso de convenções consistentes de formatação e nomenclatura ao empregar tabelas em múltiplos cenários. Esse consenso demonstra que a organização e padronização das tabelas são fatores essenciais para a eficiência e confiabilidade dos testes no padrão BDD.

Gráfico 17 - Validação do smell *Incomplete Table*



Fonte: Os autores.

A análise dos dados coletados no formulário sobre a percepção dos profissionais em relação aos *test smells* identificados em testes escritos no padrão BDD utilizando a linguagem Gherkin e o framework Cucumber permitiu extrair insights relevantes sobre a compreensão e impacto desses problemas na prática de testes de *software*. Com 22 respondentes, a pesquisa alcançou um público

diversificado, majoritariamente composto por profissionais da indústria, com diferentes níveis de experiência e ocupações diretamente ligadas ao processo de teste e qualidade de *software*.

A concordância expressiva sobre a existência e as consequências dos *test smells* avaliados evidencia que esses problemas são reconhecidos e considerados prejudiciais à clareza, rastreabilidade e manutenção dos testes. *Smells* como *Untitled Feature*, *Duplicate Feature Title*, *Duplicate Scenario Title*, *Absence of Background*, *Cryptic Tag* e *Malformed Test* receberam altos índices de concordância, refletindo a percepção unânime de que essas más práticas impactam negativamente a estrutura e interpretação dos cenários de teste.

Além disso, os comentários qualitativos fornecidos pelos participantes reforçaram que a ocorrência desses *smells* pode gerar dificuldades tanto na automação quanto na compreensão dos testes, aumentando o esforço de manutenção e reduzindo a eficiência da execução dos cenários. Foi destacada a importância de seguir as boas práticas recomendadas pelo Cucumber e adotar padrões claros na escrita dos testes para minimizar problemas estruturais e semânticos.

Por outro lado, alguns *test smells* apresentaram pequenas divergências na percepção de sua criticidade, como *Dead Feature* e *Vicious Tag*. Enquanto a maioria dos respondentes reconheceu seus impactos negativos, uma minoria argumentou que, em alguns contextos específicos, essas práticas podem ter justificativas válidas. Essa variação nas respostas indica que a severidade desses *smells* pode depender do contexto de aplicação e das estratégias adotadas por diferentes equipes de testes.

De maneira geral, os resultados confirmam que a detecção e mitigação de *test smells* em BDD são aspectos fundamentais para garantir a qualidade dos testes e a eficiência no desenvolvimento orientado a comportamento. A pesquisa também reforça a necessidade de conscientização e treinamento contínuo dos profissionais para o reconhecimento e correção dessas más práticas, visando aprimorar a legibilidade, organização e confiabilidade dos testes automatizados.

9 PROPOSTA DE DETECÇÃO AUTOMATIZADA DE SMELLS

Esta etapa refere-se ao desenvolvimento de *scripts* para identificação automática dos *test smells* nos repositórios analisados. A ferramenta foi projetada para analisar arquivos Gherkin, permitindo a detecção dos padrões identificados na análise manual.

A validação da eficácia da ferramenta foi realizada utilizando uma amostra significativa de cenários agrupados por *features*, avaliando seu desempenho por meio de métricas *precision*, *recall* e *f-measure*. O agrupamento por funcionalidades é necessário, pois existem categorias que precisam do contexto geral do arquivo, como os *Background Smells*.

Essa proposta de solução proporciona uma análise mais precisa e escalável, possibilitando a identificação de padrões recorrentes e a mitigação de inconsistências nos testes escritos em BDD. Além disso, esta etapa responde diretamente às questões de pesquisa QP2, QP3, QP4 e QP5.

9.1 ABORDAGENS CONSIDERADAS

Por se tratar de um arquivo de texto em linguagem natural, a detecção de *test smells* no Gherkin exige uma solução que suporte variações na escrita. Durante a pesquisa, duas diferentes abordagens foram analisadas para responder à QP2: *"Quais são as formas de detecção de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?"*; das quais podem ser destacadas:

1. Uso de Inteligência Artificial (IA)
    - Permite uma abordagem inteligente e adaptável a cenários complexos e a diversas formas de escrita.
    - No entanto, apresenta alto custo computacional e dificuldade de implementação.
2. Uso de Expressões Regulares (Regex)
    - Possibilita uma detecção rápida e eficiente dos padrões estruturais.
    - Entretanto, é limitada para cenários contextuais mais complexos e é difícil de se adaptar a diferentes estilos de escrita.

Dadas as necessidades do estudo, optou-se pelo uso de regex, pois sua implementação é simples e eficaz, sendo capaz de identificar a maior parte dos *smells* descobertos na análise manual. Embora a estratégia do uso de IA seja uma alternativa prioritária para estudos futuros.

Outro fator crucial é a escolha da língua inglesa como idioma de detecção para o desenvolvimento dos *scripts* com regex, por se tratar de uma linguagem de desenvolvimento padrão entre os programadores, uma vez que de acordo com o referencial teórico da seção 3.2.2, o Gherkin proporciona suporte para mais de 70 idiomas, o que seria inviável de se implementar com a atual equipe deste trabalho.

9.2 IMPLEMENTAÇÃO DE SCRIPTS COM REGEX

A linguagem Python foi escolhida pela sua flexibilidade, facilidade de desenvolvimento, vasto repertório de bibliotecas especializadas e potencial de processamento para grandes volumes de dados.

Para centralizar o código e documentar a implementação, foi criado um repositório público no GitHub nomeado Smell Detector[7], contendo os *scripts* e amostras necessárias para execução desta etapa.

A ferramenta foi projetada para ser executada via *Command-line Interface* (CLI). Seu fluxo inicia com o arquivo “main.py”, que realiza a análise dos arquivos *feature* dos diretórios, podendo processar projetos individualmente ou como um todo, como a amostra definida no **APÊNDICE D**. O Código-Fonte 18 apresenta a estrutura do arquivo “main.py” da ferramenta *Smell Detector*.

Código-Fonte 18 - Composição do arquivo “main.py” da ferramenta

```
import runner
from utils import title
from tabulate import tabulate

projects = [
    "inukshuk_jekyll-scholar",
    "keygen-sh_keygen-api",
    "opencypher_openCypher",
```

[7]https://github.com/felipesantoos/smell-detector

```
    "scality_Zenko",
    "sdkman_sdkman-cli",
    "serverlessworkflow_specification",
    "thoughtbot_factory_bot_rails",
    "RUN ALL PROJECTS",
    "sample",
    "EXIT"
]

title("Select Project", "white")
if projects:
    options = [
        [i + 1, item]
        for i, item in enumerate(projects)
    ]
    print(tabulate(options, headers=["Project Index", "Description"], tablefmt="pretty"))
else:
    print("No content available")

try:
    choice = int(input("Choice: "))
    choice -= 1

    if -1 < choice < 7 or choice == 8:
        runner.execute_project(projects[choice])
    elif choice == 7:
        runner.execute_projects(projects[:-3])
    elif choice == 9:
        print("goodbye...")
    else:
        print("choice doesn't exist")
except Exception as e:
    print("ERROR: ", e)
```

Fonte: Os autores.

A análise dos projetos ocorre por meio do método “execute_projects(projects)”, pertencente ao arquivo “runner.py”, que itera sobre os métodos de busca dos *smells* definidos no estudo para cada *feature* de um projeto. Cada *smell* possui uma característica única detectável por regex. Para um

entendimento mais aprofundado, recomenda-se a análise do código da ferramenta Smell Detector disponível no GitHub, onde a implementação aprofundada de cada método de detecção pode ser consultada.

Os resultados da detecção dos *smells* foram gravados em arquivos *Comma-Separated Values* (CSV) e armazenados em um diretório nomeado “reports”. Essa abordagem permite que os usuários visualizem as informações em ferramentas de planilhas eletrônicas, como o Excel. O Código-Fonte 19 apresenta um trecho de código responsável pela geração e apresentação dos resultados em formato CSV de um dos *smells* deste estudo.

Código-Fonte 19 - Geração do arquivo CSV de um dos *smells*

```
# Generate CSV if filename is provided
    if csv_filename:
        report_dir = './reports'
        if not os.path.exists(report_dir):
            os.mkdir(report_dir)
        file_exists = os.path.isfile(csv_filename)  # Check if file already exists
        with open(csv_filename, mode='a', newline='', encoding='utf-8') as csvfile:
            csv_writer = csv.writer(csvfile)
            if not file_exists:  # Write header only if the file is new
                csv_writer.writerow(["Filename", "Line Number", "Matched Line"])  # Write header
            csv_writer.writerows(results)  # Write data
        print(f"Results saved to {csv_filename}.")
```

Fonte: Os autores.

9.3 DEFINIÇÃO E VALIDAÇÃO DA AMOSTRA DE FEATURES

A fórmula de dados categóricos de Cochran (Ii; Kotrlik; Higgins, 2001) foi utilizada para definir uma nova amostra de cenários. Dessa vez, os parâmetros considerados foram uma taxa de confiabilidade de 95% (i.e., *z-score* 1.96) e margem de erro de 5% considerando todas as *features* e scenarios válidos presentes na base de dados do estudo. O que resultou em uma população de 351 *scenarios* que seriam utilizados para validação automatizada da ferramenta.

Para a seleção dos casos de teste a serem analisados, foi utilizada uma consulta SQL que escolhe aleatoriamente as *features* armazenadas na base de

dados. Em seguida, os cenários associados a essas *features* são contabilizados e organizados de maneira randômica com base em um identificador gerado aleatoriamente.

A consulta retorna um conjunto de *features* cuja soma acumulada de cenários não ultrapassa 351. Esse critério garante que, em diferentes execuções do processo, um subconjunto aleatório de *features* seja selecionado, mantendo a soma dos seus cenários dentro do limite definido. Dessa forma, a seleção resultante pode, eventualmente, atingir exatamente 351 cenários, permitindo a validação da amostra escolhida para análise.

Essa abordagem assegura que a ferramenta desenvolvida seja testada com diferentes combinações aleatórias de funcionalidades, permitindo verificar seu comportamento e precisão sob diferentes amostras. O Código-Fonte 20 apresenta a estrutura da consulta de seleção aleatória de *features*.

Código-Fonte 20 - Consulta randômica de *cenários* por *feature*

```
WITH RandomizedFeatures AS (
  SELECT
    SUBSTRING(f.path, 89) AS feature_reference,
    COUNT(DISTINCT s.idscenario) AS number_scenarios,
    RAND() AS random_order
  FROM
    feature f
    JOIN scenario s ON s.feature_id = f.idfeature
    JOIN step ON step.scenario_id = s.idscenario
  GROUP BY
    f.idfeature, SUBSTRING(f.path, 89)
),
CumulativeSum AS (
  SELECT
    feature_reference,
    number_scenarios,
    SUM(number_scenarios) OVER (ORDER BY random_order) AS cumulative_scenarios
  FROM
    RandomizedFeatures
)
SELECT
```

```
    feature_reference,
    number_scenarios,
    cumulative_scenarios
FROM
    CumulativeSum
WHERE
    cumulative_scenarios <= 351
ORDER BY
    cumulative_scenarios desc;
```

Fonte: Os autores.

A consulta retorna o nome da *feature* de referência, o número de cenários de teste e a soma acumulada dos cenários para visualização dos autores. A Tabela 8 detalha a distribuição das amostras de *feature* com sua composição de *scenarios* por projeto.

Tabela 8 - Distribuição da amostra automatizada por projeto

Nome do repositório	*Features*	*Scenarios*
sdkman-cli	2 (5,26%)	22 (6,27%)
factory_bot_rails	1 (2,63%)	10 (2,85%)
jekyll-scholar	2 (5,26%)	8 (2,28%)
openCypher	24 (63,16%)	206 (58,69%)
keygen-api	6 (15,80%)	99 (28,20%)
specification	2 (5,26%)	2 (0,57%)
Zenko	1 (2,63%)	4 (1,14%)
Total	**38 (100%)**	**351 (100%)**

Fonte: Os autores.

A distribuição das *features* e *scenarios* nos repositórios segue um padrão similar ao da primeira amostra de *scenarios* da seção 6, reforçando que os projetos com maior representatividade de *scenarios* também possuem uma alta quantidade de *features*.

O openCypher domina a amostra com 63,16% das *features* e 58,69% dos *scenarios*. Isso confirma que o projeto contém uma grande quantidade de funcionalidades com bastante cenários, apesar de ser um número menor em comparação com o keygen-api.

O keygen-api é o segundo repositório mais relevante na amostra, com 15,80% das funcionalidades e 28,20% dos *scenarios*. Esse comportamento aponta uma cobertura excessiva dentro de uma mesma *feature*, abrindo possibilidade para a presença de *test smells*.

O projeto sdkman-cli (5,26% das *features* e 6,27% dos *scenarios*) e factory_bot_rails (2,63% e 2,85%) apresentam um grande número de cenários em suas features, o que pode ser um fator de desorganização como no keygen-api.

O repositório jekyll-scholar (5,26% das *features*, 2,28% dos *scenarios*) apresenta um comportamento mais equilibrado, indicando um uso moderado e bem distribuído da metodologia BDD no projeto.

Para os projetos specification e Zenko, eles continuam pouco representativos, contribuindo com menos de 3% da amostra de cenários cada. Esses números indicam que esses repositórios podem possuir features muito básicas, sem um número significativo de cenários associados.

Como restrição adicional, também foi considerada a validação dos dados descrita no Código-Fonte 5, garantindo que apenas cenários válidos fossem incluídos na amostra.

Com base nas definições anteriores, para validar a precisão e eficácia da ferramenta, foi realizada uma comprovação manual dos *test smells* detectados, abrangendo todos os cenários da amostra. O desempenho da ferramenta está organizado pelas seguintes categorias:

- **Verdadeiro Positivo (TP)**: A ferramenta detectou corretamente o *smell* no cenário analisado de uma *feature* erroneamente estruturada;
- **Verdadeiro Negativo (TN)**: A ferramenta não detectou *smell*, confirmando que o cenário analisado da *feature* estava corretamente estruturado;

- **Falso Positivo (FP)**: A ferramenta detectou incorretamente o *smell* no cenário corretamente estruturado da *feature*;
- **Falso Negativo (FN)**: A ferramenta não detectou corretamente o *smell* no cenário erroneamente estruturado da *feature*.

Esta validação é fundamental para responder às questões de pesquisa QP4: *“É possível detectar de forma automatizada smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?”;* e QP5: *“O quão eficiente é a detecção automatizada de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?”*.

Com base nessas categorias, foi possível calcular a acurácia da ferramenta, tanto para cada *smell* individualmente quanto para o conjunto completo de *smells* detectados.

9.4 ANÁLISE DE OCORRÊNCIA E EFICIÊNCIA DA DETECÇÃO

A execução da ferramenta em todas as *features* tem como objetivo responder à questão de pesquisa QP3: *"Qual a frequência dos smells encontrados em casos de teste escritos no padrão BDD com a linguagem Gherkin?"*. Os resultados a seguir apresentam a validação das métricas de acurácia (*precision*, *recall* e *f-measure)*, além das informações detalhadas acerca da frequência de *smells* encontrados em todos os registros do banco de dados. A Tabela 9 apresenta a frequência de cada *test smell* identificado na amostra de 351 cenários.

Tabela 9 - Ocorrências dos *smells* por cenário da amostra

Test Smell	Ocorrências em cenários
Untitled Feature	10 (5,10%)
Duplicate Feature Title	0 (0%)
Duplicate Scenario Title	5 (2,50%)
Absence of Background	91 (46%)
Vicious Tag	28 (14%)
Duplicate Step	0 (0%)
Malformed Test	53 (26,8%)
Starting With the Left Foot	1 (0,50%)
Duplicate Test Case	10 (5,10%)
Total	**198 (100%)**

Fonte: Os autores.

Com relação à ausência de *background*, observou-se que 91 casos de teste repetiam pré-condições que poderiam ser simplificadas. Essa representatividade sugere falta de conhecimento dos profissionais sobre as funcionalidades do Cucumber ou falta de familiaridade com a organização dos arquivos “.*feature*”.

A aparição relevante do *Malformed Tests* (26,8%) detalhou os desafios na padronização e organização dos testes BDD, principalmente em cenários que utilizam lógica de execução semelhante ou que procuram realizar múltiplas etapas de verificação e ação em um único cenário, reforçando as definições levantadas na seção 7.2.4.2 - *Malformed Test*.

O uso abusivo de etiquetas (*Vicious Tag*) também se destacou com 28 aparições de mais de uma *tag* redundante em todos os cenários de teste de *features* distintas. Esse comportamento leva a crer que os desenvolvedores desconhecem otimizações do Cucumber, como a herança de *tags*.

A aparição moderada de *Duplicate Scenario Title* e *Duplicate Test Case,* demonstraram problemas na organização das *features* ou falta de critérios claros para nomeação dos testes, que seguem de acordo com a seção 7.2 - Identificação e Categorização de *Test Smells*.

A aparição de 10 casos de *Untitled Feature* sugeriu que este problema ocorre em fases iniciais do desenvolvimento e não são resolvidos. Enquanto a presença mínima de *Starting With the Left Foot* (0,5%) confirmou que a maioria dos desenvolvedores compreende a estrutura correta dos cenários BDD. Em concordância, para os *smells* que tiveram zero ocorrências, a sua ausência indicou que possivelmente os desenvolvedores reconhecem a importância da nomeação exclusiva e evitam a repetição de passos, garantindo melhor legibilidade e organização dos testes.

De forma geral, percebe-se que quanto maior o tamanho do projeto, maior o risco de surgimento desses problemas, pois a complexidade do sistema cresce e a organização pode se perder ao longo do tempo. A Tabela 10 apresenta os resultados das métricas de validação da ferramenta.

Tabela 10 - Detalhamento das métricas da ferramenta

Test Smell	TP	TN	FP	FN	*Precision*	*Recall*	*F-measuse*
Untitled Feature	10	341	0	0	1	1	0
Duplicate Feature Title	0	351	0	0	0	0	0
Duplicate Scenario Title	5	346	0	0	1	1	0
Absence of Background	91	260	0	0	1	1	0
Vicious Tag	12	339	0	0	1	1	0
Duplicate Step	0	351	0	0	0	0	0
Malformed Test	20	330	1	0	0.9523	1	0.003
Starting With the Left Foot	1	350	0	0	1	1	0
Duplicate Test Case	10	341	0	0	1	1	0
Total	**149**	**3009**	**1**	**0**	**0.9934**	**1**	**0.0003**

Fonte: Os autores.

A ferramenta apresentou uma precisão geral de 99,34% para classificação correta de verdadeiros positivos, o que sugeriu uma detecção confiável, apesar de

uma definição de falso positivo. O *recall* atingiu 100%, que indica a proporção de todos os positivos reais que foram classificados corretamente, ou seja, não houveram *smells* ignorados pela ferramenta.

O baixo *f-measure* de 0,03% significou que a ferramenta evitou a classificação incorreta de negativos como positivos, tendo apenas um alarme falso dentre todos os dados da amostra. Além disso, a acurácia de 99,97% da ferramenta (calculada à parte), indicou o percentual de acertos gerais de positivos e negativos da amostra.

9.5 AMEAÇAS À VALIDADE DA SOLUÇÃO

Como ameaça à validade interna, os resultados da ferramenta podem conter erros, que foram mitigados pela análise manual dos 351 cenários pelos autores deste trabalho.

Como ameaças à validade do constructo, alguns *smells* podem não estar devidamente categorizados ou delimitados. Portanto, esperam-se estudos futuros que realizem essas distinções detalhadamente, tomando como ponto de partida esta pesquisa.

Como ameaça externa, apesar dos resultados promissores, não se pode generalizar os resultados para outros sistemas do mercado, em virtude da amostra de apenas 7 repositórios. Para minimizar esse problema, os repositórios selecionados possuíam diferentes propósitos e classificações relevantes no GitHub, como proposto pelo capítulo 6 - Prospecção e Coleta de Casos de Teste.

10 DESENVOLVIMENTO DA FERRAMENTA WEB

Este capítulo apresenta as tecnologias utilizadas no desenvolvimento da ferramenta *web* voltada para a identificação automática de *test smells* em arquivos escritos no formato “.feature”. Além disso, descreve a estrutura geral do sistema, destacando os principais aspectos da arquitetura, a comunicação entre as partes e as funcionalidades que compõem a aplicação. A ferramenta foi concebida para proporcionar uma experiência interativa e intuitiva aos usuários, permitindo a análise detalhada dos problemas encontrados nos testes e sua organização em um catálogo acessível.

10.1 TECNOLOGIAS UTILIZADAS

O desenvolvimento da aplicação contou com tecnologias modernas, tanto no ambiente do usuário quanto no processamento dos arquivos enviados. A interface foi construída utilizando React e TypeScript, permitindo um design dinâmico e uma experiência fluida. Para garantir um desempenho eficiente durante o desenvolvimento, foi utilizada a ferramenta Vite, que possibilita uma atualização rápida da interface sem a necessidade de recarregar toda a página. A organização visual foi estruturada com TailwindCSS, um framework que facilita a estilização dos elementos gráficos e a adaptação da interface para diferentes dispositivos. Para a navegação entre as diferentes telas da ferramenta, foi empregado o React Router DOM, assegurando um acesso intuitivo e fluido.

A exibição dos resultados da análise é fundamental para o entendimento dos problemas encontrados nos arquivos processados. Para isso, a ferramenta conta com gráficos interativos gerados com Chart.js e React-Chartjs-2, os quais permitem visualizar a distribuição e frequência dos *test smells* detectados. Além disso, um conjunto de ícones fornecidos pela biblioteca Lucide-react foi utilizado para representar de maneira visual e diferenciada cada tipo de problema identificado.

No processamento dos arquivos submetidos pelos usuários, foi adotada uma abordagem baseada em um servidor construído com Node.js e Express. Esse ambiente permite o gerenciamento das requisições e a execução das análises dos arquivos enviados. Para possibilitar o envio de múltiplos arquivos de uma só vez, foi utilizado o *middleware* Multer, que realiza a manipulação e armazenamento temporário dos documentos. A lógica de detecção dos *test smells* foi desenvolvida

em JavaScript, garantindo uma maior organização do código e facilitando futuras melhorias.

10.2 ARQUITETURA DO SISTEMA

A ferramenta foi estruturada como uma aplicação *web* de página única, o que significa que toda a interação do usuário ocorre dentro de uma mesma interface, sem a necessidade de recarregar a página a cada ação realizada. O funcionamento ocorre a partir da comunicação entre a interface e o servidor, onde os arquivos são analisados e os resultados retornados ao usuário de forma clara e organizada.

O processo de envio dos arquivos ocorre por meio de um componente que permite a seleção manual dos documentos ou o uso da funcionalidade de arrastar e soltar. Assim que os arquivos são submetidos, o sistema os encaminha para o servidor, que executa a análise e retorna os dados estruturados. Esses dados incluem informações detalhadas sobre os *test smells* encontrados, como sua frequência e distribuição, permitindo que o usuário visualize os resultados de forma intuitiva e objetiva.

A identificação dos *test smells* ocorre por meio de funções desenvolvidas especificamente para reconhecer padrões problemáticos nos arquivos *feature*. Entre os principais problemas detectados estão testes sem título, títulos duplicados, ausência da seção de *background*, repetição excessiva de passos e estruturação inadequada dos testes. O sistema agrupa essas informações e gera relatórios que podem ser consultados diretamente na interface da ferramenta.

10.3 RESULTADOS

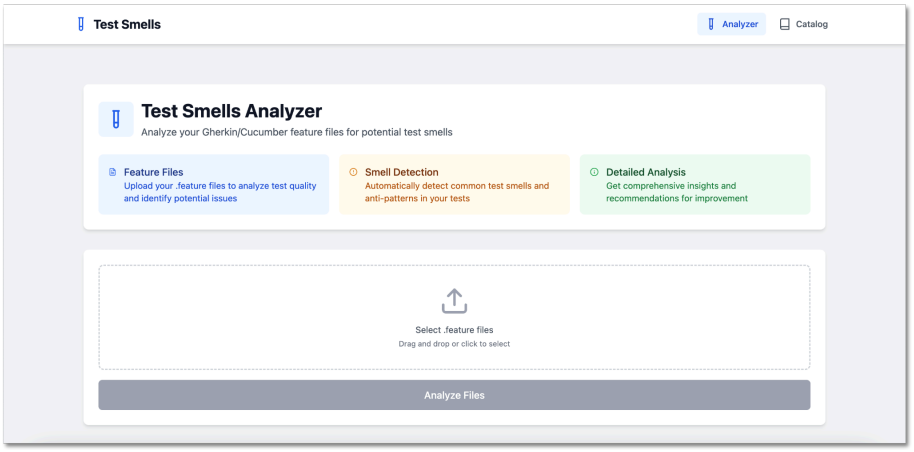
A implementação da ferramenta foi realizada com tecnologias modernas que garantem eficiência no processamento dos arquivos e uma interface interativa para a visualização dos resultados. A escolha dessas tecnologias foi fundamentada na necessidade de proporcionar uma experiência fluida e intuitiva para os usuários, assegurando que os *test smells* possam ser identificados e analisados de maneira clara e organizada.

A ferramenta disponibiliza uma série de funcionalidades que tornam a análise dos arquivos *feature* mais acessível e eficiente. O sistema permite o envio simultâneo de vários arquivos, realizando a validação automática para garantir que

estejam no formato correto antes do processamento. O servidor executa a análise e retorna um relatório estruturado em formato JSON, que é então interpretado e exibido na interface.

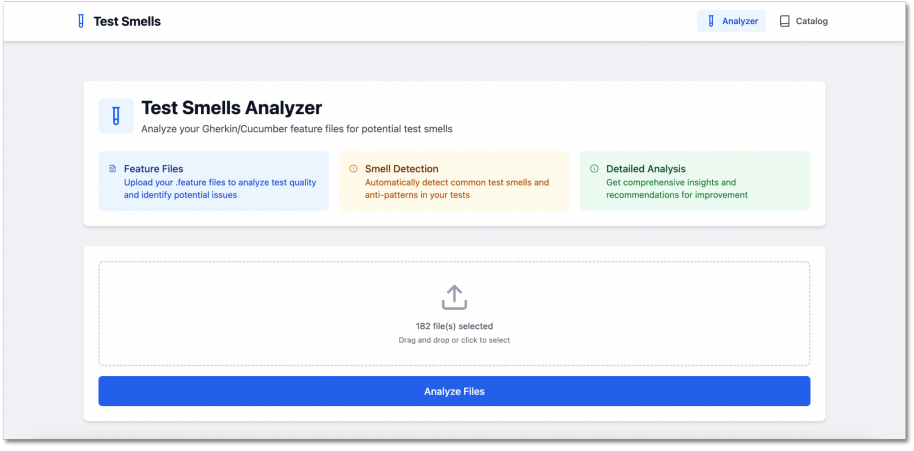
A tela inicial da ferramenta foi projetada para facilitar o envio dos arquivos *feature* que serão analisados. Nessa interface, o usuário pode fazer o *upload* dos documentos de maneira simples, utilizando a funcionalidade de arrastar e soltar ou selecionando manualmente os arquivos em seu dispositivo. A interface exibe indicações visuais sobre os formatos aceitos e o status do envio, garantindo uma experiência intuitiva e acessível para diferentes tipos de usuários.

Figura 19 - Página inicial da ferramenta *web* sem arquivos selecionados



Fonte: Os autores.

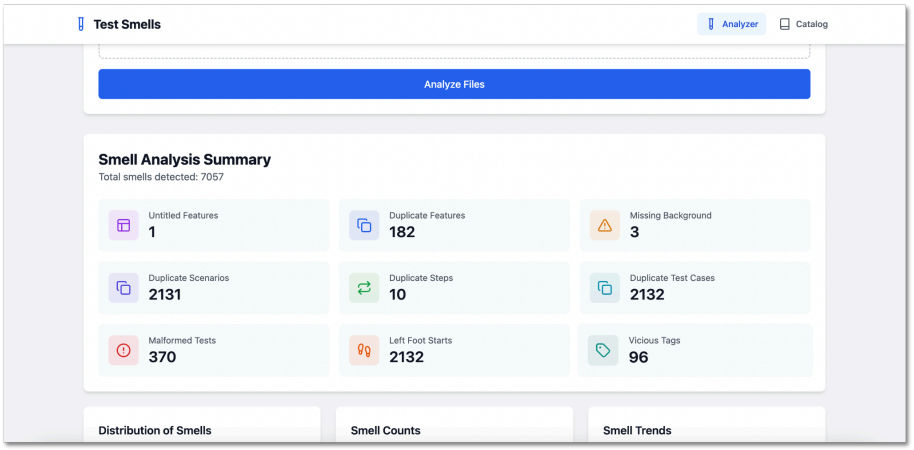
Figura 20 - Página inicial da ferramenta *web* com arquivos selecionados



Fonte: Os autores.

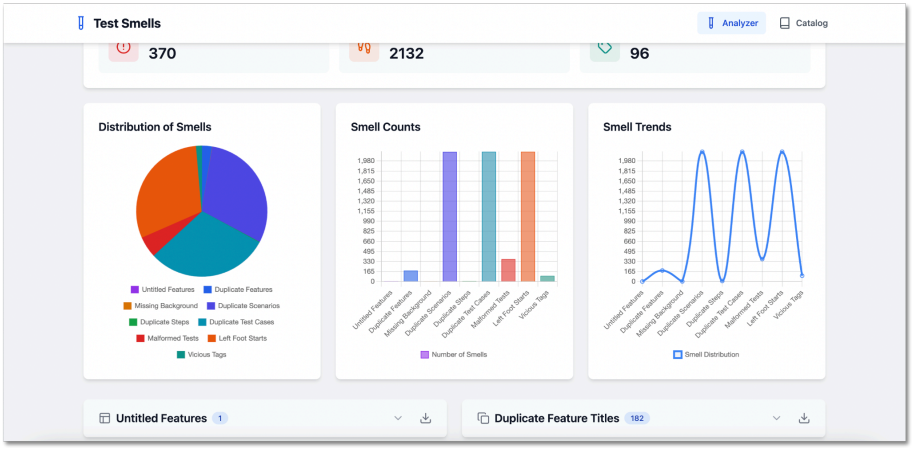
O painel principal da ferramenta apresenta um resumo dos problemas detectados, oferecendo uma visão quantitativa dos *test smells* encontrados. Esta *dashboard* exibe gráficos interativos que ajudam na interpretação dos resultados e permitem visualizar a distribuição e frequência dos problemas identificados. Com esta abordagem visual, o usuário pode identificar rapidamente quais são os *test smells* mais recorrentes em seus testes e analisar tendências dentro do conjunto de arquivos processados.

Figura 21 - Resumo da análise dos *smells* identificados pela ferramenta



Fonte: Os autores.

Figura 22 - Gráficos dos *smells* identificados pela ferramenta

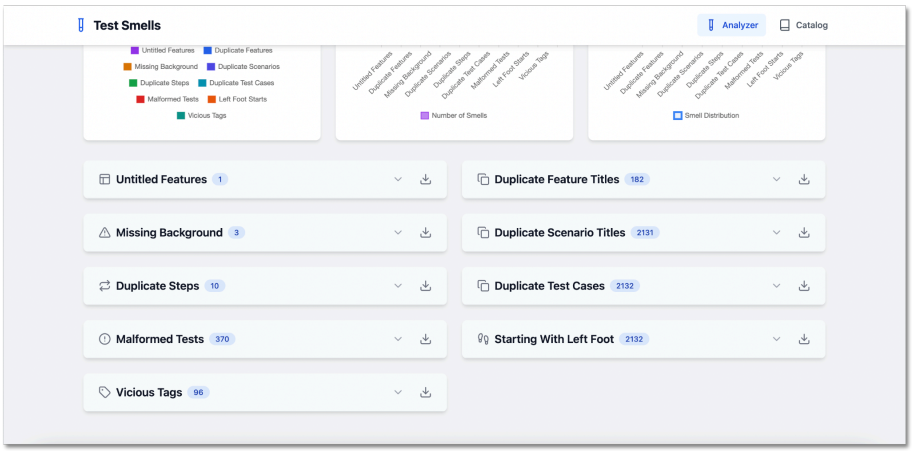


Fonte: Os autores.

Além do resumo geral, há uma seção dedicada à apresentação detalhada de cada *test smell* identificado. Essa área permite que o usuário visualize exatamente

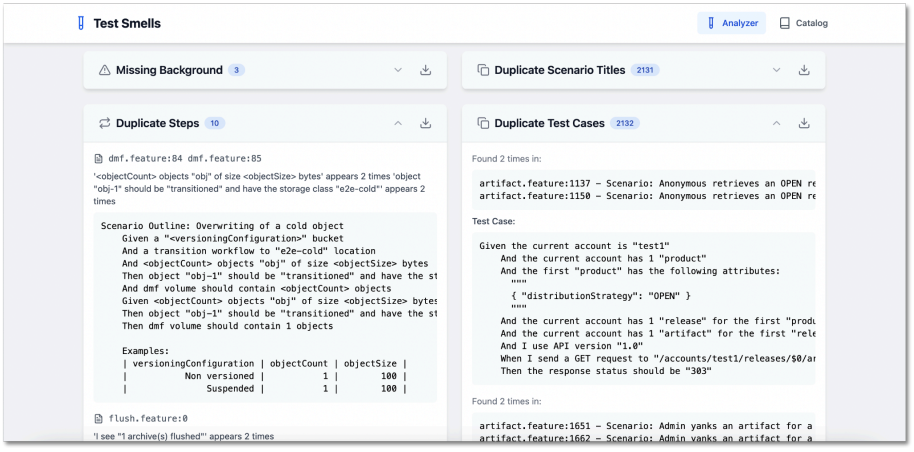
onde cada problema foi encontrado dentro dos arquivos *feature*, facilitando a compreensão e a correção dos testes. A exibição é feita de forma organizada, destacando as linhas afetadas e fornecendo explicações sobre os padrões problemáticos detectados.

Figura 23 - Lista de itens expansivos dos *smells* identificados pela ferramenta



Fonte: Os autores.

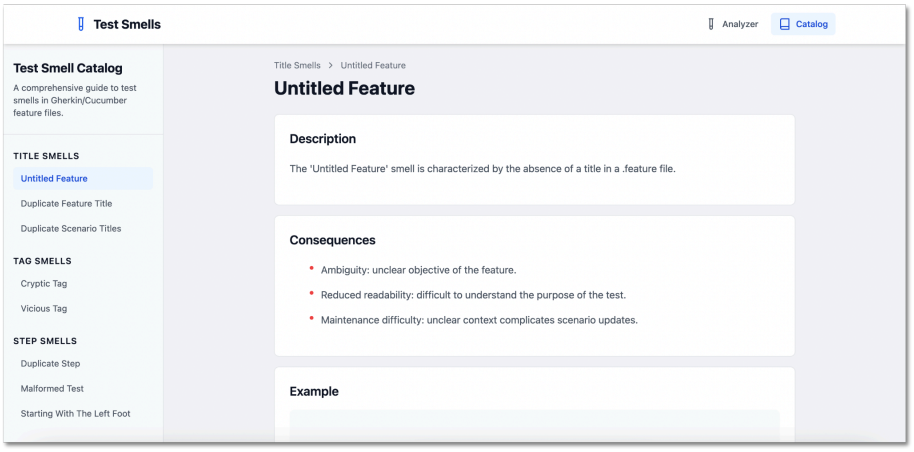
Figura 24 - Detalhes dos *smells* identificados pela ferramenta



Fonte: Os autores.

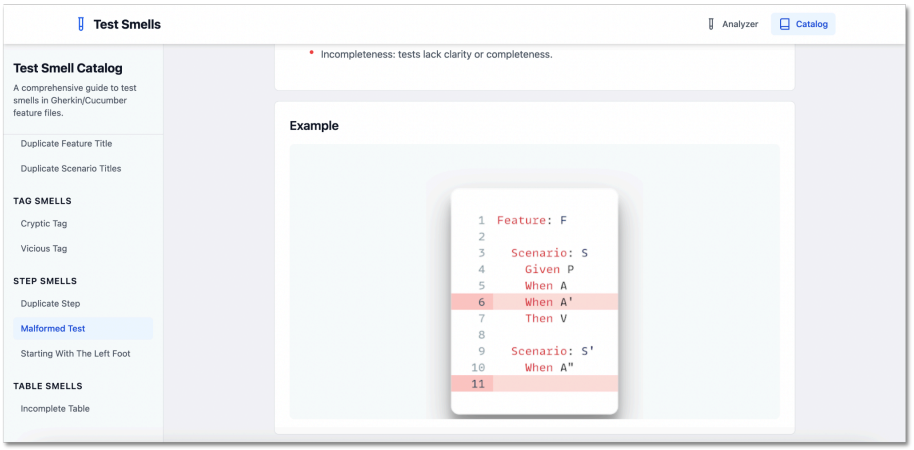
A ferramenta também conta com um catálogo interativo, onde cada *test smell* identificado é detalhado individualmente. Esse catálogo fornece títulos descritivos, explicações sobre o problema, possíveis consequências para a qualidade dos testes e exemplos ilustrativos que ajudam na interpretação do erro. Com essa abordagem, os usuários podem aprofundar seu conhecimento sobre cada *test smell* e entender como evitá-los em suas futuras implementações.

Figura 25 - Parte de cima do catálogo de *smells*



Fonte: Os autores.

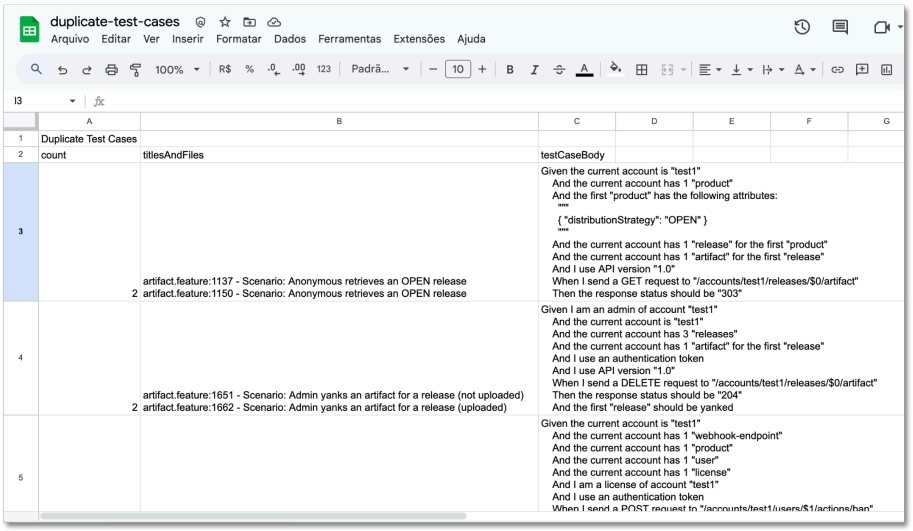
Figura 26 - Parte de baixo do catálogo de *smells*



Fonte: Os autores.

O sistema também oferece recursos para exportação dos resultados, permitindo que os usuários baixem os dados em formato CSV para análise posterior. Além disso, a navegação entre as diferentes seções da ferramenta é intuitiva, possibilitando alternar facilmente entre a visualização dos resultados e a consulta ao catálogo de *test smells*.

Figura 27 - CSV baixado através da ferramenta



duplicate-test-cases

Arquivo Editar Ver Inserir Formatar Dados Ferramentas Extensões Ajuda

	A	B	C
1	Duplicate Test Cases		
2	count	titlesAndFiles	testCaseBody
3	2	artifact.feature:1137 - Scenario: Anonymous retrieves an OPEN release artifact.feature:1150 - Scenario: Anonymous retrieves an OPEN release	Given the current account is "test1" And the current account has 1 "product" And the first "product" has the following attributes: """ { "distributionStrategy": "OPEN" } """ And the current account has 1 "release" for the first "product" And the current account has 1 "artifact" for the first "release" And I use API version "1.0" When I send a GET request to "/accounts/test1/releases/$0/artifact" Then the response status should be "303"
4	2	artifact.feature:1651 - Scenario: Admin yanks an artifact for a release (not uploaded) artifact.feature:1662 - Scenario: Admin yanks an artifact for a release (uploaded)	Given I am an admin of account "test1" And the current account is "test1" And the current account has 3 "releases" And the current account has 1 "artifact" for the first "release" And I use an authentication token And I use API version "1.0" When I send a DELETE request to "/accounts/test1/releases/$0/artifact" Then the response status should be "204" And the first "release" should be yanked
5			Given the current account is "test1" And the current account has 1 "webhook-endpoint" And the current account has 1 "product" And the current account has 1 "user" And the current account has 1 "license" And I am a license of account "test1" And I use an authentication token When I send a POST request to "/accounts/test1/users/$1/actions/ban"

Fonte: Os autores.

11 RESULTADOS E DISCUSSÃO

Este trabalho realizou uma análise abrangente de *test smells* em casos de teste escritos no padrão BDD utilizando a linguagem Gherkin. A pesquisa envolveu quatro etapas principais que se complementam para oferecer uma visão completa do problema e propor soluções.

Na primeira etapa de identificação e categorização, a análise manual de 158 cenários de teste selecionados aleatoriamente revelou 12 tipos distintos de *test smells*, organizados em cinco categorias principais. Os problemas mais frequentes foram relacionados à estruturação inadequada dos passos (*Step Smells*, 33,33%) e à falta de clareza na documentação (*Title Smells* e *Tag Smells*, 25% cada). Esses resultados indicam que as principais dificuldades na escrita de testes BDD estão na organização dos cenários e na manutenção da clareza dos requisitos.

A validação com 22 profissionais de testes confirmou a relevância dos *smells* identificados, com alto consenso (mais de 70% de concordância) sobre os impactos negativos de problemas como *Untitled Feature*, *Duplicate Scenario Title* e *Malformed Test*. Os comentários qualitativos destacaram a necessidade de padronização e a importância de seguir as boas práticas recomendadas pelo Cucumber. Alguns *smells*, como *Dead Feature* e *Cryptic Tag*, apresentaram opiniões mais divididas, sugerindo que seu impacto pode variar conforme o contexto de aplicação.

Para a detecção automatizada, desenvolveu-se uma ferramenta que alcançou excelentes índices de precisão (99,34%) e recall (100%), com apenas um falso positivo em 351 cenários analisados. A abordagem com expressões regulares mostrou-se eficaz para a maioria dos *smells*, embora problemas contextuais como *Incomplete Table* exijam análise manual. A ferramenta demonstrou especial eficácia na identificação de *Absence of Background* (46% de ocorrência) e *Malformed Test* (26,8%), enquanto alguns *smells* como *Duplicate Step* não foram encontrados na amostra analisada.

A ferramenta *web* desenvolvida como produto final deste trabalho oferece uma solução prática para a identificação de *test smells*, com funcionalidades como *upload* de arquivos *feature*, visualização interativa dos resultados e exportação em formato CSV. Sua interface intuitiva foi projetada para facilitar a adoção por equipes

de QA e desenvolvimento, promovendo a correção proativa de testes mal estruturados.

A discussão integrada dos resultados permite entender que: (1) problemas estruturais são comuns em testes BDD, especialmente relacionados à organização dos passos e documentação; (2) ferramentas automatizadas baseadas em regex são viáveis para detecção da maioria dos smells, embora soluções mais avançadas sejam necessárias para casos contextuais; e (3) há necessidade de maior conscientização e treinamento, já que muitos problemas persistem apesar de serem reconhecidos pelos profissionais.

12 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo principal prospectar a ocorrência de *test smells* em testes escritos no padrão BDD utilizando a linguagem Gherkin, bem como avaliar a viabilidade da detecção automatizada desses problemas por meio de uma ferramenta baseada em expressões regulares. A pesquisa adotou uma abordagem exploratória e empírica, partindo de uma revisão multivocal da literatura até a construção de uma solução prática validada por análise manual e pelo parecer de especialistas da área. Durante o estudo, buscou-se responder às seguintes questões de pesquisa:

- **QP1: "Quais smells podem ser encontrados em casos de teste escritos no padrão BDD com a linguagem Gherkin?"** A análise de uma amostra inicial de 158 cenários resultou na identificação de 12 *smells*, agrupados em 5 categorias distintas, revelando padrões recorrentes de má prática estrutural e semântica na escrita de testes em Gherkin.
- **QP2: "Quais são as formas de detecção de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?"** Foram consideradas abordagens com o uso de inteligência artificial e expressões regulares, sendo a segunda escolhida pela sua simplicidade de implementação, interpretabilidade e baixo custo computacional, que permitiu a detecção de 9 dos 12 *smells* encontrados.
- **QP3: "Qual a frequência dos smells encontrados em casos de teste escritos no padrão BDD com a linguagem Gherkin?"** A aplicação dos *scripts* desenvolvidos em uma amostra de 351 cenários, permitiu a identificação de 198 ocorrências de 9 *smells*, evidenciando a presença significativa desses problemas em testes BDD.
- **QP4: “É possível detectar de forma automatizada smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?”** A ferramenta proposta demonstrou ser capaz de identificar 9 *smells* de um total de 12 identificados inicialmente, o que valida a possibilidade de automatizar esse processo.
- **QP5: “O quão eficiente é a detecção automatizada de smells em casos de teste escritos no padrão BDD com a linguagem Gherkin?”** Com base nas métricas de *precision*, *recall* e *f-measure*, os resultados apontaram uma

detecção satisfatória dentro do escopo analisado neste estudo, reforçando o potencial de automação como apoio às equipes de projetos.

Os resultados apresentados ao longo deste estudo confirmam o que é apresentado na literatura, que *test smells* são uma realidade em BDD e impactam negativamente a clareza, a rastreabilidade e a manutenção dos testes. A adoção de boas práticas na escrita de testes em linguagem natural, aliada ao uso de ferramentas de apoio como a desenvolvida neste trabalho, pode mitigar esses impactos, contribuindo para a construção de cenários de teste mais claros, objetivos e sustentáveis ao longo do tempo.

Assim, este trabalho busca não apenas avançar o conhecimento acadêmico sobre o tema, mas também oferecer uma solução prática para os desafios enfrentados por equipes de projetos de *software* que adotam o BDD.

12.1 CONTRIBUIÇÕES DA PESQUISA

Os achados desta pesquisa podem oferecer contribuições relevantes para a academia e a indústria de *software*, como o mapeamento sistemático de *test smells* em BDD; desenvolvimento de uma ferramenta automatizada para detecção de *test smells*; fornecimento de um conjunto de métricas quantitativas sobre a frequência e impacto dos *test smells* em projetos de *software* reais do GitHub, auxiliando na priorização de correções; contribuição para o catálogo de Soares (2023), detalhando mais *smells* em casos de teste manuais, documentando suas características, consequências e possíveis soluções, que pode servir como referência para profissionais da área.

12.2 TRABALHOS FUTUROS

Com base nos achados desta pesquisa, pretende-se que estudos futuros explorem a expansão da análise para um número maior de repositórios, garantindo uma amostra mais abrangente e representativa da prática de BDD no mercado; investigação de novas abordagens para a detecção de *test smells*, incluindo técnicas baseadas em aprendizado de máquina ou análise semântica avançada; desenvolvimento de uma ferramenta de refatoração automática, que não apenas identifique os *smells*, mas também sugira ou aplique correções automaticamente.

É necessário mais estudos sobre o impacto dos *test smells* na qualidade de *software* que avaliem sua influência em métricas como manutenibilidade, legibilidade e eficácia dos testes. Tendo em vista a grande quantidade de resultados obtidos nessa pesquisa de amostra limitada, ainda há muito do que se extrair sobre o tema.

REFERÊNCIAS

BROWN, W. J.; MALVEAU, R. C.; MCCORMICK, H. W.; MOWBRAY, T. J. **AntiPatterns: refactoring software, architectures, and projects in crisis**. New York: Wiley, 1998.

CHACON, S.; STRAUB, B. **Pro Git**. 2. ed. New York: Apress, 2023.

CUCUMBER. **Gherkin Reference**. [S. l.]: Cucumber, [s. d.]. Disponível em: https://cucumber.io/docs/gherkin/reference. Acesso em: 13 mar. 2025.

CUCUMBER. **Introduction**. [S. l.]: Cucumber, 2024. Disponível em: https://cucumber.io/docs. Acesso em: 3 mar. 2025.

ENGLAND, T. **Cucumber anti-patterns (part one)**. [S. l.]: Cucumber, 2016. Disponível em: https://cucumber.io/blog/bdd/cucumber-antipatterns-part-one. Acesso em: 7 fev. 2025.

FAROOQ, M. S. *et al.* **Behavior driven development: a systematic literature review**. *IEEE Access*, v. 11, p. 88008–88024, 2023. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10210040. Acesso em: 3 mar. 2025.

FREEMAN, R. E. **Strategic management: a stakeholder approach**. Boston: Pitman, 1984.

GAROUSI, V.; FELDERER, M.; MÄNTYLÄ, M. V. **Guidelines for including grey literature and conducting multivocal literature reviews in software engineering**. *Information and Software Technology*, v. 106, p. 101–121, 2019. Disponível em: https://www.researchgate.net/publication/318336961_Guidelines_for_including_the_grey_literature_and_conducting_multivocal_literature_reviews_in_software_engineering. Acesso em: 7 fev. 2025.

GAROUSI, V.; KÜÇÜK, Z. **Smells in software test code: a survey of knowledge in industry and research**. *Journal of Systems and Software*, v. 138, p. 52–81, 2018. Disponível em: https://pureadmin.qub.ac.uk/ws/portalfiles/portal/178889150/MLR_Smells_in_test_code_Dec_9.pdf. Acesso em: 7 fev. 2025.

II, J. E. B.; KOTRLIK, J. W.; HIGGINS, C. C. **Organizational research: determining appropriate sample size in survey research**. *Information Technology, Learning, and Performance Journal*, v. 19, n. 1, p. 43, 2001. Disponível em: https://www.opalco.com/wp-content/uploads/2014/10/Reading-Sample-Size1.pdf. Acesso em: 10 mar. 2025.

JUHNKE, K.; NIKIC, A.; TICHY, M. **Clustering natural language test case instructions as input for deriving automotive testing DSLs**. *Journal of Object Technology*, v. 20, n. 3, p. 5:1–14, 2021. Disponível em: https://www.jot.fm/issues/issue_2021_03/article5.pdf. Acesso em: 3 mar. 2025.

KFOURI, T. O. **Análise de projetos no GitHub que utilizam Behavior Driven Development**. 2019. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade de Brasília, Brasília, 2019. Disponível em: https://bdm.unb.br/bitstream/10483/24419/1/2019_TiagoDeOliveiraKfouri_tcc.pdf. Acesso em: 8 mar. 2025.

KULESOVS, I. **iOS applications testing**. In: INTERNATIONAL SCIENTIFIC AND PRACTICAL CONFERENCE, 2015. *Proceedings...* [S.l.: s.n.], 2015. (ISPC, v. 3), p. 138–150. Disponível em: https://journals.rta.lv/index.php/ETR/issue/view/27/32. Acesso em: 7 fev. 2025.

MARABESI, M.; GARCÍA-HOLGADO, A.; GARCÍA-PEÑALVO, F. J. **Exploring the connection between the TDD practice and test smells—A systematic literature review**. *Computers*, v. 13, n. 3, p. 79, 2024. Disponível em: https://www.mdpi.com/2073-431X/13/3/79. Acesso em: 3 mar. 2025.

ROMPAEY, B. V.; BOIS, B. D.; DEMEYER, S.; RIEGER, M. **On the detection of test smells: a metrics-based approach for general fixture and eager test**. *IEEE Transactions on Software Engineering*, v. 33, n. 12, p. 800–817, 2007. Disponível em: https://www.researchgate.net/publication/3189802_On_The_Detection_of_Test_Smells_A_Metrics-Based_Approach_for_General_Fixture_and_Eager_Test. Acesso em: 7 fev. 2025.

SOARES, E. A. **A multimethod study of test smells: cataloging, removal, and new types**. 2023. Tese (Doutorado em Ciência da Computação) – Universidade

Federal de Pernambuco, Recife, 2023. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/52055/1/TESE%20Elvys%20Alves%20Soares.pdf. Acesso em: 24 mar. 2024.

SOARES, E. *et al.* **Manual tests do smell! Cataloging and identifying natural language test smells**. In: ACM/IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT (ESEM), 2023. *Proceedings...* IEEE, 2023. p. 1–11. Disponível em: https://ieeexplore.ieee.org/abstract/document/10304800. Acesso em: 3 mar. 2025.

SUNDBERG, T. **Common BDD anti-patterns**. 2016. Disponível em: https://www.thinkcode.se/blog/2016/06/22/cucumber-antipatterns. Acesso em: 7 fev. 2025.

TAHIR, A. *et al.* **A large scale study on how developers discuss code smells and anti-pattern in Stack Exchange sites**. *Information Software Technology*, v. 125, p. 106333, 2020. Disponível em: https://www.researchgate.net/publication/341208075_A_large_scale_study_on_how_developers_discuss_code_smells_and_anti-pattern_in_Stack_Exchange_sites. Acesso em: 7 fev. 2025.

TOLEDO, F. *et al.* **BDD test smells and their impact**. 2019. Disponível em: https://dzone.com/articles/a-guide-to-good-cucumber-practices. Acesso em: 7 fev. 2025.

TOM, E.; AURUM, A.; VIDGEN, R. **An exploration of technical debt**. *Journal of Systems and Software*, v. 86, n. 6, p. 1498–1516, 2013. Disponível em: https://www.researchgate.net/publication/256991988_An_exploration_of_technical_debt. Acesso em: 7 fev. 2025.

VAN DEURSEN, A. *et al.* **Refactoring test code**. In: INTERNATIONAL CONFERENCE ON EXTREME PROGRAMMING AND FLEXIBLE PROCESSES IN SOFTWARE ENGINEERING (XP), 2., 2001. *Proceedings...* Citeseer, 2001. p. 92–95. Disponível em: https://www.researchgate.net/publication/2534882_Refactoring_Test_Code. Acesso em: 3 mar. 2025.

APÊNDICE A - TEST SMELLS ENCONTRADOS NA LITERATURA

Quadro 2 - Test smells encontrados na literatura

Nº	*Test Smell*	Descrição	Ocorrências
1	*UI-Coupled Steps*	O teste contém muitas etapas acopladas a elementos da interface do usuário, tornando-o frágil a mudanças na UI.	23
2	*Misconception: "BDD is for Testing"*	Uso de ferramentas como Cucumber apenas para automação, sem a fase essencial de discussão e refinamento dos cenários, tornando os testes fracos e pouco úteis.	10
3	*Overly Detailed Steps*	O cenário contém muitos passos detalhados, dificultando a manutenção e reutilização.	10
4	*Lack of Scenario Collaboration*	Os analistas de negócio (BAs) inicialmente participam das conversas e escrevem cenários, mas depois perdem o interesse e começam a escrever os cenários diretamente na ferramenta sem colaboração com os devs.	9
5	*Multi-Rule Testing*	A combinação excessiva de múltiplas ações em um único passo de teste pode comprometer a clareza e dificultar a depuração de falhas.	7
6	*Overly Abstract Scenarios*	Cenários sem detalhes concretos, dificultando a automação e a validação dos testes.	5
7	*Actor Misuse*	Usar o pronome pessoal "I" em cenários pode causar confusão sobre quais atores do sistema estão interagindo.	4
8	*Ambiguous Phrases*	Testes escritos em linguagem natural contêm formulações ambíguas, dificultando a interpretação dos resultados esperados.	3

9	*Treating Scenarios as Requirements*	Assumir que cada cenário BDD representa um requisito completo, resultando em especificações rígidas e mal compreendidas.	3
10	*Useless Scenario Descriptions*	Descrições de cenários que não acrescentam informação útil sobre a funcionalidade sendo testada.	3
11	*Excessive Scenario Outlines*	Excesso de exemplos em Scenario Outline pode gerar testes muito lentos e difíceis de manter.	2
12	*Fixed Waits Usage*	Adicionar tempos de espera fixos nos cenários (And I wait 10 seconds), causando testes lentos e não confiáveis.	2
13	*Forever Closed Scenarios*	Muitos cenários redundantes ou que testam o mesmo comportamento, resultando em feature files inchadas e difíceis de manter.	2
14	*Hardcoded Test Data*	O uso de dados fixos nos testes dificulta a manutenção e reduz a flexibilidade dos testes automatizados.	2
15	*Multiple Whens in One Scenario*	Criar vários passos When na sequência, tornando o cenário confuso e difícil de entender.	2
16	*No Clear Given/When/Then Separation*	Mistura de Given, When e Then, dificultando a leitura e a estruturação lógica dos testes.	2
17	*Poor or Missing Scenario Name*	Criar cenários sem títulos descritivos ou com nomes genéricos que não representam claramente a intenção do teste.	2
18	*Writing Scenarios After Code*	Criar cenários Gherkin depois que o código já foi escrito, transformando BDD em apenas uma ferramenta de automação e não um processo colaborativo.	2

19	*BDD as Correctness Guarantee*	Uso inadequado do termo "ensure" em cenários BDD, implicando que a execução do cenário garante a correção do software.	2
20	*BDD Treated as Documentation*	Assumir que BDD substitui documentação formal, criando cenários com excesso de detalhes ou ambiguidades.	1
21	*Coupled Scenarios*	Quando um cenário depende implicitamente de outro para ser executado corretamente, como quando um teste requer que um estado anterior seja configurado em outro teste.	1
22	*Duplicated NFRs (Non-Functional Requirements)*	Criar múltiplos cenários separados para testar a mesma funcionalidade em diferentes dispositivos ou contextos.	1
23	*Event Sequence as Precondition*	Listar ações sequenciais na cláusula Given, misturando fluxo do sistema com pré-condições.	1
24	*Excessive Background Usage*	Quando o Background contém muitos passos, tornando difícil entender os cenários individuais.	1
25	*Excessive Tool Focus*	Adoção de ferramentas como Cucumber apenas pela tecnologia, sem considerar o valor real que agregam ao processo.	1
26	*Generic CRUD Scenario*	Criar uma única feature para operações CRUD, dificultando a compreensão do propósito específico de cada cenário.	1
27	*Ignoring Feature Narrative Section*	Não utilizar a seção inicial do arquivo de feature para contextualizar o comportamento esperado.	1
28	*Overcomplicated or Bloated Phrases*	O uso de frases excessivamente complexas ou prolixas nos testes torna difícil a compreensão e manutenção.	1

29	*Relying on Specific Production Data*	Criar cenários que dependem de dados reais da produção, tornando os testes frágeis e difíceis de reproduzir.	1
30	*Scrum is Not Agile*	Assumir que apenas seguir práticas de Scrum significa que a equipe está adotando Agile de forma efetiva, sem integrar boas práticas de desenvolvimento.	1
31	*Test Clones*	Duplicação de trechos de testes, levando a esforço desnecessário na manutenção.	1
32	*Test Flow Branches*	Testes que contêm lógica condicional (if-else) ou fluxos divergentes, dificultando a previsibilidade dos resultados.	1
33	*Test-Specific Steps*	Passos escritos de forma muito específica para um único teste, dificultando a reutilização em outros testes.	1

APÊNDICE B - CENÁRIOS DEFINIDOS NA AMOSTRAGEM MANUAL

Quadro 3 - Scenarios da amostragem manual

Feature	***Scenario***
keygen-sh_keygen-api\validations.feature	Environment validates a shared license (in isolated environment)
keygen-sh_keygen-api\validations.feature	Anonymous validates a license key scoped to a mismatched machine
keygen-sh_keygen-api\validations.feature	Anonymous validates a valid license key scoped to an array of nil fingerprints
keygen-sh_keygen-api\licenses.feature	Product retrieves a license for a user
keygen-sh_keygen-api\typed_params.feature	User sends a request containing an unpermitted parameter
inukshuk_jekyll-scholar\filter.feature	LaTeX Superscript as HTML
keygen-sh_keygen-api\password.feature	User of another account attempts to update password for another user
keygen-sh_keygen-api\artifacts.feature	User retrieves a LICENSED release with a license for it
opencypher_openCypher\Return3.feature	[1] Returning multiple expressions
opencypher_openCypher\Delete6.feature	[8] Limiting to zero results after deleting relationships affects the result set but not the side effects
keygen-sh_keygen-api\group.feature	Admin changes a user's group relationship to a group for another account
keygen-sh_keygen-api\validations.feature	License quick validates their license
keygen-sh_keygen-api\validations.feature	An admin quick validates a valid license that requires a fingerprint scope
keygen-sh_keygen-api\validations.feature	Admin validates a strict license that has too many machine cores (ALLOW_2X_OVERAGE overage strategy, within overage allowance)
keygen-sh_keygen-api\artifacts.feature	License retrieves the artifact for a release of a different product

keygen-sh_keygen-api\artifacts.feature	License retrieves the artifact for a release of their product (expired before release, restrict access)
keygen-sh_keygen-api\artifact.feature	License retrieves a CLOSED release without a license for it
keygen-sh_keygen-api\policy.feature	Admin changes an unencrypted license's policy relationship to a new encrypted policy
keygen-sh_keygen-api\generate.feature	Shared admin generates a token for an isolated environment
keygen-sh_keygen-api\second_factors.feature	Admin creates a second factor while having 2FA disabled and providing a correct OTP
keygen-sh_keygen-api\generate.feature	Sales generates a new token via basic authentication
keygen-sh_keygen-api\validations.feature	Admin quick validates a check-in license that is valid
inukshuk_jekyll-scholar\citation.feature	A prefixed citation
thoughtbot_factory_bot_rails\fixture_replacement_config.feature	Disable Factory Bot generator
opencypher_openCypher\ReturnOrderBy5.feature	[1] Renaming columns before ORDER BY should return results in ascending order
keygen-sh_keygen-api\releases.feature	User attempts to retrieve the releases for their product
keygen-sh_keygen-api\password.feature	User attempts to reset password with missing parameters
keygen-sh_keygen-api\upgrades.feature	User retrieves an upgrade for a release of their product (missing some entitlements)
opencypher_openCypher\Delete6.feature	[3] Skipping and limiting to a few results after deleting nodes affects the result set but not the side effects
keygen-sh_keygen-api\checkouts.feature	Product performs a checkout for a machine of another product (POST)
sdkman_sdkman-cli\hooks.feature	Post-install Hook returns a non-zero code

serverlessworkflow_specification\branch.feature	Fork Task With Competing Concurrent Sub Tasks
keygen-sh_keygen-api\checkouts.feature	License performs a machine checkout (POST)
keygen-sh_keygen-api\upgrade.feature	Anonymous retrieves an upgrade for a shared open release (shared env)
keygen-sh_keygen-api\generate.feature	Global admin generates a token for a global environment (from an isolated environment)
keygen-sh_keygen-api\generate.feature	Support generates a new token via basic authentication
keygen-sh_keygen-api\revoke.feature	Admin revokes another user's token
opencypher_openCypher\Literals7.feature	[5] Return a list containing a hexadecimal integer
sdkman_sdkman-cli\offline_mode.feature	Display the current version of a candidate while in Offline Mode
keygen-sh_keygen-api\heartbeats.feature	License pings a process's heartbeat
keygen-sh_keygen-api\create.feature	Admin creates a webhook endpoint with missing url
keygen-sh_keygen-api\bans.feature	License bans a user
opencypher_openCypher\New.feature	Return literal Boo
keygen-sh_keygen-api\regenerate.feature	License resets their current token while authenticating with a license key
opencypher_openCypher\Map2.feature	[3] Dynamically access a field on null results in null
opencypher_openCypher\Foo.feature	Outline Test
keygen-sh_keygen-api\generate.feature	User attempts to generate a new token with custom permissions (ent tier, CE)
opencypher_openCypher\Match5.feature	[23] Handling a variable length relationship and a standard relationship in chain, longer 1
keygen-sh_keygen-api\validations.feature	Anonymous validates a valid license key scoped to a global environment
keygen-sh_keygen-api\event.feature	A pending account receives a "customer.subscription.updated" event with a "trialing" status

opencypher_openCypher\ReturnSkipLimit1.feature	[4] Accept skip zero
keygen-sh_keygen-api\releases.feature	User attempts to retrieve the releases for a product
inukshuk_jekyll-scholar\grouping.feature	Type Names
opencypher_openCypher\Delete1.feature	[4] Delete on null node
keygen-sh_keygen-api\tokens.feature	Global admin generates a global token for themself (from a shared environment)
opencypher_openCypher\Create2.feature	[21] Fail when creating a relationship with more than one type
opencypher_openCypher\WithOrderBy2.feature	[24] Sort by an expression that is only partially orderable on a non-distinct binding table, but made distinct
opencypher_openCypher\Quantifier4.feature	[15] Fail all quantifier on type mismatch between list elements and predicate
opencypher_openCypher\Match6.feature	[15] Variable-length named path
keygen-sh_keygen-api\validations.feature	A user validates a license scoped to their own email
opencypher_openCypher\Merge5.feature	[9] Creating relationship using merged nodes
keygen-sh_keygen-api\top-urls-by-volume.feature	Endpoint should be inaccessible when account is disabled
opencypher_openCypher\String10.feature	[4] Finding strings containing whitespace
opencypher_openCypher\Match6.feature	[10] Named path with alternating directed/undirected relationships
inukshuk_jekyll-scholar\removeduplicate.feature	Remove duplicates by year
keygen-sh_keygen-api\validations.feature	Anonymous validates a license key and sends a nonce
opencypher_openCypher\Precedence1.feature	[24] Null predicates take precedence over boolean negation on every truth values
opencypher_openCypher\Match7.feature	[20] Variable length optional relationships with bound nodes, no matches
sdkman_sdkman-cli\upgrade_candidate.feature	Don't update all upgradable candidates versions and set them as default

keygen-sh_keygen-api\generate.feature	Admin generates a product token
opencypher_openCypher\Call2.feature	[5] Standalone call to procedure should fail if input type is wrong
opencypher_openCypher\Quantifier1.feature	[6] None quantifier on list literal containing lists
keygen-sh_keygen-api\destroy.feature	Anonymous user attempts to delete a webhook event for their account
keygen-sh_keygen-api\validations.feature	Admin validates a strict node-locked license that has too many machines (ALWAYS_ALLOW_OVERAGE overage strategy)
opencypher_openCypher\Graph3.feature	[6] `labels()` should accept type Any
keygen-sh_keygen-api\limits.feature	Endpoint should be accessible when account is subscribed and has exceeded its daily request limit
opencypher_openCypher\List5.feature	[7] IN should return false when types of LHS and RHS don't match - singleton list
sdkman_sdkman-cli\per_project_configuration.feature	The env clear subcommand is issued without an active project configuration
keygen-sh_keygen-api\upgrades.feature	License retrieves an upgrade for a release of their product (expired, access allowed)
opencypher_openCypher\MatchWhere1.feature	[5] Filter end node of relationship with property predicate on multi variables with multiple bindings
keygen-sh_keygen-api\billing.feature	Admin applies a coupon to their account
keygen-sh_keygen-api\upgrades.feature	License upgrades a release with a download expiration basis (not set)
keygen-sh_keygen-api\artifact.feature	Product retrieves the artifact for a release of their product (1 week TTL)
keygen-sh_keygen-api\bans.feature	Environment unbans a shared user
keygen-sh_keygen-api\search.feature	Environment performs a search
opencypher_openCypher\Literals4.feature	[7] Return a long negative octal integer
opencypher_openCypher\Quantifier7.feature	[4] Any quantifier is equal the boolean negative of the none quantifier

keygen-sh_keygen-api\platforms.feature	Product retrieves the platforms for a product
keygen-sh_keygen-api\update.feature	A new candidate is available
keygen-sh_keygen-api\license.feature	Admin attempts to retrieve the license for a process of another account
keygen-sh_keygen-api\search.feature	Admin performs a search by machine type on the name attribute using a suffix
keygen-sh_keygen-api\tokens.feature	Shared admin generates a global token for themself (from a shared environment)
opencypher_openCypher\Return2.feature	[9] Returning a projected map
opencypher_openCypher\Literals8.feature	[21] Fail on a map containing key with dot
sdkman_sdkman-cli\install_sdkman.feature	Add Init Snippet to the .profile if present
opencypher_openCypher\MatchWhere5.feature	[3] Filter out on null if the AND'd predicate evaluates to true
opencypher_openCypher\Quantifier2.feature	[15] Single quantifier is true if the predicate is statically true and the list has exactly one non-null element
inukshuk_jekyll-scholar\cite_details.feature	A Simple Cite Details Link With A Text Argument
keygen-sh_keygen-api\permits.feature	Admin revokes a license
keygen-sh_keygen-api\upgrade.feature	License retrieves an upgrade for a release of their product (upgrade available)
opencypher_openCypher\WithWhere7.feature	[2] WHERE sees a variable bound after but not before WITH
keygen-sh_keygen-api\tokens.feature	User generates a user token
opencypher_openCypher\List3.feature	[6] Equality between different nested lists with null should return false
keygen-sh_keygen-api\group.feature	Admin removes a user's group relationship
opencypher_openCypher\Pattern2.feature	[7] Use a pattern comprehension inside a list comprehension
opencypher_openCypher\Union3.feature	[2] Failing when mixing UNION ALL and UNION

keygen-sh_keygen-api\permits.feature	Admin attempts to renew a license for another account
keygen-sh_keygen-api\validations.feature	Anonymous validates a license using scheme RSA_2048_PKCS1_PSS_SIGN by key using the legacy encrypted flag
opencypher_openCypher\Boolean3.feature	[7] Exclusive disjunction is associative on null
opencypher_openCypher\Literals5.feature	[16] Return a positive float with negative lower case exponent
opencypher_openCypher\Quantifier5.feature	[5] None quantifier is equal whether the size of the list filtered with same the predicate is zero
opencypher_openCypher\Precedence4.feature	[4] String predicate takes precedence over binary boolean operator
keygen-sh_keygen-api\validations.feature	Anonymous validates a valid license key scoped to an isolated environment (no environment header)
serverlessworkflow_specification\call.feature	Call HTTP Using Basic Authentication
keygen-sh_keygen-api\licenses.feature	License attempts to retrieve their user's licenses
keygen-sh_keygen-api\publish.feature	Admin publishes a published release for their account
opencypher_openCypher\Boolean5.feature	[8] De Morgan's law on non-null the negation of a conjunction is the disjunction of the negations
keygen-sh_keygen-api\validations.feature	Environment validates a shared license (in nil environment)
keygen-sh_keygen-api\owners.feature	Environment detaches shared owners from a shared group
opencypher_openCypher\List12.feature	[2] Collect and filter using a list comprehension
opencypher_openCypher\Literals5.feature	[12] Return a very long negative float without integer digits
keygen-sh_keygen-api\processes.feature	Shared environment retrieves the license for a global machine
inukshuk_jekyll-scholar\string.feature	Join strings
opencypher_openCypher\Null2.feature	[5] IS NOT NULL on a map

opencypher_openCypher\WithWhere3.feature	[2] Join between node properties of disconnected nodes
keygen-sh_keygen-api\machine.feature	Anonymous attempts to retrieve a processes's license
opencypher_openCypher\Precedence3.feature	[4] List appending takes precedence over list element containment
keygen-sh_keygen-api\create.feature	User creates a webhook endpoint for their account
keygen-sh_keygen-api\search.feature	Admin performs a search by request log type on requestor ID (full)
opencypher_openCypher\Return5.feature	[1] DISTINCT inside aggregation should work with lists in maps
inukshuk_jekyll-scholar\page_config.feature	Template access to page config
keygen-sh_keygen-api\validations.feature	Environment validates a global license (in shared environment)
sdkman_sdkman-cli\offline_mode.feature	Enable Offline Mode with internet reachable
keygen-sh_keygen-api\owners.feature	License attempts to attach owners to a group (is not member)
opencypher_openCypher\Map1.feature	[6] Fail when performing property access on a non-map
opencypher_openCypher\List1.feature	[3] Use list lookup based on parameters when there is no type information
keygen-sh_keygen-api\search.feature	Admin attempts to perform a search by license type on the key attribute for another account
scality_Zenko\azureArchive.feature	Restore an already restored object
keygen-sh_keygen-api\artifacts.feature	License retrieves the artifact for a release of their product (key auth, expired after release, revoke access)
keygen-sh_keygen-api\validations.feature	Anonymous validates a valid license key scoped to an array of majority invalid components (matching strategy MATCH_MOST)
opencypher_openCypher\Match6.feature	[2] Return a simple path
keygen-sh_keygen-api\generate.feature	Read-only generates a new token via basic authentication

opencypher_openCypher\Boolean3.feature	[2] Exclusive disjunction of three truth values
opencypher_openCypher\TriadicSelection1.feature	[10] Handling triadic friend of a friend that is not a friend with implicit superset of labels
keygen-sh_keygen-api\package.feature	Product retrieves the package for a release
opencypher_openCypher\Call1.feature	[1] Standalone call to procedure that takes no arguments and yields no results
keygen-sh_keygen-api\permits.feature	User renews their license that implements a protected policy
opencypher_openCypher\Merge4.feature	[1] Merge should be able to set labels on match and on create
keygen-sh_keygen-api\constraints.feature	Admin attaches constraints to a release when the constraint already exists
opencypher_openCypher\TypeConversion2.feature	[6] `toInteger()` on a complex-typed expression
opencypher_openCypher\Create4.feature	[2] Many CREATE clauses
keygen-sh_keygen-api\uses.feature	Product resets the usage count for a license
opencypher_openCypher\Precedence2.feature	[1] Numeric multiplicative operations takes precedence over numeric additive operations
opencypher_openCypher\Quantifier6.feature	[2] Single quantifier can nest itself and other quantifiers on the same list
opencypher_openCypher\List3.feature	[2] Equality of lists of different length should return false despite nulls
opencypher_openCypher\Literals8.feature	[16] Return 40-deep nested maps
keygen-sh_keygen-api\subscription.feature	Admin attempts to pause their paused account
keygen-sh_keygen-api\machines.feature	User attempts to retrieve their machines

APÊNDICE C - FORMULÁRIO DE VALIDAÇÃO DA PROPOSTA DE SMELLS

Formulário 1 - Validação da proposta de smells

Identificação de Smells em Testes de Software Escritos no Padrão BDD Utilizando a Linguagem Gherkin e o Framework Cucumber

---

Formulário de Consentimento Informado para a Pessoa Responsável

Caro(a) participante,

Esta pesquisa objetiva a obtenção de respostas de profissionais de testes de software sobre sua compreensão de *smells* em testes escritos utilizando o padrão BDD (*Behavior-Driven Development* ou Desenvolvimento Orientado a Comportamento).

***Test smells* indicam problemas potenciais no design e implementação de testes de software. Um *test smell* não significa necessariamente um problema já existente, mas um ponto de atenção no código de teste que, se não for abordado, pode aumentar para dificuldades adicionais, como baixa manutenibilidade, falta de cobertura ou ações ou verificações não determinísticas.**

Este questionário é composto por 12 perguntas rápidas, com exemplos de smells encontrados e identificados em casos de teste escritos no padrão BDD. Se você decidir participar, o preenchimento das respostas levará de 10 a 15 minutos.

Instruções

1. Nos exemplos de código você irá se deparar com as seguintes letras: F (Feature), S (Scenario), P (Precondition), A (Action) e V (Verification).
2. Nos exemplos de código, os trechos marcados com uma cor de fundo vermelha indicam onde estão os *smells*.

Gostaríamos de enfatizar que:

1. Sua participação é inteiramente voluntária e anônima;
2. Você pode desistir da participação a qualquer momento;
3. Sua recusa não prejudicará seu relacionamento com o pesquisador ou a instituição;
4. Não haverá divulgação de dados coletados de forma que permita sua identificação;
5. Sua resposta nos ajudará a interpretar como os profissionais de teste de *software* entendem essa questão.

Pesquisadores responsáveis:

- Felipe Santos (Graduando em Sistemas de Informação - IFAL) - fss30@aluno.ifal.edu.br
- Robson Gominho (Graduando em Sistemas de Informação - IFAL) - rag2@aluno.ifal.edu.br
- Tarsis Marinho (Docente - IFAL) - tarsis.souza@ifal.edu.br
- Elvys Soares (Docente - IFAL) - elvys.soares@ifal.edu.br

Se necessário, sinta-se à vontade para entrar em contato.

Entendo os objetivos, riscos e benefícios de participar da pesquisa.

☐ Eu concordo em participar.

☐ Eu não concordo em participar.

Ocupação

Qual é sua principal área de atuação?

☐ Academia

☐ Indústria

Qual é a sua experiência com testes de software (considere o tempo de estudo)?

☐ 0 (menos de um)

☐ 1

☐ 2

- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ 8
- ☐ 9
- ☐ 10 (ou mais)

Qual é a sua ocupação?

- ☐ Analista de Teste de Software
- ☐ Engenheiro de Teste de Software
- ☐ Analista de Garantia de Qualidade
- ☐ Outra: _____

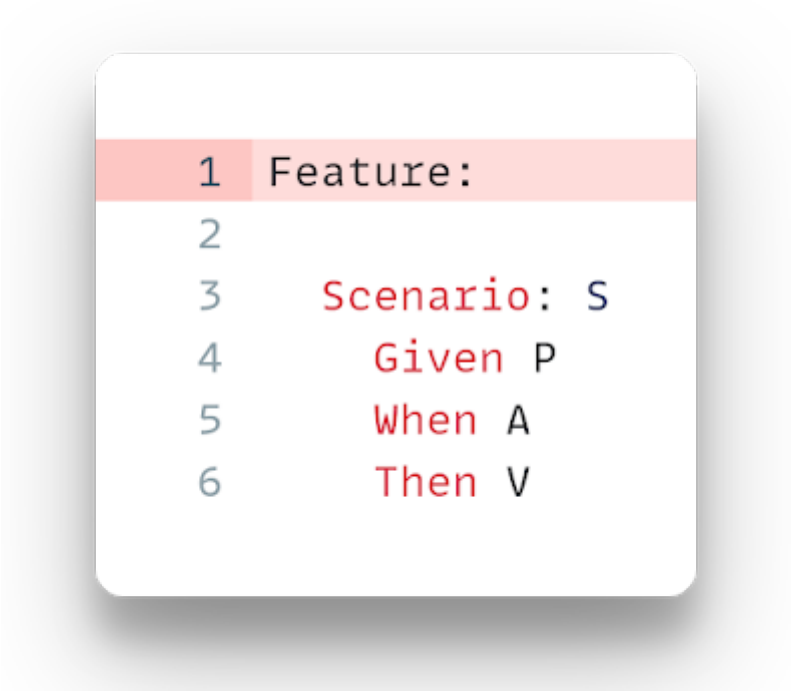
Questões

Questão 1

O smell "Untitled Feature" é caracterizado pela ausência de título em um arquivo .feature.

Consequências

- Ambiguidade: objetivo da funcionalidade fica pouco claro.
- Rastreabilidade reduzida: dificulta a associação entre cenários e requisitos.
- Desalinhamento e comunicação falha: risco de divergência entre intenção e implementação e prejuízo no entendimento entre equipes.



- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

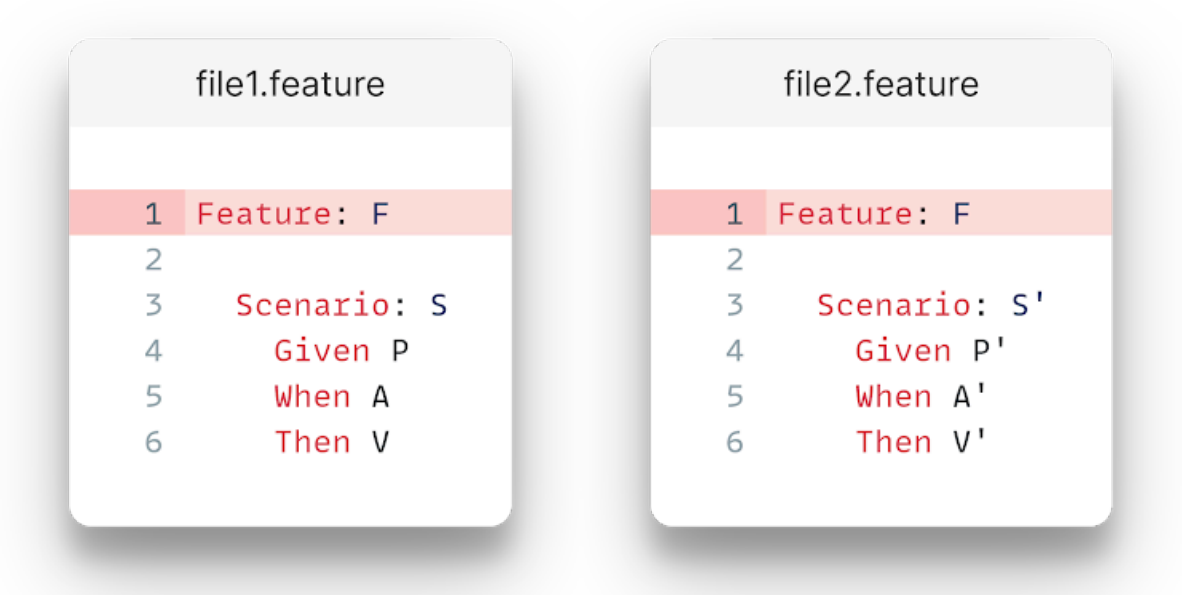
_____

Questão 2

O smell "Duplicate Feature Title" é caracterizado pela duplicação de um título de funcionalidade em arquivos .feature distintos.

Consequências

- Confusão de escopo: dificulta a distinção entre funcionalidades diferentes.
- Rastreabilidade prejudicada: complica a identificação de qual funcionalidade está sendo avaliada.
- Interpretação incorreta: aumenta o risco de erros ao analisar os resultados dos testes.



- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

Questão 3

O smell "Duplicate Scenario Title" é caracterizado pela duplicação de um título de cenário em um mesmo arquivo .feature ou em arquivos distintos.

Consequências

- Confusão nos resultados: dificulta entender qual cenário foi executado ou falhou.
- Inconsistência: dificulta entender qual é o comportamento esperado.
- Rastreabilidade comprometida: dificulta ligar cenários aos requisitos.

file1.feature

```
1  Feature: F
2
3    Scenario: S
4      Given P
5      When A
6      Then V
```

file2.feature

```
1  Feature: F'
2
3    Scenario: S
4      Given P'
5      When A'
6      Then V'
```

- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

Questão 4

O smell "Absence of Background" ocorre quando múltiplos cenários repetem as mesmas precondições dentro de um mesmo arquivo .feature e o recurso Background não é utilizado para agrupar essas precondições duplicadas.

Consequências

- Redundância de código: aumenta de forma desnecessária o tamanho do arquivo.
- Redução da clareza: passos repetidos desviam o foco da lógica de teste.
- Dificuldade de manutenção: mudanças nas precondições exigem atualizações em vários cenários.

```
1  Feature: F
2
3    Scenario: S
4      Given P
5      When A
6      Then V
7
8    Scenario: S'
9      Given P
10     When A'
11     Then V'
12
13   Scenario: S"
14     Given P
15     When A"
16     Then V"
```

- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

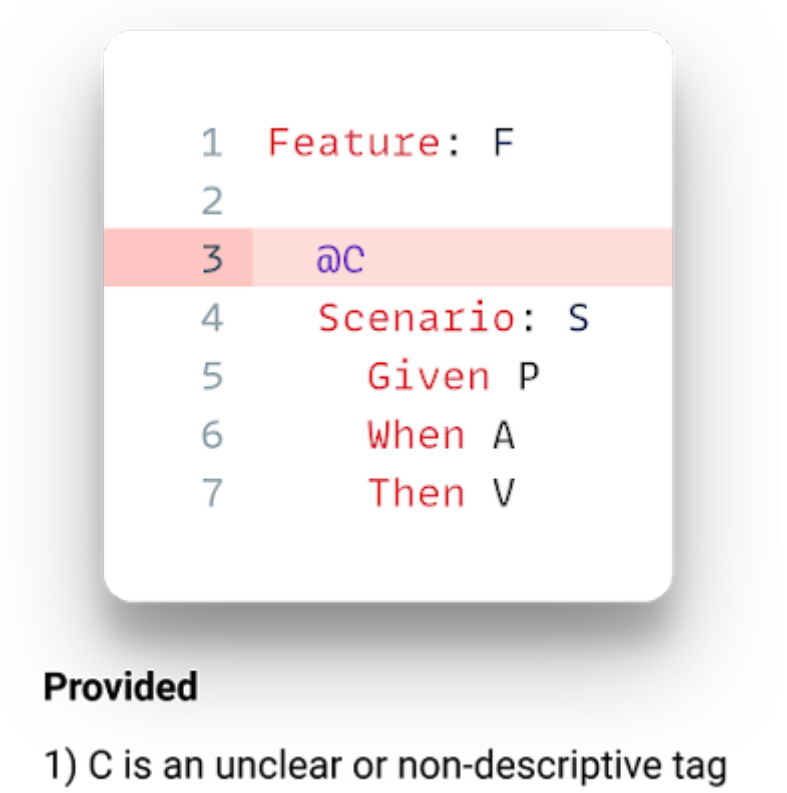
_____

Questão 5

O smell "Cryptic Tag" ocorre quando uma tag usada em um cenário ou funcionalidade não é clara ou não comunica seu propósito.

Consequências

- Ambiguidade: dificulta compreender a finalidade da tag.
- Manutenção complexa: torna difícil identificar onde e por que a tag é usada.
- Erros na organização: prejudica a categorização e execução dos testes.



- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

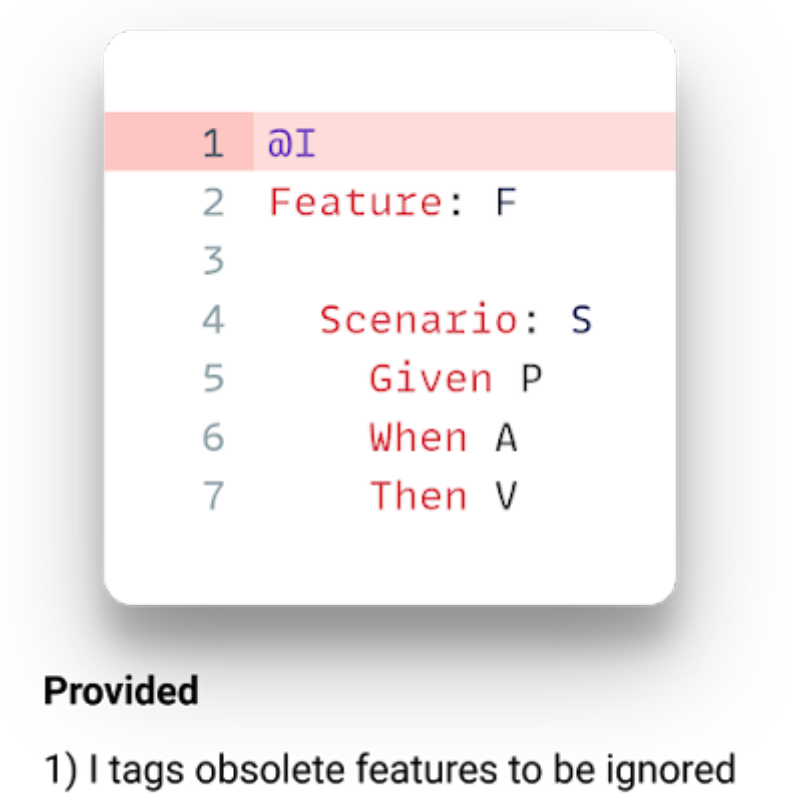
_____

Questão 6

O smell "Dead Feature" ocorre quando um arquivo .feature contém cenários desatualizados ou obsoletos, que são marcados com tags como @deprecated, mas permanecem no projeto sem utilidade.

Consequências

- Acúmulo de código obsoleto: torna o projeto mais pesado e difícil de manter.
- Confusão para os testadores: pode gerar dúvida sobre a relevância dos cenários ao executar os testes.
- Perda de foco: desvia a atenção de funcionalidades ativas e importantes.



- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

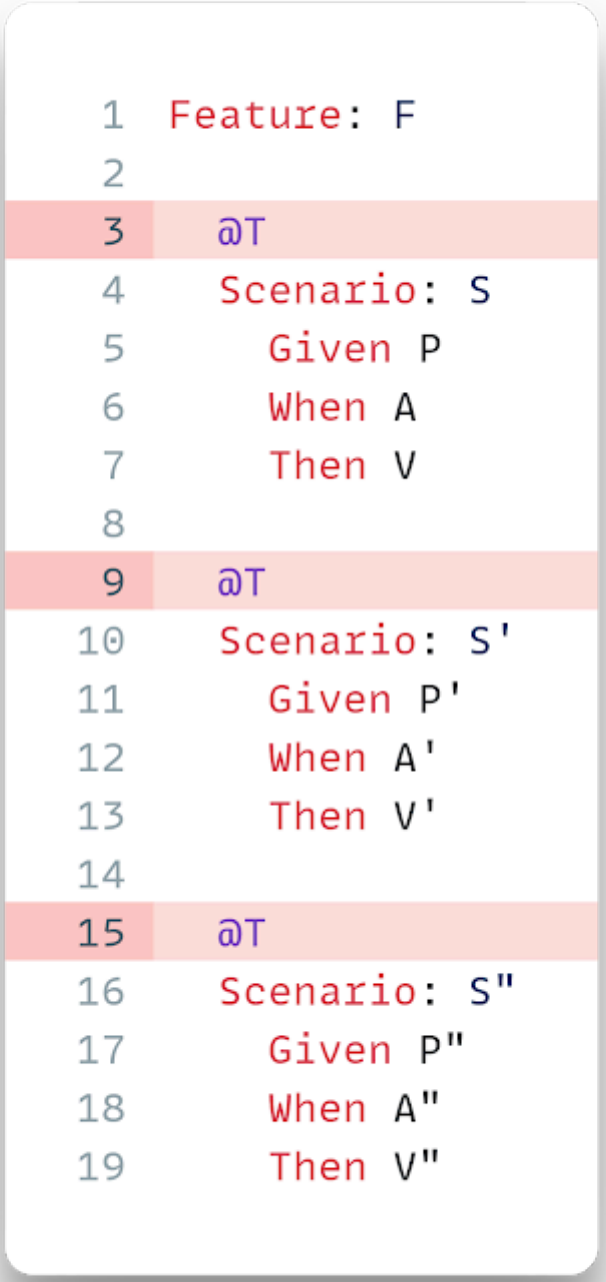
Poderia comentar sua resposta?

Questão 7

O smell "Vicious Tag" ocorre quando uma tag é repetida em todos os cenários de um arquivo .feature.

Consequências

- Redundância de código: aumenta o tamanho e reduz a legibilidade do arquivo.
- Manutenção trabalhosa: mudanças na tag exigem atualizações em todos os cenários.
- Propensão a erros: maior risco de inconsistências ao gerenciar tags manualmente.



```
1  Feature: F
2
3    @T
4    Scenario: S
5      Given P
6      When A
7      Then V
8
9    @T
10   Scenario: S'
11     Given P'
12     When A'
13     Then V'
14
15   @T
16   Scenario: S"
17     Given P"
18     When A"
19     Then V"
```

- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

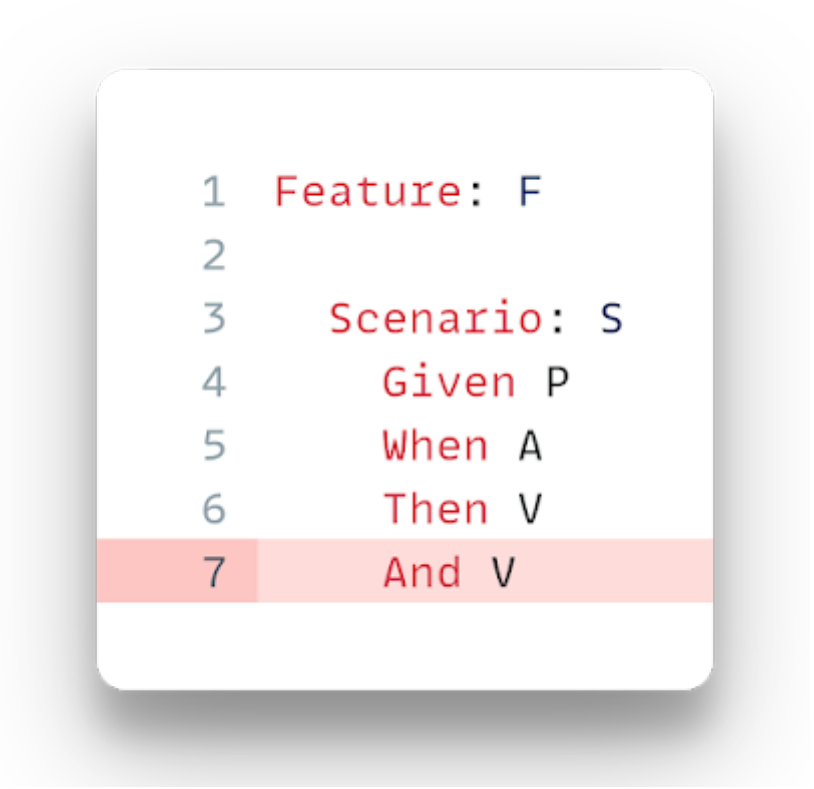
_____

Questão 8

O smell "Duplicate Step" ocorre quando um mesmo passo é duplicado dentro de um cenário.

Consequências

- Redundância de código: aumenta o tamanho.
- Execução duplicada: executa um mesmo passo múltiplas vezes de forma desnecessária.



- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

_____

Questão 9

O smell "Malformed Test" ocorre quando as palavras reservadas Given, When ou Then são repetidas indevidamente ao invés de usar And para continuidade, ou quando um cenário não inclui as palavras reservadas When ou Then essenciais para sua estrutura.

Consequências

- Quebra de legibilidade: dificulta entender a lógica do teste devido à inconsistência na estrutura.
- Perda de clareza no objetivo: cenários sem When ou Then ficam incompletos e ambíguos.

- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente

☐ Indiferente

☐ Não sei

Poderia comentar sua resposta?

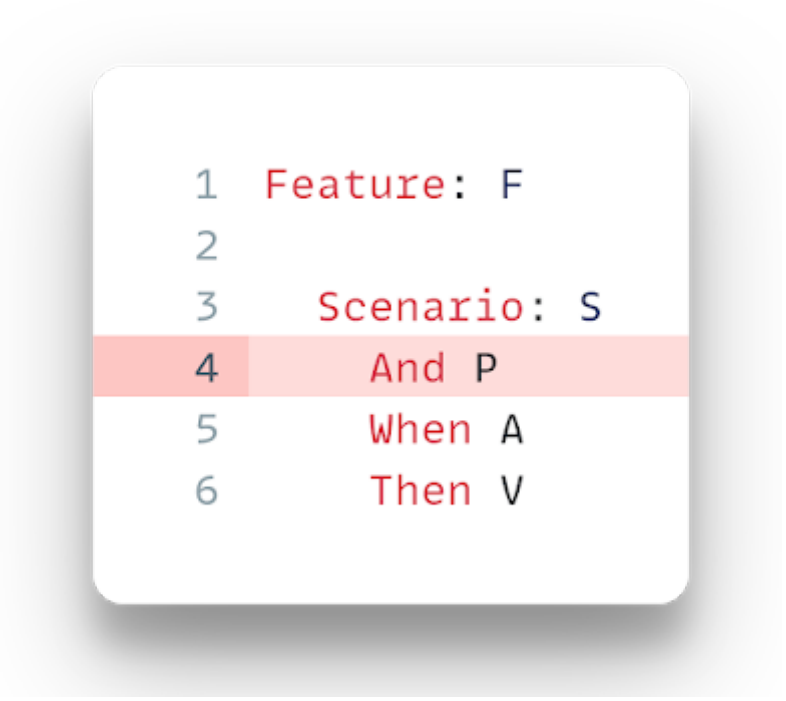
_____

Questão 10

O smell "Starting With The Left Foot" ocorre quando um cenário começa com And, But ou Then, e não com Given ou When.

Consequências

- Confusão no entendimento: dificulta identificar o contexto inicial do teste.
- Inconsistência estrutural: quebra as boas práticas definidas pela documentação do Cucumber.
- Diagnóstico trabalhoso: exige revisões adicionais do Background para entender o cenário.



☐ Concordo totalmente

☐ Concordo

☐ Discordo

☐ Discordo totalmente

☐ Indiferente

☐ Não sei

Poderia comentar sua resposta?

_____

Questão 11

O smell "Duplicate Test Case" ocorre quando o conteúdo de um cenário é idêntico a outro, seja no mesmo arquivo .feature ou em arquivos distintos.

Consequências

- Redundância desnecessária: aumenta o tamanho do arquivo sem agregar valor.
- Manutenção ineficiente: alterações precisam ser replicadas em cenários idênticos.
- Resultados confusos: relatórios de testes podem conter duplicatas, dificultando a análise.

file1.feature

```
1  Feature: F
2
3    Scenario: S
4      Given P
5      When A
6      Then V
```

file2.feature

```
1  Feature: F'
2
3    Scenario: S'
4      Given P
5      When A
6      Then V
```

- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

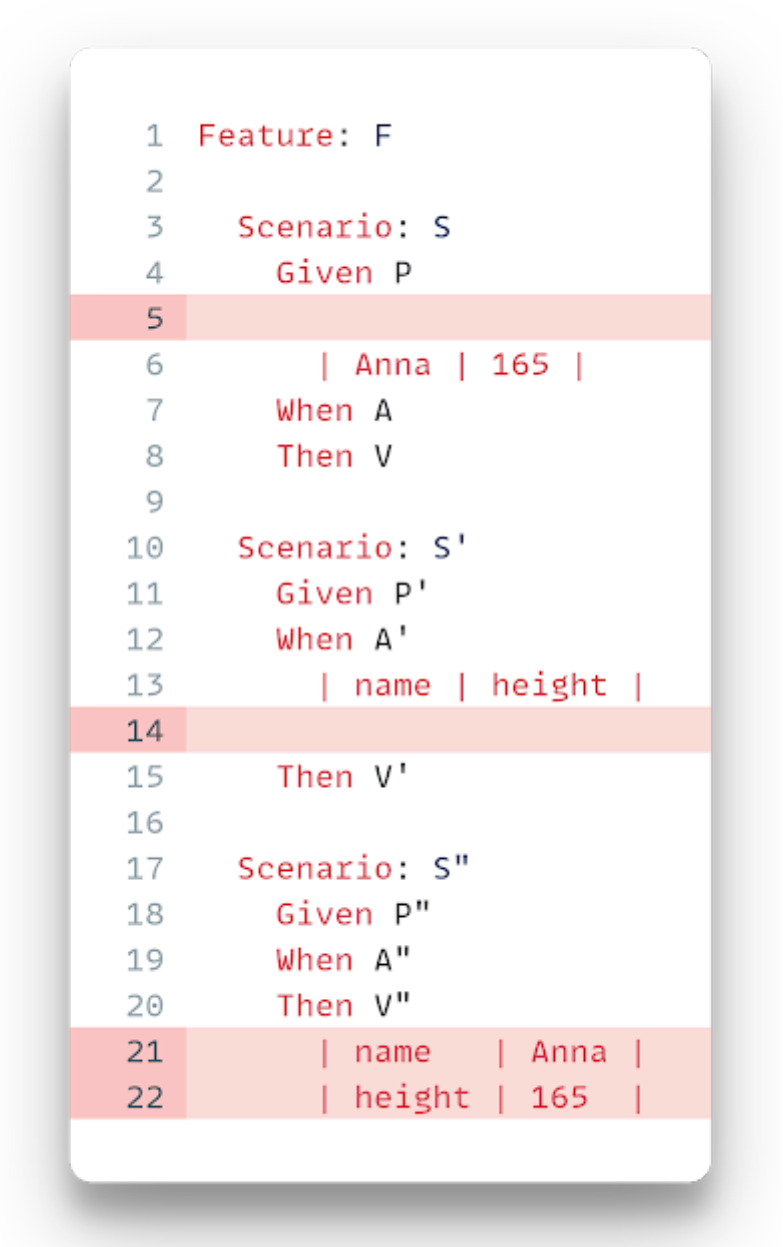
_____

Questão 12

O smell "Incomplete Table" ocorre quando uma tabela está incompleta, sem cabeçalho, sem corpo ou com valores mal formatados na horizontal.

Consequências

- Erros na automação: ferramentas podem processar incorretamente a tabela mal formatada.
- Confusão na interpretação: dificulta o entendimento da estrutura e dos dados da tabela.
- Perda de clareza: compromete a lógica do teste e a rastreabilidade das informações.



```
1  Feature: F
2
3    Scenario: S
4      Given P
5
6        | Anna | 165 |
7      When A
8      Then V
9
10   Scenario: S'
11     Given P'
12     When A'
13       | name | height |
14
15     Then V'
16
17   Scenario: S"
18     Given P"
19     When A"
20     Then V"
21       | name   | Anna |
22       | height | 165  |
```

- ☐ Concordo totalmente
- ☐ Concordo
- ☐ Discordo
- ☐ Discordo totalmente
- ☐ Indiferente
- ☐ Não sei

Poderia comentar sua resposta?

_____

APÊNDICE D - CENÁRIOS DEFINIDOS DA AMOSTRAGEM AUTOMATIZADA

Quadro 4 - Scenarios da amostragem automatizada

Feature	***Scenarios***
inukshuk_jekyll-scholar\186.feature	Simple bibliography count, Simple bibliography count with query, Simple bibliography count with query with cited
inukshuk_jekyll-scholar\string.feature	String replacement, String replacement disabled, Join strings, Join strings disabled, String replacement with queries involving negative conditions
keygen-sh_keygen-api\banned.feature	Banned user attempts to authenticate with a token, Banned user attempts to authenticate with a license key
keygen-sh_keygen-api\machines.feature	Endpoint should be inaccessible when account is disabled, Admin retrieves the machines for a user, Environment retrieves the machines for an isolated user, Product retrieves the machines for a user, Admin retrieves a machine for a user, Product retrieves a machine for a user, Product retrieves the machines of a user from another product, License attempts to retrieve the machines for another user, License attempts to retrieve their user's machines, User attempts to retrieve the machines for another user, User attempts to retrieve the machines for an associated user, User attempts to retrieve their machines, Admin attempts to retrieve the machines for a user of another account
keygen-sh_keygen-api\policies.feature	Endpoint should be inaccessible when account is disabled, Admin retrieves the policies for a product, Environment retrieves the policies for an isolated product, Environment retrieves the policies for a global product, Product retrieves the policies for a product, Admin retrieves a policy for a product,

	Product retrieves a policy for a product, Product retrieves the policies of another product, License attempts to retrieve the policies for their product, License attempts to retrieve the policies for a product, User attempts to retrieve the policies for their product, User attempts to retrieve the policies for a product, Admin attempts to retrieve the policies for a product of another account
keygen-sh_keygen-api\second_factors.feature	Endpoint should be inaccessible when account is disabled, Admin lists second factors while having 2FA disabled, Admin lists second factors while having 2FA enabled, Admin lists second factors while having no second factor, Admin lists a user's second factors and the user has 2FA disabled, Admin lists a user's second factors and the user has 2FA enabled, Admin lists a user's second factors and the user has no second factors, User lists second factors while having 2FA disabled, User lists second factors while having 2FA enabled, User lists second factors while having no second factor, User lists an admin's second factors, Environment lists an isolated admin's second factors, Environment lists an isolated user's second factors, Product lists an admin's second factors, Product lists a user's second factors, Admin retrieves a second factor while having 2FA disabled, Admin retrieves a second factor while having 2FA enabled, User retrieves a second factor while having 2FA disabled, User retrieves a second factor while having 2FA enabled, Environment retrieve a shared user's second factor, Product retrieve a user's second factor, Admin creates a second factor while having no other second factor and providing a correct password, User

	creates a second factor while having no other second factor and providing a correct password, Environment creates a second factor for an isolated user with no other second factor and provides a correct password, Product creates a second factor for a user with no other second factor and provides a correct password, Admin creates a second factor while having no other second factor and providing an incorrect password, Admin creates a second factor while having 2FA disabled and providing a correct password, Admin creates a second factor while having 2FA disabled and providing a correct OTP, Admin creates a second factor while having 2FA enabled and providing a correct OTP, Admin creates a second factor while having 2FA enabled and providing a correct password, Admin creates a second factor while having 2FA enabled and providing an incorrect password, Admin creates a second factor while having 2FA enabled and providing an incorrect OTP, Admin enables a second factor while having 2FA disabled and providing a correct OTP, Admin enables a second factor while having 2FA enabled and providing a correct OTP, Admin enables a second factor while having 2FA disabled and providing an incorrect OTP, Admin enables a second factor while having 2FA disabled and without an OTP, User enables a second factor while having 2FA disabled and providing a correct OTP, Environment attempts to enable a shared user's second factor, Product attempts to enable a user's second factor, Admin disables a second factor while having 2FA disabled and providing a correct OTP, Admin disables a second

	factor while having 2FA enabled and providing a correct OTP, Admin disables a second factor while having 2FA enabled and providing an incorrect OTP, Admin disables a second factor while having 2FA enabled and without an OTP, User disables a second factor while having 2FA enabled and providing a correct OTP, Environment attempts to disable a shared user's second factor, Product attempts to disable a user's second factor, Admin deletes a second factor while having 2FA enabled and providing a correct OTP, Admin deletes a second factor while having 2FA disabled, Admin deletes a second factor while having 2FA enabled and providing an incorrect OTP, User deletes a second factor while having 2FA enabled and providing a correct OTP, Environment attempts to delete a shared user's second factor, Product attempts to delete a user's second factor
keygen-sh_keygen-api\top-licenses-by-volume.feature	Endpoint should be inaccessible when account is disabled, Admin retrieves analytics of top licenses by volume for their account, Environment attempts to retrieve analytic counts for their account, Product attempts to retrieve analytic counts for their account, User attempts to retrieve analytics for their account
keygen-sh_keygen-api\user.feature	Endpoint should be inaccessible when account is disabled, Admin retrieves the user for a machine, Isolated environment retrieves the product for an isolated machine, Shared environment retrieves the product for a shared machine, Shared environment retrieves the product for a global machine, Product retrieves the user for a machine, Product retrieves the user for a machine of another product, User attempts to retrieve the user for a machine they own,

	User attempts to retrieve the user for a machine they have, User attempts to retrieve the user for a machine they don't own, License attempts to retrieve their user (default permission), License attempts to retrieve their user (has permission), License attempts to retrieve the user for a different license, Admin attempts to retrieve the user for a machine of another account
opencypher_openCypher\Aggregation5.feature	[1] `collect()` filtering nulls, [2] OPTIONAL MATCH and `collect()` on node property
opencypher_openCypher\Call6.feature	[1] Calling the same STRING procedure twice using the same outputs in each call, [2] Project procedure results between query scopes with WITH clause, [3] Project procedure results between query scopes with WITH clause and rename the projection
opencypher_openCypher\Conditional1.feature	[1] Run coalesce
opencypher_openCypher\Create2.feature	[1] Create two nodes and a single relationship in a single pattern, [2] Create two nodes and a single relationship in separate patterns, [3] Create two nodes and a single relationship in separate clauses, [4] Create two nodes and a single relationship in the reverse direction, [5] Create a single relationship between two existing nodes, [6] Create a single relationship between two existing nodes in the reverse direction, [7] Create a single node and a single self loop in a single pattern, [8] Create a single node and a single self loop in separate patterns, [9] Create a single node and a single self loop in separate clauses, [10] Create a single self loop on an existing node, [11] Create a single relationship and an end node on an existing starting

	node, [12] Create a single relationship and a starting node on an existing end node, [13] Create a single relationship with a property, [14] Create a single relationship with a property and return it, [15] Create a single relationship with two properties, [16] Create a single relationship with two properties and return them, [17] Create a single relationship with null properties should not return those properties, [18] Fail when creating a relationship without a type, [19] Fail when creating a relationship without a direction, [20] Fail when creating a relationship with two directions, [21] Fail when creating a relationship with more than one type, [22] Fail when creating a variable-length relationship, [23] Fail when creating a relationship that is already bound, [24] Fail when creating a relationship using undefined variable in pattern
opencypher_openCypher\Delete5.feature	[1] Delete node from a list, [2] Delete relationship from a list, [3] Delete nodes from a map, [4] Delete relationships from a map, [5] Detach delete nodes from nested map/list, [6] Delete relationships from nested map/list, [7] Delete paths from nested map/list, [8] Failing when using undefined variable in DELETE, [9] Failing when deleting an integer expression
opencypher_openCypher\Graph4.feature	[1] `type()`, [2] `type()` on two relationships, [3] `type()` on null relationship, [4] `type()` on mixed null and non-null relationships, [5] `type()` handling Any type, [6] `type()` failing on invalid arguments, [7] Failing when using `type()` on a node
opencypher_openCypher\List2.feature	[1] List slice, [2] List slice with implicit end, [3] List slice with implicit start, [4] List slice with singleton range, [5] List

	slice with empty range, [6] List slice with negative range, [7] List slice with invalid range, [8] List slice with exceeding range, [9] List slice with null range, [10] List slice with parameterised range, [11] List slice with parameterised invalid range
opencypher_openCypher\List9.feature	[1] Returning nested expressions based on list property
opencypher_openCypher\Match2.feature	[1] Match non-existent relationships returns empty, [2] Matching a relationship pattern using a label predicate on both sides, [3] Matching a self-loop with an undirected relationship pattern, [4] Matching a self-loop with a directed relationship pattern, [5] Match relationship with inline property value, [6] Match relationships with multiple types, [7] Matching twice with conflicting relationship types on same relationship, [8] Fail when using parameter as relationship predicate in MATCH, [9] Fail when a node has the same variable in a preceding MATCH, [10] Fail when a path has the same variable in a preceding MATCH, [11] Fail when a node has the same variable in the same pattern, [12] Fail when a path has the same variable in the same pattern, [13] Fail when matching a relationship variable bound to a value
opencypher_openCypher\Mathematical3.feature	[1] Fail for invalid Unicode hyphen in subtraction
opencypher_openCypher\Merge4.feature	[1] Merge should be able to set labels on match and on create, [2] Merge should be able to use properties of bound node in ON MATCH and ON CREATE
opencypher_openCypher\Null2.feature	[1] Property not null check on non-null node, [2] Property not null check on optional non-null node, [3] Property not null check on null node, [4] A literal null is not IS NOT null, [5] IS NOT NULL on

	a map, [6] IS NOT NULL is case insensitive
opencypher_openCypher\Pattern1.feature	[1] Matching on any single outgoing directed connection, [2] Matching on a single undirected connection, [3] Matching on any single incoming directed connection, [4] Matching on a specific type of single outgoing directed connection, [5] Matching on a specific type of single undirected connection, [6] Matching on a specific type of single incoming directed connection, [7] Matching on a specific type of a variable length outgoing directed connection, [8] Matching on a specific type of variable length undirected connection, [9] Matching on a specific type of variable length incoming directed connection, [10] Matching on a specific type of undirected connection with length 2, [10] Fail on introducing unbounded variables in pattern, [11] Fail on checking self pattern, [12] Matching two nodes on a single directed connection between them, [13] Fail on matching two nodes on a single undirected connection between them, [14] Matching two nodes on a specific type of single outgoing directed connection, [15] Matching two nodes on a specific type of single undirected connection, [16] Matching two nodes on a specific type of a variable length outgoing directed connection, [17] Matching two nodes on a specific type of variable length undirected connection, [18] Matching two nodes on a specific type of undirected connection with length 2, [19] Using a negated existential pattern predicate, [20] Using two existential pattern predicates in a conjunction, [21] Using two existential pattern predicates in a disjunction, [22] Fail on using

	pattern in RETURN projection, [23] Fail on using pattern in WITH projection, [24] Fail on using pattern in right-hand side of SET
opencypher_openCypher\Quantifier11.feature	[1] Any quantifier is always false if the predicate is statically false and the list is not empty, [2] Any quantifier is always true if the predicate is statically true and the list is not empty, [3] Any quantifier is always true if the single or the all quantifier is true, [4] Any quantifier is always equal the boolean negative of the none quantifier, [5] Any quantifier is always equal the boolean negative of the all quantifier on the boolean negative of the predicate, [6] Any quantifier is always equal whether the size of the list filtered with same the predicate is grater zero
opencypher_openCypher\Quantifier6.feature	[1] Single quantifier can nest itself and other quantifiers on nested lists, [2] Single quantifier can nest itself and other quantifiers on the same list, [3] Single quantifier is equal whether the size of the list filtered with same the predicate is one
opencypher_openCypher\Remove3.feature	[1] Limiting to zero results after removing a property from nodes affects the result set but not the side effects, [2] Skipping all results after removing a property from nodes affects the result set but not the side effects, [3] Skipping and limiting to a few results after removing a property from nodes affects the result set but not the side effects, [4] Skipping zero results and limiting to all results after removing a property from nodes does not affect the result set nor the side effects, [5] Filtering after removing a property from nodes affects the result set but not the side effects, [6] Aggregating in `RETURN` after removing a property

	from nodes affects the result set but not the side effects, [7] Aggregating in `WITH` after removing a property from nodes affects the result set but not the side effects, [8] Limiting to zero results after removing a label from nodes affects the result set but not the side effects, [9] Skipping all results after removing a label from nodes affects the result set but not the side effects, [10] Skipping and limiting to a few results after removing a label from nodes affects the result set but not the side effects, [11] Skipping zero result and limiting to all results after removing a label from nodes does not affect the result set nor the side effects, [12] Filtering after removing a label from nodes affects the result set but not the side effects, [13] Aggregating in `RETURN` after removing a label from nodes affects the result set but not the side effects, [14] Aggregating in `WITH` after removing a label from nodes affects the result set but not the side effects, [15] Limiting to zero results after removing a property from relationships affects the result set but not the side effects, [16] Skipping all results after removing a property from relationships affects the result set but not the side effects, [17] Skipping and limiting to a few results after removing a property from relationships affects the result set but not the side effects, [18] Skipping zero result and limiting to all results after removing a property from relationships does not affect the result set nor the side effects, [19] Filtering after removing a property from relationships affects the result set but not the side effects, [20] Aggregating in `RETURN` after removing a property from relationships affects the result set but not the side

	effects, [21] Aggregating in `WITH` after removing a property from relationships affects the result set but not the side effects
opencypher_openCypher\ReturnOrderBy2.feature	[1] ORDER BY should return results in ascending order, [2] ORDER BY DESC should return results in descending order, [3] Sort on aggregated function, [4] Support sort and distinct, [5] Support ordering by a property after being distinct-ified, [6] Count star should count everything in scope, [7] Ordering with aggregation, [8] Returning all variables with ordering, [9] Using aliased DISTINCT expression in ORDER BY, [10] Returned columns do not change from using ORDER BY, [11] Aggregates ordered by arithmetics, [12] Aggregation of named paths, [13] Fail when sorting on variable removed by DISTINCT, [14] Fail on aggregation in ORDER BY after RETURN
opencypher_openCypher\Set3.feature	[1] Add a single label to a node with no label, [2] Adding multiple labels to a node with no label, [3] Add a single label to a node with an existing label, [4] Adding multiple labels to a node with an existing label, [5] Ignore whitespace before colon 1, [6] Ignore whitespace before colon 2, [7] Ignore whitespace before colon 3, [8] Ignore null when setting label
opencypher_openCypher\String10.feature	[1] Finding exact matches with non-proper substring, [2] Finding substring of string, [3] Finding the empty substring, [4] Finding strings containing whitespace, [5] Finding strings containing newline, [6] No string contains null, [7] No string does not contain null, [8] Handling non-string operands for CONTAINS, [9] NOT with CONTAINS

opencypher_openCypher\Temporal3.feature	[1] Should select date, [2] Should select local time, [3] Should select time, [4] Should select date into local date time, [5] Should select time into local date time, [6] Should select date and time into local date time, [7] Should select datetime into local date time, [8] Should select date into date time, [9] Should select time into date time, [10] Should select date and time into date time, [11] Should datetime into date time
opencypher_openCypher\Temporal9.feature	[1] Should truncate date, [2] Should truncate datetime, [3] Should truncate localdatetime, [4] Should truncate localtime, [5] Should truncate time
opencypher_openCypher\TriadicSelection1.feature	[1] Handling triadic friend of a friend, [2] Handling triadic friend of a friend that is not a friend, [3] Handling triadic friend of a friend that is not a friend with different relationship type, [4] Handling triadic friend of a friend that is not a friend with superset of relationship type, [5] Handling triadic friend of a friend that is not a friend with implicit subset of relationship type, [6] Handling triadic friend of a friend that is not a friend with explicit subset of relationship type, [7] Handling triadic friend of a friend that is not a friend with same labels, [8] Handling triadic friend of a friend that is not a friend with different labels, [9] Handling triadic friend of a friend that is not a friend with implicit subset of labels, [10] Handling triadic friend of a friend that is not a friend with implicit superset of labels, [11] Handling triadic friend of a friend that is a friend, [12] Handling triadic friend of a friend that is a friend with different relationship type, [13] Handling triadic friend of a friend that is a friend with superset of relationship type, [14] Handling triadic friend of a friend that is a friend with implicit subset

	of relationship type, [15] Handling triadic friend of a friend that is a friend with explicit subset of relationship type, [16] Handling triadic friend of a friend that is a friend with same labels, [17] Handling triadic friend of a friend that is a friend with different labels, [18] Handling triadic friend of a friend that is a friend with implicit subset of labels, [19] Handling triadic friend of a friend that is a friend with implicit superset of labels
opencypher_openCypher\Union3.feature	[1] Failing when mixing UNION and UNION ALL, [2] Failing when mixing UNION ALL and UNION
opencypher_openCypher\WithWhere3.feature	[1] Join between node identities, [2] Join between node properties of disconnected nodes, [3] Join between node properties of adjacent nodes
scality_Zenko\backbeatServiceUser.feature	Backbeat Service Users are authorized to perform the actions and get success response, Backbeat Service Users are authorized to perform the actions and get expected error response, Backbeat Service Users are authorized to perform the actions, Backbeat Service Users are not authorized to perform the actions
sdkman_sdkman-cli\install_sdkman.feature	Creates and initialises .bash_profile on absence of login shell dot files, Add Init Snippet to the .bash_profile if present, Add Init Snippet to the .profile if present, Creates and initialises .bashrc on absence of non-login dot files, Always adds Init Snippet to the .bashrc, Creates and initialises .zshrc on absence of the file, Always adds Init Snippet to the .zshrc, Source the Initialisation Script on first invokation of the Init Snippet, Do not Source the Initialisation Script on subsequent invocation of the Init Snippet, Upgrade an installation without configuration

sdkman_sdkman-cli\upgrade_candidate.feature	Display upgradable candidate version in use when it is upgradable, Display upgradable candidate version in use when it is not upgradable, Display upgradable candidate version when none is in use, Display upgradable candidate versions when none is specified and none is in use, Display upgradable candidate versions when none is specified and one is in use, Display upgradable candidate versions when none is specified and multiple are in use, Display upgradable candidate versions when none specified and multiple in use but not upgradable, Update all upgradable candidates versions and set them as default, Don't update all upgradable candidates versions and set them as default, Update upgradable candidate version and set it as default, Update upgradable candidate version and auto-answer to make it default, Don't update upgradable candidate version and set it as default
serverlessworkflow_specification\do.feature	Task With Sequential Sub Tasks
serverlessworkflow_specification\raise.feature	Raise task with inline error
thoughtbot_factory_bot_rails\fixture_replacement_config.feature	Using Factory Bot and Factory Bot Rails with Test Unit generates a factory file and does not generate a fixture file, Using Factory Bot and Factory Bot Rails with RSpec should generate a factory file, Using Factory Bot and Factory Bot Rails with RSpec and suffix configuration should generate a factory file with suffix, Using Factory Bot and Factory Bot Rails does not override a manually-configured factories directory using RSpec, Using Factory Bot and Factory Bot Rails does not override a manually-configured factories directory using Test::Unit, Using

	Factory Bot Rails with MiniTest should generate a factory file, Using Factory Bot Rails with MiniTest and a custom directory should generate a factory file, Disable Factory Bot generator, Use a suffix with the Factory Bot generator, Use a filename_proc with the Factory Bot generator