UNIVERSIDADE FEDERAL FLUMINENSE

CAMILA PARDO GARCIA MORELLI

APLICANDO O SIGNIFYING API PARA INVESTIGAR A TESTABILIDADE DE APIS

NITERÓI

2022

CAMILA PARDO GARCIA MORELLI

APLICANDO O SIGNIFYING API PARA INVESTIGAR A TESTABILIDADE DE APIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação.

Orientadora:
Vânia de Oliveira Neves

Co-orientadora:
Luciana Cardoso de Castro Salgado

NITERÓI

2022

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

M842a Morelli, Camila Pardo Garcia
APLICANDO O SIGNIFYING API PARA INVESTIGAR A TESTABILIDADE DE APIS / Camila Pardo Garcia Morelli. - 2022.
122 f.

Orientador: Vânia de Oliveira Neves.
Coorientador: Luciana Cardoso de Castro Salgado.
Dissertação (mestrado)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2022.

1. Teste (Computação). 2. Semiótica. 3. Interface (Computador). 4. Significação (Psicologia). 5. Produção intelectual. I. Neves, Vânia de Oliveira, orientadora. II. Salgado, Luciana Cardoso de Castro, coorientadora. III. Universidade Federal Fluminense. Instituto de Computação.IV. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

Camila Pardo Garcia Morelli

APLICANDO O SIGNIFYING API PARA INVESTIGAR A TESTABILIDADE DE APIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação.

Aprovada em dezembro de 2022.

BANCA EXAMINADORA

Prof$^{\underline{a}}$. Dr$^{\underline{a}}$. Vânia de Oliveira Neves - Orientadora - UFF

Prof$^{\underline{a}}$. Dr$^{\underline{a}}$. Luciana Salgado - Coorientadora - UFF

Prof$^{\underline{a}}$. Dr$^{\underline{a}}$. Daniela Gorski Trevisan - UFF

Prof$^{\underline{a}}$. Dr$^{\underline{a}}$. Tayana Conte - UFAM

Dr. Luiz Marques Afonso - 3ELOS

Niterói

2022

"Queremos ter certezas e não dúvidas, resultados e não experiências, mas nem mesmo percebemos que as certezas só podem surgir através das dúvidas e os resultados somente através das experiências." Carl Jung

Agradecimentos

Agradeço a Deus por sempre estar presente em minha vida, iluminando-a e me dando forças para vencer os obstáculos.

Agradeço à minha família por todo o apoio e por me proporcionar condições para chegar até aqui.

Agradeço ao Vinícius Henriques por toda compreensão, apoio e paciência ao longo de todo o curso.

Agradeço à minha orientadora, Vânia de Oliveira Neves, por me apoiar desde o começo, desde a busca por um tema que eu me identificasse. Agradeço por toda a paciência e por nossos encontros semanais. Agradeço pelo apoio durante a pesquisa, especialmente durante a pandemia.

Agradeço à minha coorientadora, Luciana Cardoso de Castro Salgado, por me apoiar e me ajudar a dar rumo à minha pesquisa. Agradeço pelos encontros com o grupo SERG da UFF, que fizeram uma grande diferença, principalmente pelo networking e pela troca de experiências.

Agradeço aos meus professores e colegas de graduação, Ausberto S. Castro Vera e ao Luís Antônio Rivera, que me apoiaram e incentivaram a ingressar no mestrado.

Agradeço à Bruna e ao Marcel, que apareceram em um momento especial e contribuírem para a realização da pesquisa.

Agradeço a todos os voluntários que participaram do experimento, pois sem vocês essa pesquisa não teria sido possível.

Por fim, deixo meus agradecimentos à UFF, ao Instituto de Computação e a CAPES.

Sou muito grata a cada um dos envolvidos nessa conquista. O aprendizado que obtive nesses anos de pesquisa foi, sem dúvida, uma experiência muito valiosa para mim. Obrigada!

Resumo

Diante da diversidade de sistemas de informações presentes atualmente, a comunicação entre serviços exige que APIs sejam bem projetadas e compreendidas, tanto pelo produtor como pelo consumidor do serviço. APIs mal documentadas geram falhas na compreensão não só pela equipe de desenvolvimento como também pela equipe de testadores, que acabam projetando casos de testes ineficazes. Em consequência, são produzidos softwares com baixa qualidade e com defeitos que poderiam ter sido evitados. Esses defeitos podem estar relacionados com a dificuldade que o testador possa ter ao projetar melhores casos de testes, ocasionado por uma baixa testabilidade do sistema. Nesse sentido, este estudo investiga o uso do *SigniFYIng APIs*, uma ferramenta da Engenharia Semiótica, para apoiar a testabilidade das documentações das aplicações consumidoras de APIs. Para isso, foi elaborada uma proposta que consiste em inserir o *SigniFYIng APIs* no processo de teste de software. Essa proposta foi validada através da realização de um experimento considerando APIs de dois sistemas baseados na web, o Trello e o Moodle. O experimento foi conduzido com dez participantes voluntários que possuíam conhecimentos ou atuavam na área de teste de software. Os participantes foram divididos em dois grupos, o controle e o *SigniFYIng APIs*. O experimento possibilitou a identificação de pontos favoráveis e desfavoráveis para a utilização da proposta. Os pontos desfavoráveis relacionam-se principalmente à dificuldade dos participantes em relação ao aprendizado e entendimento da ferramenta e a sobrecarga gerada pela sua inclusão no processo de teste de software. Os resultados favoráveis mostram que o conhecimento adquirido da documentação da API, a partir do uso do *SigniFYIng APIs*, possibilitou o testador: elaborar casos de teste a partir das falhas de comunicabilidade, identificar as principais melhorias a serem realizadas nas APIs testadas e escrever casos de testes mais concisos e melhor projetados, conforme a documentação.

Palavras-chave: APIs, Testabilidade, Comunicabilidade, Teste de software, SigniFYIng APIs

Abstract

Given the diversity of information systems today, communication between services requires that APIs be well-designed and understood by both service producers and consumers. Poorly documented APIs lead to misunderstandings between developers and testers teams who end up designing ineffective test cases. As a result, they may produce software with low quality and avoidable errors. These defects can be related to the difficulty that the tester may have in designing better test cases caused by the low testability of the system. In this sense, this study investigates the use of *SigniFYIng APIs* a Semiotic Engineering tool, to support the testability of documentation of API consumer applications. For this, we elaborated a proposal that consists in applying *SigniFYIng APIs* in the software testing process. This proposal was validated by an experiment considering APIs from two web-based systems, Trello and Moodle. We separated ten volunteers into two groups, control, and *SigniFYIng APIs*. All of them knew or worked in software testing. The experiment allowed us to identify favorable and unfavorable points for using the proposal. The negative points are mainly related to the participant's difficulties in learning and understanding the tool and the overload generated by adding it to the software testing process. The favorable results show that the knowledge acquired from the API documentation that comes from the use of *SigniFYIng APIs* enables us to: develop test cases based on communicability failures, identify the main improvements to be made in the tested APIs, write more concise and better-designed test cases, following the documentation.

Keywords: APIs, Testability, Communicability, Software Test, SigniFYIng APIs

Lista de Figuras

1 Espaço design da Engenharia Semiótica adaptado (DE SOUZA, 2005). . . 17

2 Visão geral das etapas do *SigniFYIng APIs*, adaptada (SOUZA et al., 2010). 20

3 Frame de metacomunicação. 21

4 Cenário mostrando uma típica atividade de teste, adaptado (DELAMARO; JINO; MALDONADO, 2021), ... 23

5 Cenário mostrando os principais fatores que impactam na testabilidade, adaptado (BINDER, 1994). 26

6 Cenário da suíte de teste, adaptado (BINDER, 1994). 26

7 Etapas executadas do mapeamento sistemático. 30

8 Estudo da testabilidade em *API* com o *SigniFYIng APIs*, elaborada pela autora. 41

9 Aplicação de teste. 46

10 *Tabela verdade* construída para o cenário 1. 49

11 Casos de testes relacionados ao cenário 1. 49

12 Modelo de caso de teste para o grupo controle. 52

13 Modelo de caso de teste para o grupo SAPI. 53

14 Correlação dos casos de teste. 56

15 Categorização dos *feedbacks*. 56

16 Tempo total do grupo controle. 81

17 Tempo total do grupo SAPI. 81

18 Quantidade de casos de teste mal projetados. 87

19 Tempo gasto para elaboração de caso de teste. 88

20 Quantidade de *bugs* encontrados por participante. 89

21 Quantidade de *bugs* encontrados. 90

22 Tabela *verdade* construída para o cenário 2 109

23 Casos de testes relacionados ao cenário 2 109

24 Tabela *verdade* construída para o cenário 3 110

25 Casos de testes relacionados ao cenário 3 110

26 Tabela *verdade* construída para o cenário 4 111

27 Casos de testes relacionados ao cenário 4 111

28 Tabela *verdade* construída para o cenário 5 112

29 Casos de testes relacionados ao cenário 5 112

Lista de Tabelas

1 Dimensões cognitivas (BLACKWELL et al., 2001). 21

2 Efeitos da metacomunicação adaptada (DE SOUZA et al., 2016). 22

3 Termos pesquisados. 28

4 Palavras-chave e sinônimos. 28

5 Quantidade de artigos selecionados em cada iteração. 30

6 Artigos selecionados. 31

7 Artigos selecionados e as questões de pesquisa encontradas em cada estudo. 32

8 Estrutura do GQM. 43

9 Divisão dos Grupos. 58

10 Perfil dos participantes. 59

11 CDNs distintas. 62

12 Etiquetas distintas. 62

13 Situações observadas 72

14 Consolidação dos dados - pt.1. 73

15 Consolidação dos dados - pt.2. 74

16 Trechos de depoimentos sobre a *API* do Moodle. 75

17 Trechos de depoimentos sobre a *API* do Trello. 75

18 Trechos de depoimentos referentes às dificuldades percebidas. 78

19 Trechos de depoimentos referentes a sobrecarga. 79

20 Diferença do tempo total gasto entre o grupo controle e o SAPI. 82

21 Quantidade de iterações realizadas pelo grupo SAPI. 82

22 Trechos de depoimentos que auxiliaram nos testes. 84

23 Trechos de depoimentos que contribuíram nos testes. 85

24 Etiquetas de comunicabilidade, parte 1. 106

25 Etiquetas de comunicabilidade, parte 2. 107

26 Etiquetas de comunicabilidade, parte 3. 108

Sumário

1 Introdução **12**

1.1 Motivação 13

1.2 Objetivos 14

1.3 Organização 14

2 Fundamentação Teórica **15**

2.1 *APIs* e qualidade em *APIs* 15

2.2 Engenharia Semiótica e comunicabilidade 16

2.3 Ferramenta *SigniFYIng APIs* 18

2.4 Teste de software 22

2.4.1 Conceitos básicos 22

2.4.2 Técnicas de teste funcional 23

2.4.3 Testabilidade 25

2.5 Considerações do capítulo 26

3 Trabalhos Relacionados **27**

3.1 Metodologia de pesquisa 27

3.1.1 Objetivos e questões de pesquisa 27

3.1.2 Processo de seleção dos artigos 29

3.2 Dados coletados 31

3.3 Análise dos resultados 31

3.3.1 Questão de pesquisa 1: Quais abordagens foram utilizadas para garantir e medir a testabilidade? 32

3.3.2 Questão de pesquisa 2: Quais estudos de caso foram realizados para avaliar/explorar o conceito de testabilidade em *APIs*? 34

3.3.3 Questão de pesquisa 3: Quais são os desafios na testabilidade em *APIs*? 36

3.4 Ameaças à validade 38

3.5 Considerações do capítulo 39

4 Aplicando *SigniFYIng APIs* para investigar a testabilidade 40

4.1 Estudo - aplicando o *SigniFYIng APIs* 40

4.2 Metodologia 42

4.2.1 *Questões de pesquisa* 42

4.2.2 Definição do objeto de estudo 44

4.2.2.1 A aplicação de teste 45

4.2.2.2 Descrição dos cenários 45

4.2.2.3 Mapeamento dos casos de teste 48

4.2.3 Estudo piloto 50

4.2.4 Seleção de participantes 50

4.2.5 Preparação dos participantes 51

4.2.6 Execução do experimento 51

4.2.7 Análise de dados 55

4.2.7.1 Correlação dos casos de teste 55

4.2.7.2 Categorização dos *feedbacks* 56

4.3 Execução do experimento 57

4.3.1 Estudo piloto 57

4.3.2 Seleção dos participantes 58

4.3.3 Preparação dos participantes 59

4.3.4 Execução do experimento 59

4.4 Considerações do capítulo 59

5 Resultados e Discussões **61**

5.1 Questões de pesquisa 61

5.1.1 Análise da Q1: O que foi identificado com a aplicação da ferramenta *SigniFYIng APIs*? 61

5.1.1.1 Análise do cenário 1 63

5.1.1.2 Análise do cenário 2 64

5.1.1.3 Análise do cenário 3 66

5.1.1.4 Análise do cenário 4 66

5.1.1.5 Análise do cenário 5 67

5.1.1.6 Análise da aplicação do *SigniFYIng APIs* nos cenários . . 68

5.1.1.7 Análise dos comentários do grupo SAPI 69

5.1.1.8 Análise dos comentários do grupo controle 75

5.1.2 Análise da Q2: De que forma *SigniFYIng APIs* sobrecarrega e de que forma dificulta o processo tradicional de teste? 77

5.1.3 Análise da Q3: De que forma o *SigniFYIng APIs* auxiliou no processo de teste? E como *SigniFYIng APIs* contribui para melhorar a testabilidade da *API* testada? 83

5.1.4 Análise da Q4: De que forma o *SigniFYIng APIs* obteve melhores resultados? 87

5.2 Discussão dos Resultados 91

5.3 Limitações do *SigniFYIng APIs* 95

5.4 Limitações da aplicação do experimento 96

5.5 Ameaças à validade 97

5.6 Considerações do capítulo 98

6 Conclusões **99**

REFERÊNCIAS 102

Apêndice A – Etiquetas de comunicabilidade 106

Apêndice B – Mapeamento dos casos de testes 109

Apêndice C – Termo de Consentimento Livre e Esclarecido (TCLE) 113

Apêndice D – Carta Convite 116

1 Introdução

O aumento das expectativas no desenvolvimento de software em diversas esferas da sociedade, como em serviços de sistemas e na criação de cidades inteligentes, bem como o aumento da complexidade desses sistemas exige cada vez mais que eles consigam se comunicar entre si. Geralmente, essa comunicação se dá por meio de componentes reutilizáveis chamados de serviços (EILERTSEN; BAGGE, 2018). Além disso, essa complexidade muitas vezes é causada pela diversidade de sistemas de informação e pela quantidade de interações entre eles. Sendo assim, esses sistemas que podem ter sido desenvolvidos por diferentes organizações necessitam possuir alta interoperabilidade, ou seja, necessitam que haja uma comunicação transparente e eficiente.

As integrações de serviços são realizadas via *APIs Interface de Programação de Aplicativos*[1]. A utilização de *APIs* possibilita que uma equipe de software utilize ou consuma um serviço sem se preocupar com a implementação do mesmo. Essa abstração facilita o processo de desenvolvimento de software diante da complexidade e competitividade do mercado.

A utilização e o desenvolvimento de *APIs* se tornaram práticas necessárias, uma vez que *APIs* são peças fundamentais para a integração entre diferentes sistemas. Em uma análise realizada em vários projetos no GitHub mostrou que 93,3% dos projetos utilizam *APIs* (THUNG, 2016). Dada a ubiquidade desses mecanismos, é importante que haja soluções e métodos sólidos de desenvolvimento que garantam a qualidade do software que as consomem. No entanto, manter a qualidade de uma aplicação que utiliza um serviço torna-se uma tarefa não tão trivial. Para que a equipe de qualidade possa determinar se o sistema é adequado ou não, ela deve conferir questões como se os padrões de documentação foram seguidos durante o desenvolvimento, ou se o software foi testado adequadamente (SOMMERVILLE, 2010). Dessa forma, para a *API* ser efetiva, ela deve ser compreensível e utilizada da forma correta pela equipe de qualidade e, sendo assim, a qualidade da documentação se torna um fator crítico, já que se observa que o design e escrita de alta

[1] **Interface de Programação de Aplicativos:** do inglês, *"Application Programming Interface"*.

qualidade não são universalmente consistentes (WATSON et al., 2013).

1.1 Motivação

A utilização de *APIs* ainda representa um grande desafio de uso e aprendizado para programadores. Seu uso requer que o desenvolvedor/testador do software tenha conhecimento sobre o que a *API* irá executar mediante seu contrato, para assim poder estabelecer uma comunicação entre serviços eficazes com a ferramenta. Estudos revelam que, em média, 40 por cento das *APIs* testadas falham, seja em definição ou na implementação (ED-DOUIBI; IZQUIERDO; CABOT, 2018). Em definição descrita por Delamaro, Jino e Maldonado (2021), a falha é o resultado produzido pela execução no qual o resultado difere do esperado. Esse resultado é gerado por meio da existência de um defeito que pode ocasionar um erro durante uma execução de software, o que pode ocasionar um estado inconsistente ou inesperado do software (DELAMARO; JINO; MALDONADO, 2021).

Diante disso, a comunicação efetiva de uma *API* é um fator de grande importância durante o desenvolvimento de software, uma vez que as falhas de comunicação estão entre as causas de perdas de produtividade, ou até mesmo em falhas de projeto (LOPES; OLIVEIRA et al., 2019). Além da comunicação, outro fator relevante é a análise da testabilidade de software.

Projetar serviços altamente testáveis torna-se uma tarefa importante e desafiadora para os desenvolvedores de serviços. Da mesma forma que verificar e medir a testabilidade e a qualidade dos serviços também é importante para engenheiros de aplicação e usuários de serviços. A testabilidade, segundo IEEE (1990), é definida como: "O grau em que um sistema, ou componente, facilita o estabelecimento de critérios de teste e a execução deles para determinar se esses critérios foram atendidos [...]." Uma *API* com boa testabilidade é um indicador de qualidade de software, como visto por Tsai et al. (2006), que mostrou que a baixa testabilidade do componente não apenas sugere a baixa qualidade, mas também indica que o processo de teste do software é ineficaz. Semelhante aos requisitos e falhas de projeto, quanto mais tarde a baixa testabilidade for detectada durante o desenvolvimento do software, mais caro será para reparar.

1.2 Objetivos

Este trabalho tem como objetivo principal investigar se o testador, ao aplicar o *SigniFYIng APIs* ferramenta da Engenharia Semiótica (DE SOUZA, 2005), pode melhorar a testabilidade, ou seja, a capacidade de projetar melhores casos de teste de uma *API*.

O *SigniFYIng APIs* orienta a reflexão do analista durante a inspeção da *API* em busca de potenciais falhas de comunicabilidade. Com base nesse contexto surge a seguinte questão de pesquisa:

(i) Será que a identificação de problemas de comunicabilidade pode levar o testador a encontrar elementos para a melhoria dos projetos de casos de testes e a apoiar a testabilidade?

A investigação dessa proposta foi dividida da seguinte forma:

- definir uma abordagem para aplicação da ferramenta *SigniFYIng APIs* com a elaboração de casos de teste de software. Os testes de software aqui considerados são testes manuais e de caixa preta, ou seja, sem acesso ao código-fonte.
- validar a abordagem criada por meio de um experimento, no qual os participantes serão divididos em dois grupos, grupo controle e grupo *SigniFYIng APIs*. No qual o grupo *SigniFYIng APIs* irá aplicar a proposta que foi elaborada.

1.3 Organização

O restante deste trabalho está organizado da seguinte maneira: o Capítulo 2 trata dos conceitos e metodologias utilizados, como *APIs* e qualidade em *APIs*, Engenharia Semiótica e comunicabilidade, a ferramenta *SigniFYIng APIs*, teste de software, teste funcional e a testabilidade; no Capítulo 3 é exposto um mapeamento sistemático para encontrar trabalhos relacionados à testabilidade em *APIs*; o Capítulo 4 expõe os resultados obtidos, bem como uma análise dos mesmos; no Capítulo 5 encontram-se as discussões dos resultados do trabalho; o Capítulo 6 apresenta as conclusões deste trabalho.

2 Fundamentação Teórica

Neste capítulo são abordados os principais conceitos utilizados nessa pesquisa: APIs e qualidade de *APIs*, engenharia semiótica e comunicabilidade, assim como a ferramenta *SigniFYIng APIs*.

2.1 *APIs* e qualidade em *APIs*

Uma *API* é uma interface fornecida por terceiros que possibilita a equipe de software a utilizar e consumir um determinado tipo de serviço sem se preocupar com a implementação do mesmo. O uso de *APIs* envolve, geralmente, engenheiros de software que combinam seu próprio código com bibliotecas de software já existentes. Elas permitem que diferentes peças de software interajam entre si, reduzindo as dependências que existem entre elas (LOURENÇO MARCOS; PUCCINELLI DE OLIVEIRA, 2019). Projetar *APIs* utilizáveis é fundamental e um dos motivos mais convincentes para isso é que as *APIs* que são utilizáveis podem impulsionar a adoção e o uso sustentado de uma tecnologia específica (FAROOQ; ZIRKLER, 2010).

Dessa forma, uma *API* é uma representação das abstrações criadas por seus projetistas. Uma *API* precisa ser entendida pelos seus usuários (programadores, testadores) para ser usada de maneira eficaz. Assim, podemos considerar que o uso de uma *API* é como um processo de comunicação entre a pessoa que criou a *API* e a pessoa usuária da *API*, mediado pelos artefatos de software envolvidos (como especificações, documentação, código, binários, mensagens). Mais precisamente, esses artefatos comunicam aos usuários da *API* como eles devem interagir com a interface (AFONSO; CERQUEIRA; DE SOUZA, 2012).

Afonso, Cerqueira e de Souza (2012) comentam que ao utilizar uma *API* para construir uma aplicação, o programador precisa ter um bom conhecimento dos artefatos do software que estão sendo reutilizados, pois isso o leva a aplicar corretamente esses modelos em seu próprio *design*, de modo a chamar as operações e os serviços conforme a definição da *API*. Isso é importante, pois qualquer falta de entendimento sobre como usar a *API* pode

ocasionar falhas na aplicação consumidora.

A seção a seguir apresenta conceitos da Engenharia Semiótica e de comunicabilidade, conceitos nos quais estabelecem base para a construção e/ou análise de uma *API* bem projetada, ou seja, preocupada com a qualidade de comunicação do desenvolvedor que for utilizar a *API*.

2.2 Engenharia Semiótica e comunicabilidade

A Engenharia Semiótica, proposta originalmente por De Souza (2005), considera a interação humano-computador (IHC) como um caso especial de metacomunicação humana mediada por computador. Alguns conceitos-chave, que se relacionam com este trabalho, serão abordados: significação, comunicação entre os interlocutores e o espaço de design.

O processo de significação envolve signos e semiose. Em uma das definições existentes, um signo é algo (qualquer coisa) que representa alguma (outra) coisa para alguém Hoopes e Peirce (1991). O signo, segundo Hoopes e Peirce (1991), possui uma estrutura ternária: a representação, referência e seu significado. A semiose é o termo utilizado para designar o processo semiótico que consiste na produção de significados com a utilização de signos linguísticos e suas respectivas interpretações (ECO, 1976). A interface de uma aplicação computacional, segundo a teoria da Engenharia Semiótica, permite que o usuário interaja com ela por meio de ações. Essas ações dependem dos signos selecionados e comunicados pelo projetista (designer) por meio dessa interface. Esses signos transmitem como o usuário irá interagir com essa interface para atingir os seus objetivos.

A linguagem de interação é, portanto, constituída de signos definidos em tempo de design em um processo de significação, no qual certos signos são estabelecidos diante de convenções culturais e sociais. O usuário também pode ser um produtor ou um intérprete desses signos em tempo de interação.

Um exemplo de como um designer se relaciona com um usuário diante de um sistema, envolvendo a semiose, pode ser vista da seguinte forma De Souza (2005):

> "O sistema é a cristalização do estado final da semiose do projetista em relação às expectativas do usuário e o que a experiência do usuário deve ser. O usuário, ao utilizar o sistema, detecta esses significados no sistema e reage de acordo. Embora os usuários não possam identificar conscientemente o propósito do designer, a presença do designer no nível raiz de uma ontologia de IHC permite teorizar sobre como o

sistema se representa e alcança o objetivo."

Os interlocutores envolvidos no processo de comunicação em IHC são: os designers, os sistemas e os usuários. De um lado da comunicação temos o produtor da tecnologia, os designers e os desenvolvedores, enquanto do outro lado está quem consome essa tecnologia desenvolvida, os quais são denominados como consumidores, ou seja, os usuários. A comunicação é feita por meio de um sistema que possibilita a interação entre o consumidor e o produtor e ela representa o(s) projetistas do artefato computacional em tempo de interação. Outro elemento importante da ontologia da Engenharia Semiótica é o espaço de design que foi inspirado em elementos do modelo proposto originalmente pelo JAKOBSON (1960). O modelo adotado na Engenharia Semiótica é composta pelos seguintes elementos: o emissor (o designer), as mensagens (a metacomunicação de alto nível e as mensagens interativas de nível inferior), o código (a linguagem da interface), a tecnologia como canal (a interface), o receptor (o usuário) e o contexto (do usuário) no qual a interação ocorre (DE SOUZA, 2005), o que pode ser visto na Figura 1.

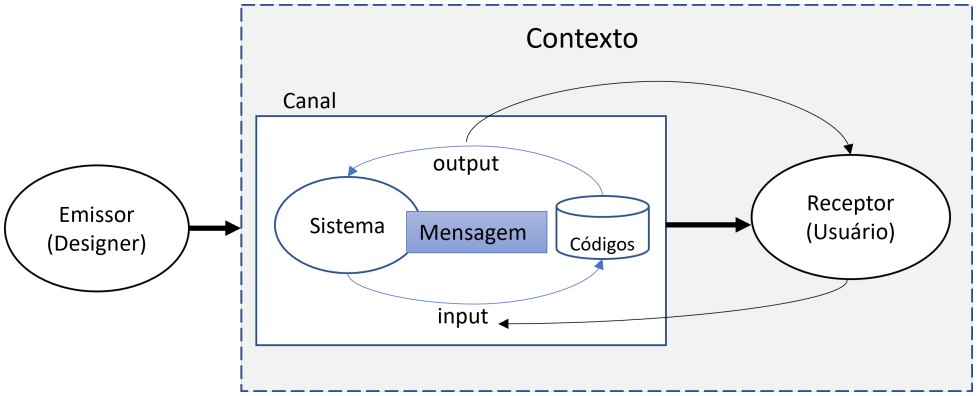

Figura 1: Espaço design da Engenharia Semiótica adaptado (DE SOUZA, 2005).

Na Engenharia Semiótica, "IHC é um tipo específico de metacomunicação de dois níveis, mediada por computador, na qual designers de sistemas computacionais enviam aos usuários uma mensagem unidirecional"(DE SOUZA, 2005), a mensagem que é transmitida para os usuários tem como intuito levar o usuário a se comunicar com o sistema de modo a atingir o seu objetivo. Ela é unidirecional, pois, do ponto de vista do designer, transmite um conteúdo completo e imutável, a qual é codificado e disponibilizado pela interface do sistema (DE SOUZA, 2005). Os dois níveis mostram como a metacomunicação ocorre. No primeiro nível cabe o emissor enviar uma mensagem para o receptor. Assim que essa mensagem chega para o receptor, por conta do segundo nível, o mesmo deve interagir com essa mensagem, ou seja, a comunicação só é de fato completamente efetuada se o

receptor se comunica com a mensagem que foi transmitida. Caso o receptor não interaja com a mensagem, a metacomunicação não é alcançada. Isso se deve porque o signo que representa a intenção do designer não é interpretado devidamente pelo usuário.

O conteúdo da mensagem de metacomunicação segue o seguinte modelo genérico De Souza (2005):

> "Eis a minha interpretação de quem você é, o que aprendi que você tem de fazer, preferencialmente de que forma e por quê. Eis, portanto, o sistema que consequentemente concebi para você, o qual você pode ou deve usar assim, a fim de realizar uma série de objetivos associados com esta minha visão.[1]"

Afonso (2015) afirma que comunicabilidade é um conceito chave na engenharia semiótica e tem evoluído ao longo dos anos com o refinamento da teoria. Souza et al. (2010) diz que a comunicação eficiente e eficaz é simplesmente a comunicação organizada e com recursos (eficiente), ou seja, que alcança o objetivo desejado (eficaz).

Durante o desenvolvimento de software a comunicação eficaz é um fator importante, pois se a comunicação não for clara e bem definida, pode gerar um software que não atende o que foi pedido pelo cliente. É justamente por meio de situações como essas que acontece a maioria das falhas e perdas em um projeto. Em Lopes, Oliveira et al. (2019), é explorada a comunicação no desenvolvimento de software utilizando modelos como comunicação indireta por meio de software, em que pode ser caracterizada como um processo de entendimento mútuo entre os produtores *"producers"* — quem concebe a modelagem do software, e os consumidores *"consumers"* — quem pega a informação dos modelos para desenvolver os artefatos.

2.3 Ferramenta *SigniFYIng APIs*

A Engenharia Semiótica, em 2016, estendeu sua perspectiva original da comunicação para uma perspectiva baseada na computação centrada no humano (DE SOUZA et al., 2016). Essa área de pesquisa visa entender o comportamento humano, integrando tecnologias em contextos sociais e culturais. Nesse contexto, foi definida a ferramenta denominada *SigniFYI (Signs for Your Interpretation)*, composta por um conjunto de ferramentas conceituais e metodológicas. O denominado *SigniFYI Suite* auxilia na investigação de

[1]Texto original: "Here is my understanding of who you are, what I have learned you want or need to do, in which preferred ways, and why. This is the system that I have, therefore, designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision."

significados em um software durante o seu processo de desenvolvimento, assim como auxilia, também, na comunicação entres os produtores e os consumidores de software (LOPES; CONTE; SOUZA, 2018).

A ferramenta *SigniFYI* apresenta um módulo chamado de *SigniFYIng APIs*, o qual propõe um método de inspeção para avaliar a comunicabilidade de pacotes de programação usados em atividades típicas de implementação de sistemas. Ela pode ser utilizada não apenas para analisar, mas também para projetar *API* (DE SOUZA et al., 2016).

Segundo Robillard e DeLine (2011 apud AFONSO; CERQUEIRA; DE SOUZA, 2012) são descritas descobertas qualitativas sobre as dificuldades em aprender *API*, nas quais se concluíram que a intenção da documentação é um dos fatores mais importantes que impactam na experiência de quem está aprendendo a utilizá-la. Como destacado pelos autores Afonso, Cerqueira e de Souza (2012), a documentação da intenção durante a construção de uma *API* enfatiza a importância de se comunicar efetivamente aos programadores as intenções do designer da *API* ao desenvolver, como suas abstrações e conceitos. Além disso, a responsabilidade de documentar uma *API* não pode ser separada da responsabilidade de projetar uma *API*.

O *SigniFYIng APIs* institui três fases para a análise da interface de uma *API* qualquer, como pode-se observar na Figura 2, adaptada de De Souza et al. (2016). As três fases compõem uma iteração da ferramenta, sendo possíveis várias iterações.

Na primeira fase denominada como *intenção*, a análise centra-se na intenção do projetista da *API*, na qual se inclui, por exemplo, uma definição da audiência da interface, por meio de um *frame* de metacomunicação. O *frame* de metacomunicação, como mostrado como exemplo da Figura 3, é composto por elementos como: a pessoa que envia a mensagem, a mensagem que é enviada e o usuário que recebe, assim como interpreta mediante ao modelo de aplicação. A ilocução de um projetista, como vista na Figura 3 corresponde ao efeito de comunicação pretendido pelo remetente, enquanto a perlocução de um usuário corresponde aos efeitos de comunicação realmente alcançados pela comunicação.

Na segunda fase denominada como *efeito*, a análise é centrada no vocabulário, na sintaxe e na semântica da *API*. O analista vislumbra os efeitos de se usar a *API*, ou seja, determina-se uma análise cognitiva das notações de programação denominadas como *CDNs Dimensão Cognitiva de Notação*[2], que podem ser vistas na Tabela 1. Nela se tem as dimensões cognitivas com suas respectivas descrições.

[2]**Dimensão Cognitiva de Notação:** do inglês, *"Cognitive Dimension of Notation"*.

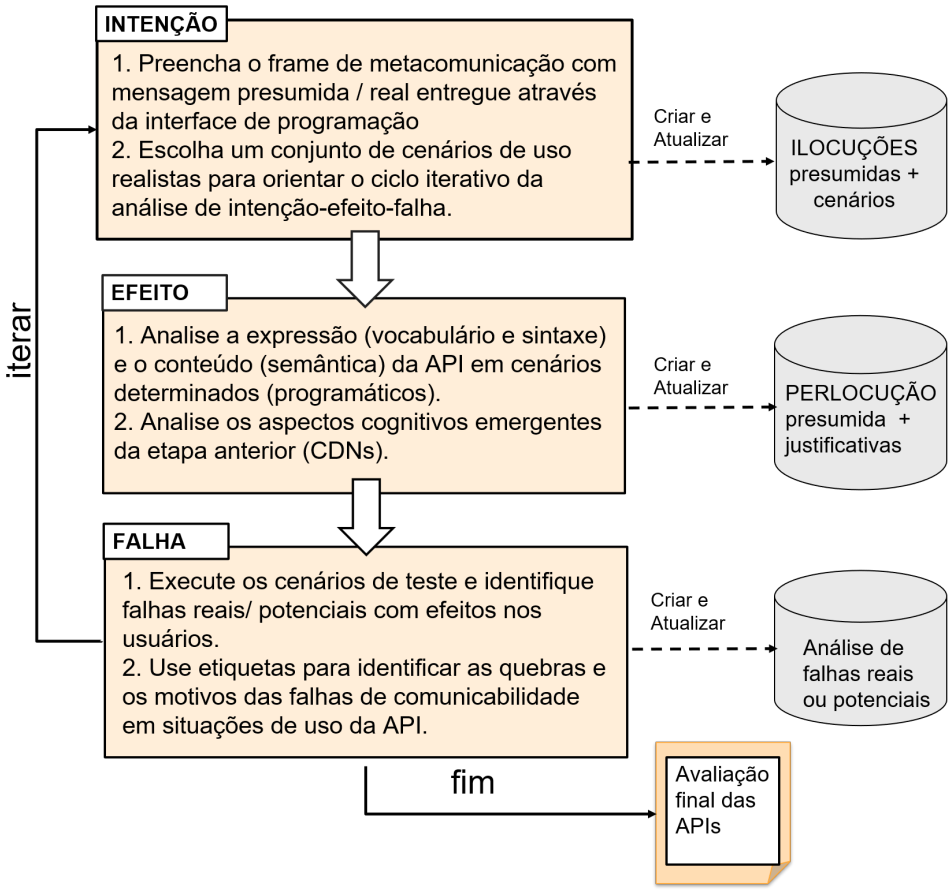

Figura 2: Visão geral das etapas do *SigniFYIng APIs*, adaptada (SOUZA et al., 2010).

Na terceira fase denominada como *falha*, ocorre a interação com a *API*. Testes concebidos na fase anterior e que agora são executados, em que sucessos e falhas são documentados com o uso de etiquetas de comunicabilidade (por exemplo, *Epa!*, *Vai de outro jeito*, *Desisto*, etc.), com o vocabulário provido pelo *SigniFYIng APIs* visando estabelecer uma linguagem comum. Por exemplo, se o cenário concebido na primeira fase for de impossível realização, usa-se a etiqueta *Falha consciente da tarefa* associada à etiqueta de comunicabilidade *Desisto*, assumindo neste exemplo que o analista tenha percebido a impossibilidade de se satisfazer o objetivo do cenário, portanto nada há a fazer exceto desistir. A lista completa das etiquetas de comunicabilidade podem ser vistas no Apêndice A, nas tabelas 24, 25 e 26.

Nessa etapa, também é necessário identificar e categorizar os diferentes tipos de problemas que podem ser encontrados. Fraquezas e inconsistências detectadas na comunicação da interface da *API* podem ser qualificadas conforme a Tabela 2 de efeitos da comunicação.

Os defeitos de comunicação podem ser agrupados em três categorias: falhas completas, falhas parciais e falhas temporárias. As falhas completas são as mais graves, pois impedem

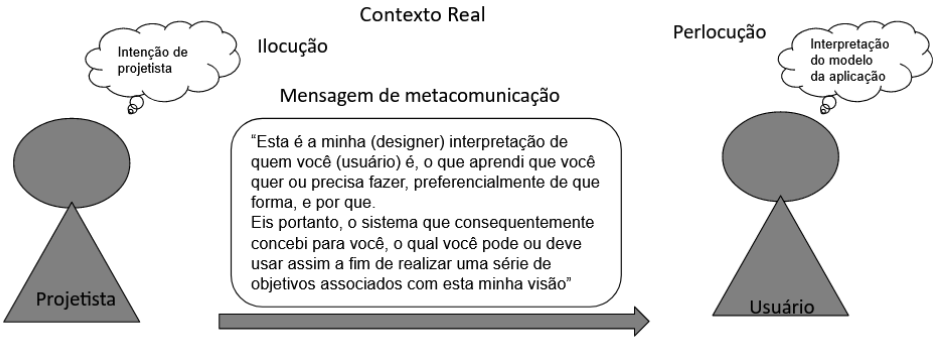

Figura 3: Frame de metacomunicação.

Dimensão Cognitiva	**Descrição**
Nível de Abstração (Abstraction Level)	Correspondência entre abstrações de API e expectativas do usuário ou interpretação
Proximidade do Mapeamento (Closeness of mapping)	Proximidade da representação ao domínio
Consistência (Consistency)	Semânticas semelhantes são expressas em formas sintáticas semelhantes
Difusão (Diffuseness)	O usuário precisa escrever mais código do que precisava ou queria contornar as limitações da API
Propensão a erros (Error-proneness)	A API convida a erros e oferece pouca proteção no contexto das estratégias erradas do usuário
Operações mentais difíceis (Hard mental operations)	Alta demanda de recursos cognitivos
Dependências ocultas (Hidden dependencies)	Links importantes entre entidades não são visíveis ou não são óbvios
Expressividade do papel (Role-expressiveness)	A finalidade de um elemento de API é prontamente inferida pelo usuário, sem a necessidade de procurar mais esclarecimentos e/ou desambiguação
Visibilidade (Visibility)	Capacidade de visualizar entidades facilmente

Tabela 1: Dimensões cognitivas (BLACKWELL et al., 2001).

o usuário de conseguir a informação desejada, ou de concluir a tarefa definida. Em falhas parciais, o usuário ainda pode cumprir o objetivo principal usando a aplicação — o defeito está em uma discrepância entre a forma de comunicação ocorrida e a esperada pelo usuário. Falhas temporárias, por outro lado, são as que o usuário pode se recuperar por conta própria. Os tipos de defeitos são associados a etiquetas de comunicabilidade, ver Apêndice A. Nele é apresentado as subcategorias das etiquetas de comunicabilidade nas tabelas 24,25 e 26, as quais se referem aos erros de expressão, conteúdo e intenção comunicada entre o usuário e o testador da *API*. Essas etiquetas de comunicabilidade servem para identificar qual tipo de erro foi encontrado durante as fases da análise.

Percepção do usuário e entendimento do recurso da API			**Efeito da metacomunicação**	**ID**
Usuário entende o recurso provido da API	Usuário compreendeu a API	usuário aceita o recurso da API	Sucesso	E1
		usuário rejeita o recurso da API	Recusa	E2
		recurso que a API prove é diferente das expectativas do usuários	Inesperado	E3
	Usuário não compreendeu completamente a API	pela operação da API não ser compatível com o modelo conceitual de resolução do problema do usuário	Incompreendido	E4
		por ter feito considerações erradas sobre o contexto de uso - o usuário esperava que a API faria algo e fez outra	Mal usado	E5
Usuário não entende o recurso provido da API	API provê recurso		Falha	E6
	API não provê recurso	usuário precisa aceitar o que a API prove	Esperado	E7
		usuário não aceita o recurso provido pela API	Sucesso	E8

Tabela 2: Efeitos da metacomunicação adaptada (DE SOUZA et al., 2016).

2.4 Teste de software

Nesta seção, são explicados conceitos de teste que são necessários para o entendimento deste trabalho, como: o que são casos de teste, as fases de um teste, conceitos de testabilidade e técnicas de teste, em específico teste funcional.

2.4.1 Conceitos básicos

Realizar teste é uma atividade complexa, pois diversos fatores podem colaborar com a ocorrência de um erro, como visto em (DELAMARO; JINO; MALDONADO, 2021). É importante destacar que o processo de teste inclui três etapas bem definidas: planejamento, elaboração de casos de teste e execução dos cenários.

Em um cenário de teste, como apresentado em Delamaro, Jino e Maldonado (2021), o domínio de entrada é um conjunto de dados de um programa, como visto na Figura 4. A atividade de teste consiste em elaborar um conjunto de casos de testes e executá-los em um programa determinado. A resposta que se obtém é avaliada por meio de um oráculo que pode ser, por exemplo, um testador, o qual define se a saída obtida de uma determinada execução coincide com a saída esperada, baseado na especificação do programa. Se o resultado obtido do caso de teste no programa é o que era esperado, então o teste teve sucesso. Se a resposta do teste for diferente do esperado, diz-se que o teste falhou, como pode ser visto na Figura 4.

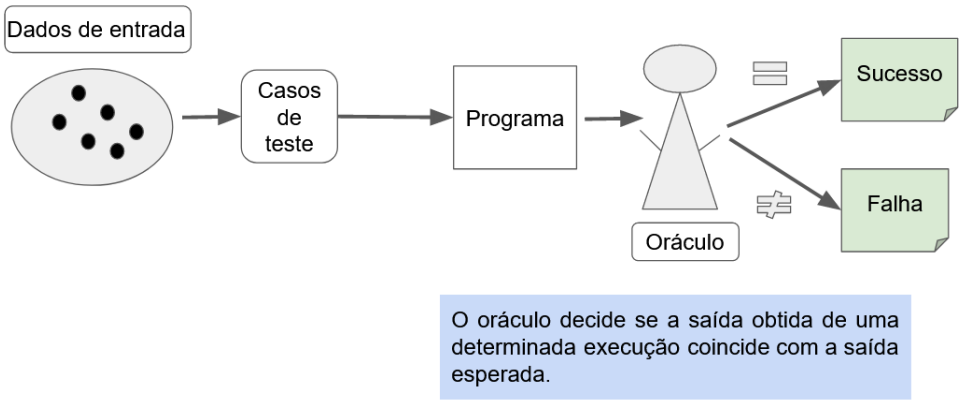

Figura 4: Cenário mostrando uma típica atividade de teste, adaptado (DELAMARO; JINO; MALDONADO, 2021), .

2.4.2 Técnicas de teste funcional

No cenário ideal, uma forma de se garantir que um programa está funcionando adequadamente seria testar todos os elementos do domínio de entrada, ou seja, fazer um teste exaustivo. Como isso não é viável, existem para isso os critérios de teste que consistem em selecionar elementos de cada um dos subdomínios de teste, seja por técnicas de modo aleatório ou por técnicas de teste de partição. Essencialmente nas técnicas de teste de partição, existem três tipos: funcional, estrutural e baseada em defeitos (DELAMARO; JINO; MALDONADO, 2021).

O teste funcional é uma técnica que considera o sistema a ser testado uma caixa preta, em que para testá-lo considera-se apenas a especificação do sistema a ser testado, independendo de detalhes de implementação. Nesse tipo de técnica, qualquer informação ausente ou incompleta na documentação torna difícil a aplicação do teste funcional (DELAMARO; JINO; MALDONADO, 2021).

A técnica do teste funcional considera critérios como o particionamento de equivalência, análise do valor limite, grafo causa-efeito. O critério de particionamento de equivalência, como visto em Delamaro, Jino e Maldonado (2021), considera a divisão do domínio de entrada em classes de equivalência, em que qualquer elemento de uma classe pode ser considerado o representante da mesma. Ou seja, se o elemento detecta um defeito, qualquer outro da classe também detecta.

A determinação de uma classe de equivalência representa um conjunto de estados válidos ou inválidos, dadas as condições de entrada (DELAMARO; JINO; MALDONADO, 2021).

> *Classe de Equivalência.* "Uma vez identificadas as classes de equivalência, devem-se determinar os casos de teste, escolhendo um elemento de cada classe, de forma que cada novo caso de teste cubra o maior número de classes válidas possíveis. Já para as classes inválidas, devem ser gerados casos de testes exclusivos, uma vez que um elemento de uma classe inválida pode mascarar a validação do elemento de outra classe inválida (DELAMARO; JINO; MALDONADO, 2021)."

Já a análise do valor limite completa a ideia do critério de particionamento em classes de equivalência, em que é estabelecido um intervalo de valores e são elaborados casos de teste para o limite, ultrapassando o limite estabelecido.

Tanto o critério do particionamento de equivalência como o critério do valor limite possibilitam a redução no tamanho do domínio de entrada e na criação de dados de teste baseados unicamente na especificação. O critério de valor limite é adequado quando as variáveis de entrada podem ser identificadas com facilidade e assumem valores específicos (DELAMARO; JINO; MALDONADO, 2021).

O grafo de causa-efeito, também conhecido de teste de tabela de decisão, combina os diferentes valores de entrada nos quais podem resultar em diferentes ações a serem tomadas. O critério consiste basicamente nas seguintes etapas (DELAMARO; JINO; MALDONADO, 2021):

1. Dividir a especificação em partes.
2. Identificar as causas e efeitos na especificação e atribuir número único para cada.
3. Gerar o Grafo Causa-Efeito.
4. Aprimorar o grafo, considerando combinações de causas e efeitos que são impossíveis.
5. Converter o grafo em uma tabela de decisão.
6. Converter as regras da tabela de decisão em casos de teste.

Esse critério tem como vantagem explorar a combinação dos dados de teste que possivelmente não seriam considerados, fazendo com que os resultados esperados produzidos façam parte da própria tabela de decisão.

2.4.3 Testabilidade

Teste de software é uma das principais técnicas para se garantir a qualidade, mas para que ela possa ser conduzida efetivamente, possibilitando a criação de bons casos de testes, uma das formas possíveis é por meio de uma boa documentação. O grau em que um testador consegue projetar casos de teste está relacionado ao conceito de testabilidade. Ou seja, segundo IEEE (1990), testabilidade é definida como:

> O grau em que um sistema, ou componente, facilita o estabelecimento de critérios de teste e a execução deles para determinar se esses critérios foram atendidos; e o grau em que um requisito é declarado, em termos que permitem o estabelecimento de critérios de teste e o desempenho deles para determinar se esses critérios foram atendidos.

De acordo com Tsai et al. (2006), quando um serviço apresenta uma boa testabilidade, a qualidade de serviço aumenta e consequentemente reduz o custo de teste. Como visto em Watson et al. (2013), a testabilidade não apresenta uma forma clara de definir quais aspectos do software estão realmente relacionados a ela, sugerindo a busca de meios e técnicas para ajudar a melhorar a testabilidade em um software.

Binder (1994) define o conceito de testabilidade por dois fatores: a controlabilidade e a observabilidade no teste de componente. O autor comenta que para testar um componente deve se ter o controle de suas entradas e observar as saídas geradas, pois se não for possível controlar a entrada, logo não se pode afirmar o que causou a saída obtida.

Existem alguns obstáculos para aplicar esses fatores: a controlabilidade e a observabilidade. Esses obstáculos se devem pelo fato de um componente em teste ser incorporado em outro sistema. Remover os obstáculos para controlar os dados de entrada e gerar saídas que possam ser observáveis é o desafio principal no *design* para testabilidade (BINDER, 1994).

Design for testability, como abordado por Binder (1994), é uma estratégia para alinhar o processo de desenvolvimento para tornar o teste mais efetivo. A Figura 5 mostra os seis principais fatores que impactam na testabilidade de um software. Entrando em detalhes na espinha *Suíte de teste*, como observado na Figura 6, é possível observar o conjunto de fatores que afetam em específico o processo de teste.

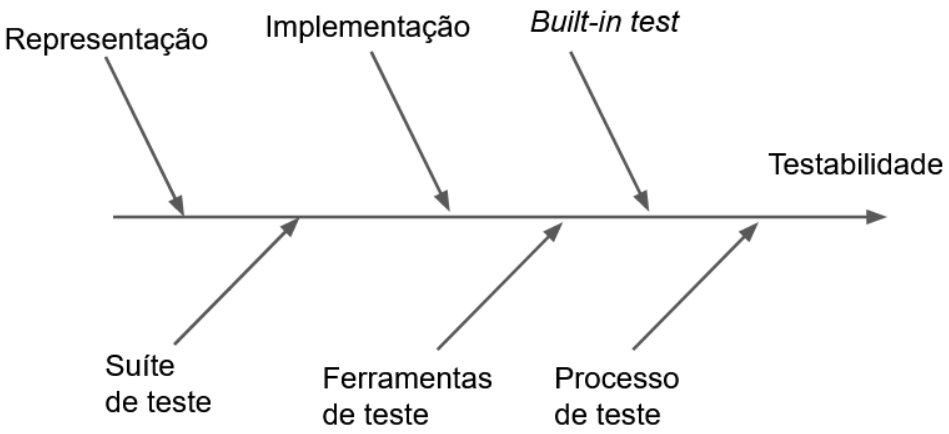

Figura 5: Cenário mostrando os principais fatores que impactam na testabilidade, adaptado (BINDER, 1994).

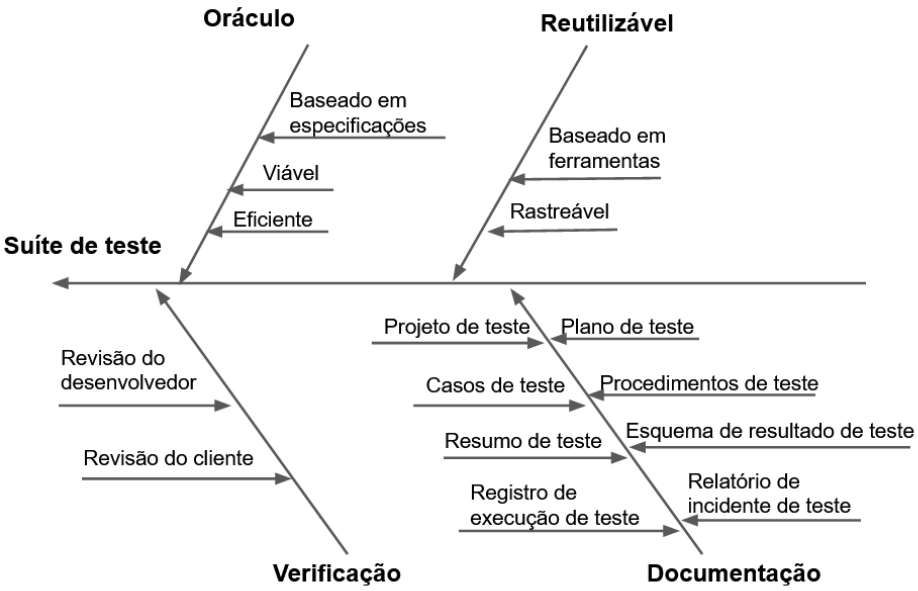

Figura 6: Cenário da suíte de teste, adaptado (BINDER, 1994).

2.5 Considerações do capítulo

Nesse capítulo, foram vistos os principais conceitos que serão utilizados no desenvolver da pesquisa envolvendo conceitos da Engenharia Semiótica e de testabilidade.

3 Trabalhos Relacionados

Neste capítulo, é abordado o mapeamento sistemático conduzido com o objetivo de verificar os trabalhos já existentes relacionados à testabilidade em *APIs*.

3.1 Metodologia de pesquisa

A realização do mapeamento sistemático foi conduzida durante o período de fevereiro a março de 2021. A condução da revisão se deu com o apoio da ferramenta Parsifal[1] e com a abordagem PICOC (KITCHENHAM; CHARTERS, 2007), a qual permitiu planejar o protocolo a ser utilizado na pesquisa.

3.1.1 Objetivos e questões de pesquisa

Na etapa inicial, foi necessário definir o objetivo da pesquisa que envolveu o seguinte questionamento: Como a testabilidade tem sido considerada em sistemas que utilizam *APIs*? Em seguida, foram levantadas um conjunto de questões para ajudar a responder a questão principal da pesquisa. Essas questões foram as seguintes:

- RQ1: Quais abordagens foram utilizadas para garantir e medir a testabilidade em *APIs*?
- RQ2: Quais estudos de caso foram realizados no contexto de testabilidade em *API*?
- RQ3: Quais são os desafios na testabilidade em *APIs*?

O próximo passo realizado foi a definição da *string* de busca. Para estruturar a *string* de busca, ela foi dividida em três etapas, seguindo a abordagem PICOC apresentada em Kitchenham e Charters (2007). Para aplicar essa metodologia é necessário, primeiramente,

[1] https://parsif.al/

estruturar as questões de busca em termos do PICOC composto por: população, intervenção, comparação, saídas e contexto. Essa estrutura permite identificar e encontrar as possíveis palavras-chave relacionadas à pesquisa. Por seguinte, estabeleceu-se os sinônimos para cada palavra-chave encontrada. E por último foi construída uma *string* baseada nas palavras-chave e com os sinônimos como descrito na Tabela 3.

População	("Application Programming Interface")
Intervenção	("testability")
Comparação	-
Resultados	("case study" OR "metrics" OR "evaluation" OR "problem" OR "models")
Contexto	("software testing")

Tabela 3: Termos pesquisados.

As palavras-chave e os sinônimos da pesquisa foram definidos, como podem ser vistos na Tabela 4. A escolha dos sinônimos *API, microservices, soa* e *web service* para a palavra-chave *Application Programming Interface* se deu pelo fato de a busca apenas por *API* ter retornado poucos resultados. Esses termos relacionados foram escolhidos, pois esses tipos de sistemas se comunicam por meio de *APIs*.

Palvras-chaves	**Sinônimos**	**Relacionado a**
Application Programming Interface	api microservices soa web services	População
case study	-	Resultado
evaluation	assessment measuring valuation	Resultado
metrics	-	Resultado
models	-	Resultado
problem	proposition	Resultado
testability	-	Intervenção

Tabela 4: Palavras-chave e sinônimos.

Definidas as palavras-chave e os sinônimos, Figura 4, foi elaborada a string de busca:

String de busca: ((SOA) OR (Service Oriented Architecture) OR (Web services) (webservice) OR (API) OR (Application Programming Interface) OR (rest) OR (microservices) OR) AND (Testability))

Identificados os termos que devem ser buscados, critérios foram criados, denominados aqui com a letra c, em que foram utilizados para realizar a seleção dos artigos. Seguem os critérios definidos:

C1 - Estudos que apresentam meios/métricas/técnicas de medir testabilidade em *APIs*.

C2 - Estudos que apresentam as dificuldades/desafios em garantir uma boa testabilidade em *APIs*.

C3 - Estudos abordando experimentos em testabilidade em *APIs*.

C4 - Estudos que falem de qualidade envolvendo testabilidade em *APIs*.

Os critérios de exclusão, denominados aqui com a letra E, foram utilizados como critério para excluir os artigos da seleção. Seguem os critérios definidos:

E1 - Estudos duplicados.

E2 - Estudos fora do tema TI/Computação/SIS.

E3 - Estudos não localizados de forma completa na biblioteca digital.

E4 - Estudos que não contenham métricas/meios/técnicas, ou não apresentem dificuldades/desafios, ou não abordem experimentos no cenário de testabilidade.

E5 - Estudos que não estão escritos em inglês ou português.

3.1.2 Processo de seleção dos artigos

O mapeamento foi realizado nas bases do IEEExplorer e no Scopus, adotando os critérios e a *string* de busca descritas na seção 3.1.1. A escolha dessas fontes se deu devido a uma busca prévia no Google Scholar sobre o tema que indicou que a maioria dos artigos relevantes estavam nessas bases. Conforme Mourão et al. (2020), quando aplicada uma estratégia híbrida que combine a biblioteca Scopus com a técnica *snowballing* (WOHLIN,

2014), se tem uma eficiente alternativa em processos de mapeamento da literatura. Diante a isso, foi realizada essa estratégia híbrida.

A seleção ocorreu ao longo da análise dos títulos e resumos dos estudos, realizada por uma dupla de pesquisadores especializados na área de teste de software para obter, assim, uma seleção mais criteriosa. Para avaliar os artigos encontrados, foram aplicados os critérios de inclusão para tornar-se um artigo selecionado, assim como foram aplicados os critérios de exclusão para rejeitar o artigo retornado na busca.

A Tabela 5 mostra a quantidade de artigos selecionados em cada iteração. Inicialmente, foi executada a *string* de busca na biblioteca IEEExplorer, a qual retornou o total de 53 artigos. Já na Scopus, a busca retornou 149 artigos. Ambos os resultados foram expressos como Iteração 1 na Tabela 5. Ao fim da Iteração 4, obteve-se 6 artigos selecionados, sendo 3 da IEEExplorer e 3 da Scopus.

Fonte de pesquisa	**Iteração 1**	**Iteração 2**	**Iteração 3**	**Iteração 4**
IEEExplore	53	21	3	3
Scopus	149	38	7	3

Tabela 5: Quantidade de artigos selecionados em cada iteração.

Além da busca realizada nas duas fontes de pesquisa, foi aplicada também a técnica de *snowballing*, como pode ser visto na Figura 7. Nela é mostrada de forma geral todas as etapas executadas para a realização do mapeamento. Após a aplicação da técnica do *snowballing* nos 6 artigos selecionados, pode-se observar que foram encontrados 63 artigos possíveis para o estudo. Após a avaliação realizada pela dupla de pesquisadores, encontrou-se 3 novos artigos. Ao final da leitura completa desses artigos, quatro foram descartados, totalizando 5 artigos relevantes para o mapeamento.

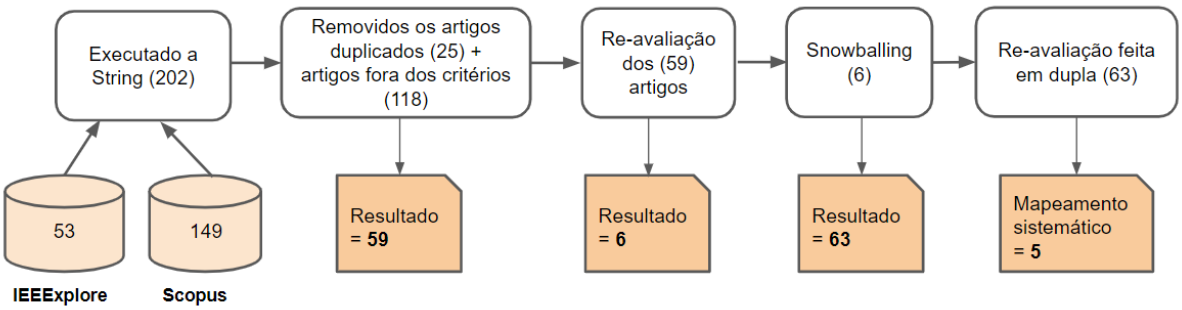

Figura 7: Etapas executadas do mapeamento sistemático.

3.2 Dados coletados

Os 5 artigos encontrados e selecionados foram publicados durante o período de 2006, 2007, 2009, 2013 e 2015. Com o conjunto final de artigos selecionados, a extração dos dados foi realizada nos 5 artigos. Os artigos selecionados foram os seguintes, como pode ser visto na Tabela 6:

IDs	Título	Autores	Referência
A1	A Model for the Measurement of the Runtime Testability of Component-based Systems	A. González É. Piel H. Gross	(GONZÁLEZ; PIEL; GROSS, 2009)
A2	Testability of Software in Service-Oriented Architecture	W. T. Tsai J. Gao X. Wei Y. Chen	(TSAI et al., 2006)
A3	A maintainability assessment model for service-oriented systems	Senivongse, Twittie Puapolthep, Assawin	(SENIVONGSE; PUAPOLTHEP, 2015)
A4	Applying Structural Testing to Services Using Testing Interfaces and Metadata	Eler, Marcelo Medeiros Bertolino, Antonia Masiero, Paulo Cesar	(ELER; BERTOLINO; MASIERO, 2013)
A5	Quality Properties for Service-oriented Architectures	L. O'Brien P. Merson L. Bass	(O'BRIEN; MERSON; BASS, 2007)

Tabela 6: Artigos selecionados.

3.3 Análise dos resultados

Os resultados foram analisados da seguinte forma: em cada estudo que foi selecionado, procurou-se responder as questões de pesquisa: RQ1, RQ2, e RQ3. Como não foi possível identificar essas questões de pesquisa em cada um dos estudos analisados, na Tabela 7, foram relacionados quais estudos apresentaram as questões que respondem à R1, R2 e R3.

Artigo Selecionado	R1	R2	R3
A model for the Measurement of the runtime testability of component-based systems	X	X	X
Testability of Software in Service-Oriented Architecture	X	X	
A maintainability assement model for service oriented systems	X	X	X
Applying Structural Testing of Services using Testing Interfaces and Metadata	X	X	
Quality properties for Service-oriented Architectures			X

Tabela 7: Artigos selecionados e as questões de pesquisa encontradas em cada estudo.

3.3.1 Questão de pesquisa 1: Quais abordagens foram utilizadas para garantir e medir a testabilidade?

Nos trabalhos encontrados: González, Piel e Gross (2009), Tsai et al. (2006), Senivongse e Puapolthep (2015) e Eler, Bertolino e Masiero (2013), obtiveram-se diferentes abordagens utilizadas para garantir e medir a testabilidade.

Em González, Piel e Gross (2009), os autores definem o que é testabilidade para testes que devem ser conduzidos em tempo de execução. Nessa definição, leva-se em consideração a sensitividade e o isolamento do teste. O primeiro caracteriza quais operações realizadas como parte de um teste interferem no estado do sistema em execução, ou de seu ambiente, de forma inaceitável. Assim, o isolamento do teste representa meios para evitar que operações de teste interfiram no estado, ou no ambiente, do sistema e que o estado produzido e as interações do sistema influenciem os resultados do teste. Portanto, o isolamento do teste representa a capacidade de um sistema de permitir o teste de tempo de execução, fornecendo contramedidas para sua sensibilidade de teste. Com isso, os autores também definem uma métrica de medição de testabilidade em tempo de execução, em que ela é calculada pelo número de funcionalidades que são capazes de serem testadas em tempo de execução e pelo número total de funcionalidades que devem ser testadas.

Em Tsai et al. (2006), o artigo estende os critérios de avaliação de testabilidade, adicionando mais quatro critérios referentes a especifidades de SOA. Esses novos critérios consideram (i) a atomicidade do serviço, (ii) a proveniência dos dados, (iii) integridade dos serviços e (iv) colaboração entre os serviços. Os autores também comentam que a testabilidade de SOA deve ser considerada em quatro níveis: (i) disponibilidade do

código-fonte, (ii) disponibilidade do código binário, (iii) disponibilidade do modelo e (iv) disponibilidade da assinatura.

Em Senivongse e Puapolthep (2015), os autores definem uma hierarquia de atributos de manutenibilidade de serviços e um conjunto de métricas para esses atributos. Entre essas métricas, eles consideram que a testabilidade é composta de dois atributos: foco no ambiente e processo de simulação. A primeira leva em conta que o teste deve considerar o serviço no ambiente de operação, sua QoS e como eles funcionam com o *middleware*, barramento do serviço e sistemas externos. A segunda considera que os testes devem considerar também o processo, ou seja, simular a orquestração de processos de negócios e entrada/saída ao longo do fluxo do processo. Levando isso em conta, os autores definem a métrica de testabilidade sendo de 0,5* foco no ambiente + 0,5* simulação do processo.

Em Eler, Bertolino e Masiero (2013), a fim de melhorar a falta de controle e observabilidade, os autores propõem a abordagem chamada de Serviço mais testável por teste de Metadados [2], definido pela sigla MTxTM. Essa abordagem tem o objetivo de tornar os serviços ainda mais testáveis fornecendo aos integradores de serviço (uma pessoa responsável por fazer a integração) informações sobre as atividades de testes conduzidas pelos desenvolvedores dos serviços testáveis. Os integradores podem usar essa informação para avaliar a cobertura alcançada dos serviços testáveis usados e para criar novos casos de testes para exercitar as instruções, caminhos e dados que não haviam sido executados.

Um serviço pode ter baixa cobertura por possuir casos de testes insuficientes, ou porque o cliente do serviço não vai utilizar determinadas funcionalidades, sendo que a cobertura é calculada do sistema como um todo e neste segundo caso, os requisitos de testes cobertos por essas funcionalidades não usadas dentro de um contexto específico são chamados de irrelevantes. Para o primeiro caso de insuficiência, os autores propõem publicar os metadados de teste juntamente com os serviços testáveis para ajudar os integradores a melhorar seu conjunto de testes. Para resolver o segundo problema de não utilização, os autores propõem utilizar um perfil de orquestração que é usado pelo serviço testável para gerar as análises de cobertura para cada contexto usado.

Esses metadados projetados para os serviços testáveis podem vir de duas formas: a priori ou sob demanda. Os metadados a priori fornecem informações que foram previamente criadas e anexadas ao serviço testável quando ele foi lançado. Os metadados sob demanda fornecem informações que foram calculadas/geradas em tempo de execução, baseadas nos metadados a priori. Eles consistem de sugestões fornecidas para ajudar os integradores

[2] Do inglês: More Testable Service by Test Metadata

a melhorarem a cobertura dos serviços testáveis. Integradores devem entender as regras de negócio da composição e dos serviços testáveis para serem capaz de criarem testes de integração, usando os casos de testes sugeridos. É, também, o integrador quem cria o perfil o qual identifica quais operações dos serviços testáveis são realmente usados. Esse perfil também expressa quais requisitos de testes devem ser excluídos do cálculo da medida de cobertura.

Em cada um dos trabalhos que foram analisados, foram adotadas abordagens diferentes para garantir e medir a testabilidade. Em resumo, cada um dos artigos adotou a seguinte: González, Piel e Gross (2009) a testabilidade foi conduzida para teste em tempo de execução no qual considerou a sensitividade e o isolamento do teste; Tsai et al. (2006) observou a testabilidade ao nível da atomicidade, da proveniência dos dados, da integridade dos serviços e da colaboração entre os serviços; Senivongse e Puapolthep (2015) adotou a testabilidade composta por dois atributos: foco no ambiente e processo de simulação; e em Eler, Bertolino e Masiero (2013) foi proposto, a fim de melhorar a falta de controle e observabilidade de serviços a abordagem do MTxTM, a avaliação da cobertura das funcionalidades dos casos de testes, assim como o contexto da sua cobertura.

3.3.2 Questão de pesquisa 2: Quais estudos de caso foram realizados para avaliar/explorar o conceito de testabilidade em *APIs*?

Buscaram-se exemplos concretos de avaliação, ou exploração da testabilidade, por meio de estudos de casos que aplicaram este conceito. Os estudos de casos encontrados que aplicaram o conceito de testabilidade em *APIs* foram os seguintes artigos: González, Piel e Gross (2009), Tsai et al. (2006), Senivongse e Puapolthep (2015) e Eler, Bertolino e Masiero (2013).

O estudo de González, Piel e Gross (2009) teve como ideia principal medir a testabilidade em tempo de execução em sistemas baseados em componentes. Os experimentos realizados foram utilizados para avaliar o custo da cobertura de um recurso específico com possíveis sequências de teste em tempo de execução. O intuito desse experimento foi melhorar a qualidade e a confiabilidade do sistema integrado e para isso foi preciso identificar os sistemas/componentes que não estariam aptos para realizar teste em tempo de execução que possuíssem um custo inaceitável, assim podendo tomar decisões sobre como lidar com essa situação. Os autores conduziram dois estudos de caso, um do domínio marítimo e outro do domínio de segurança. Para isso, eles modelaram os sistemas que eram baseados em componentes, utilizando um grafo de interação de componentes. Eles

mediram a testabilidade considerando os seguintes critérios de cobertura: todos-as-arestas e todas-dependências- do-contexto. O segundo critério requer a invocação dos vértices entre todos os contextos possíveis.

No estudo de Tsai et al. (2006), é ilustrado um estudo de caso de negociações de ações. O autor relata que ao similar com a testabilidade baseada em componentes, a testabilidade de um software SOA pode ser avaliada com a abordagem de pontuar cada critério de avaliação dentro da escala de 0 a 1, de forma a calcular a testabilidade final para todo o software SOA, calculando a soma de todas as pontuações ponderadas de cada critério de avaliação.

Em Senivongse e Puapolthep (2015), houve a realização de um experimento para avaliar a manutenibilidade de um módulo de gerenciamento de atendimento ao cliente. Essa avaliação foi aplicada a um sistema de pagamentos de um provedor de soluções de comunicação na Tailândia. Os autores criaram um questionário e pediram para 6 analistas avaliarem duas versões do sistema, então eles utilizaram a média entre as respostas dos analistas para cada versão.

Eler, Bertolino e Masiero (2013) conduziram um estudo de caso exploratório que considerou comparar a abordagem MTXMT com uma abordagem funcional. O objetivo desse estudo era avaliar se os metadados eram úteis para auxiliar os integradores a melhorar o percentual de cobertura de uma composição de serviços e também se a abordagem era mais efetiva do que uma geração de testes aleatória. Os resultados mostraram que a abordagem ajudou os integradores a criarem novos casos de teste significativos usando um serviço testável e que também foi possível aumentar a medida de cobertura, a qual era baixa no início.

Os autores Eler, Bertolino e Masiero (2013) também conduziram um experimento com o objetivo de medir o quão efetivo é o MTxTM para apoiar integradores a alcançarem uma alta cobertura em um único serviço ao testar sua integração com uma composição. O experimento foi realizado na perspectiva de integradores que tiveram que criar casos de teste para uma composição que utiliza um serviço testável. O principal objetivo do integrador era testar a integração entre a composição e o serviço testável para exercitar ao máximo a estrutura do serviço testável. O objetivo secundário era encontrar falhas no serviço testável. Para isso foram selecionados 12 alunos de pós-graduação no experimento. Em que foram aplicados dois estudos de caso: WSGolfReservation, um serviço do mundo real reusado de um aplicação web open-source projetada para gerenciar as reservas de clubes de golfe; e GolfChampionship, desenvolvida por um dos autores e cuja funcionalidade

é gerenciar jogadores e partidas de campeonatos de golfe. Essa segunda aplicação utiliza a primeira para fazer reservas de campos de golfe para alocar partidas do campeonato.

Os participantes foram divididos em dois grupos. Um grupo que usou a abordagem MTxTM e que teve acesso à interface do serviço testável para obter a análise de cobertura dos metadados sob demanda. O segundo grupo usou a abordagem funcional e só teve acesso à interface da composição GolfChampionship. Como resultado do experimento, os autores verificaram que o MTxTM não ajudou como o esperado para aumentar o número de falhas encontradas em comparação com a abordagem funcional. Entretanto, o número de falhas diferentes encontradas pelos participantes que usaram o MTxTM foi maior que o segundo grupo.

O conjunto de trabalhos descritos exploram o conceito de testabilidade diante de alguns estudos de caso. No estudo de González, Piel e Gross (2009), aplicou-se em dois estudos, um no domínio marítimo e outro do domínio de segurança. Nele, foi considerada a medição da testabilidade diante dos critérios de cobertura do contexto e suas dependências. Em Eler, Bertolino e Masiero (2013), foram conduzidos dois estudos: o primeiro foi um estudo de caso exploratório para avaliar se os metadados eram úteis para auxiliar os integradores a melhorar o percentual de cobertura de uma composição de serviços e também se a abordagem era mais efetiva do que uma geração de testes aleatórios. Como resultado teve o aumento da testabilidade diante da criação de novos casos de teste e o aumento da cobertura. O segundo estudo foi a condução de um experimento com o objetivo de medir o quão efetivo era o MTxTM. Como resultado desse experimento, obteve-se que o método não ajudou como o esperado para aumentar o número de falhas encontradas em comparação com a abordagem funcional. Porém, o método possibilitou identificar falhas diferentes das encontradas pelo grupo que não o utilizou.

O Tsai et al. (2006) e o Senivongse e Puapolthep (2015), apesar de terem conduzido um caso de estudo, em ambos faltam apresentação de guidlines e checklist para melhor compreensão de como foi aplicado e realizado o estudo.

3.3.3 Questão de pesquisa 3: Quais são os desafios na testabilidade em *APIs*?

Com o intuito de encontrar os principais desafios relacionados à testabilidade em *APIs*, procuraram-se, nos artigos, as principais visões, desafios e relatos que pudessem ajudar a responder à questão. Foi possível encontrar desafios em testabilidade nos seguintes trabalhos: González, Piel e Gross (2009), Eler, Bertolino e Masiero (2013) e O'Brien,

Merson e Bass (2007).

Em González, Piel e Gross (2009), foram apresentadas as dificuldades que afetam a verificação da testabilidade em tempo de execução e a dificuldade nos principais pilares levantados: a sensibilidade dos testes e o isolamento de testes. Um pré-requisito que pode ser utilizado para desenvolver as técnicas de teste de isolamento é um estudo prévio da sensibilidade do teste, a fim de encontrar os problemas de testabilidade do tempo de execução que farão o tempo de execução do teste parar, falhar, ou falhar parcialmente. Algumas ideias que podem ser utilizadas para desenvolver as técnicas de teste de isolamento para ganhar testabilidade em tempo de execução são: a técnica de separação de estado, a separação da interação, o monitoramento de recursos e o agendamento de testes.

Em Eler, Bertolino e Masiero (2013), o primeiro desafio citado é que serviços de terceiros geralmente apresentam baixa testabilidade e isso se deve por dois motivos: o primeiro é o custo da atividade de teste, em que testar serviço de terceiros pode ser custoso, pois os custos podem estar associados à sua execução. Enquanto o segundo motivo é por ser caixa preta, em que as integrações, quando fornecidas como caixas pretas, não possuem acesso aos dados internos dos serviços de terceiros.

O'Brien, Merson e Bass (2007) exploram como SOA impacta os diferentes atributos de qualidade, entre eles testabilidade. Segundo os autores, testar um sistema baseado em SOA é mais complexo porque:

(i) é mais difícil configurar e rastrear a execução do teste uma vez que os elementos do sistema residem em diferentes máquinas na rede;

(ii) o código-fonte de serviços externos podem não estar disponíveis. Os casos de teste precisam ser definidos baseados exclusivamente na interface publicada e documentação. Além disso, usuários do serviço podem não ter acesso aos arquivos de log, ou outras saídas geradas quando o serviço externo é executado;

(iii) em alguns casos, serviços são descobertos em tempo de execução, então pode ser impossível prever qual serviço é realmente usado por um sistema até que o sistema seja executado. Além disso, diferentes serviços de diferentes fornecedores podem ser usados várias vezes quando o sistema executa. Os serviços usados podem estar sendo executados em diferentes plataformas, ou sistemas operacionais, que usam diferentes tecnologias de middleware. Assim, construir testes receptíveis e automatizar o processo de teste para tal sistema é desafiador;

(iv) Em soluções web services, algumas vezes o erro está no documento XML. Sendo

que lidar com documentos XML é pesado.

Se um problema em tempo de execução acontece, pode ser difícil encontrar aonde ocorreu. Porque pode estar no usuário do serviço, no fornecedor do serviço, na infraestrutura de comunicação, no agente de descoberta (se houver), ou pode ser devido à carga na plataforma em que o serviço executa. Replicar os problemas em um ambiente de teste pode ser desafiador.

Em resumo, nos trabalhos de González, Piel e Gross (2009) e O'Brien, Merson e Bass (2007), há relatos das dificuldades em relação ao tempo de execução. Já González, Piel e Gross (2009) apresentam a dificuldade em identificar quais são os problemas que farão o teste parar, falhar, ou falhar parcialmente em tempo de execução; e O'Brien, Merson e Bass (2007) relatam a dificuldade de encontrar o local da falha em tempo de execução. Já Eler, Bertolino e Masiero (2013) relacionam as principais dificuldades que se têm ao testar serviços de terceiros: a baixa testabilidade de serviço devido ao custo da atividade de teste e a falta de conhecimento do código, no qual o software é visto como uma caixa preta.

3.4 Ameaças à validade

Como visto na literatura por Malla e Gurung (2012), ameaças à validade podem e afetam a precisão do resultado que se encontra sob investigação. Levando isso em consideração, há possíveis ameaças que podem afetar os resultados alcançados pela condução do mapeamento sistemático.

Primeiramente foi considerada a busca da *string* elaborada em duas fontes digitais de artigos, IEEExplorer e o Scopus, e levou-se em consideração para a busca o título e o abstract. Esse método pode levar à exclusão de estudos relevantes e redução nos estudos avaliados, tanto por conta da forma de seleção como pela quantidade limitada das fontes utilizadas. Então, para mitigar a ameaça da possível exclusão de um estudo válido, toda a seleção aconteceu mediante a uma revisão pareada. Ou seja, todo estudo excluído foi validado pela dupla de especialistas na área em questão. Já para mitigar a ameaça à fonte de pesquisa escolhida, foi previamente feita uma busca nas fontes de pesquisa e as duas escolhidas foram as que apresentaram os resultados mais relevantes, por isso a escolha. De maneira complementar, a técnica aplicada para o *snowballing backward* possibilitou a inclusão por pesquisas nos indexadores do Google Schoolar.

3.5 Considerações do capítulo

Esse mapeamento sistemático foi realizado para encontrar trabalhos que apresentassem a abordagem e/ou a utilização do conceito de testabilidade em aplicações de interface de programação-*APIs*. Nenhum deles definiu em específico uma forma de prover qualidade em *APIs* por meio da testabilidade. No próximo capítulo, é apresentada a proposta desta dissertação que é verificar se o *SigniFYIng APIs* possibilita apoiar a testabilidade mediante a elaboração de testes manuais funcionais. Além disso, também é apresentado o experimento elaborado que aplica a ferramenta *SigniFYIng APIs* no momento em que o testador elabora casos de teste para verificar se a investigação das falhas de comunicabilidade podem apoiar a testabilidade de uma *API*.

4 Aplicando *SigniFYIng APIs* para investigar a testabilidade

Alguns fatores interferem na testabilidade de um software. O controle dos dados de entrada em uma suíte de teste, por exemplo, depende da qualidade da documentação existente. O processo de documentação envolve tarefas como elaboração de casos de teste, projeto de teste, resumo de teste, registro de execução de teste e procedimentos de teste (BINDER, 1994). Com o intuito de melhorar a elaboração dos casos de teste, em específico de *APIs*, este capítulo apresenta os resultados da aplicação da ferramenta *SigniFYIng APIs* para investigar a testabilidade.

4.1 Estudo - aplicando o *SigniFYIng APIs*

O objetivo deste estudo é a investigação da capacidade da ferramenta conceitual *SigniFYIng APIs* em ser utilizada como um método capaz de identificar fatores que interferem na testabilidade de um software. Considerando que o *SigniFYIng APIs* orienta a reflexão do analista a identificar as falhas de comunicação presentes em uma *API*, surge o seguinte questionamento:

(i) Será que a identificação de problemas de comunicabilidade pode levar o testador a encontrar elementos para a melhoria dos projetos de casos de testes e a apoiar a testabilidade?

A elaboração da pesquisa foi dividida em quatro etapas: elaboração da proposta, planejamento e execução do experimento, metodologia de análise de dados e análise dos resultados.

A elaboração do estudo consistiu em inserir a metodologia proveniente da Engenharia Semiótica, a ferramenta *SigniFYIng APIs* , no contexto de teste de software. Para ilustrar a situação do estudo da pesquisa, na Figura 8, se tem uma situação genérica, na qual os envolvidos são: o projetista da *API*, o desenvolvedor e o testador. O projetista da *API*

é o responsável por desenvolver a *API* e a documentação da mesma. O desenvolvedor é quem irá utilizar a *API* que foi elaborada pelo projetista, para assim incluir a *API* no software que está sendo produzido. O testador precisa verificar se o software elaborado pelo programador consumidor foi feito adequadamente, o que envolve testar a *API* que se encontra nesse software. Esses envolvidos, muitas vezes, são de equipes distintas e a comunicação entre eles é realizada apenas por documentações. O estudo desenvolvido foca no testador que deve ter em mãos a documentação fornecida pelo projetista da *API* ao iniciar os testes. Com a documentação da *API*, o testador aplica a ferramenta *SigniFYIng APIs* a fim de encontrar possíveis falhas nessa documentação. Como hipótese da pesquisa, se tem que o testador, ao aplicar o *SigniFYIng APIs* previamente, pode projetar casos de teste abrangendo as falhas de comunicabilidade relatadas e, consequentemente, forneça uma melhor qualidade do software produzido.

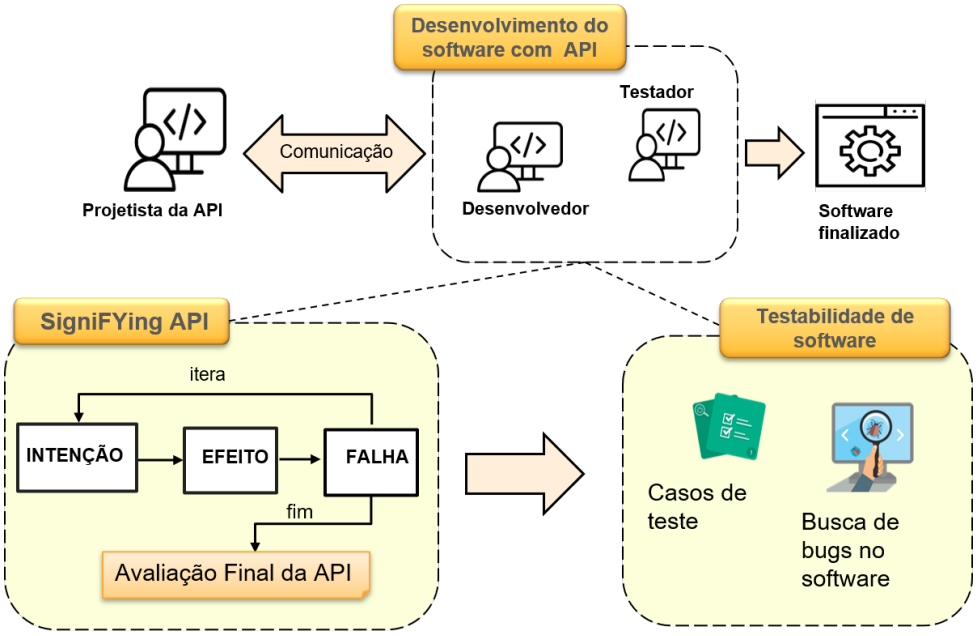

Figura 8: Estudo da testabilidade em *API* com o *SigniFYIng APIs*, elaborada pela autora.

A segunda etapa consistiu no planejamento e execução do experimento para verificar se a identificação dos problemas de comunicabilidade por testadores, com a aplicação da ferramenta *SigniFYIng APIs*, ajuda-os a projetar melhores casos de testes. O planejamento da metodologia envolveu: a definição das questões de pesquisa; a definição do objeto do estudo; a execução de um estudo piloto; o recrutamento e seleção de participantes; o treinamento dos participantes; a definição das etapas e materiais do experimento; e a análise dos resultados. A metodologia será explicada em detalhes na seção 4.2. Na execução do experimento, é abordado o que foi modificado após a aplicação do estudo piloto: como os participantes selecionados foram agrupados, a preparação dos participantes

e como foi a execução do experimento. Os detalhes da execução do experimento pode ser visto na seção 4.3

4.2 Metodologia

Essa seção aborda questões que envolvem a metodologia/planejamento que foram necessárias para verificar se o estudo de aplicar o *SigniFYIng APIs* pela equipe de testadores ajuda na elaboração de melhores casos de teste. Primeiramente foi necessário definir os objetivos e as questões de pesquisa, assim como as métricas que deveriam ser consideradas, como pode ser visto na seção 4.2.1. Com as questões de pesquisas definidas e para obter as respostas dessas questões, foi necessário definir um objeto de estudo para ser aplicado em um experimento. Para a realização do experimento, foi necessária a preparação do material que seria utilizado, assim como foi definido que os participantes seriam divididos em dois grupos: um grupo controle e um grupo utilizando o *SigniFYIng APIs*, abreviado como grupo SAPI. Foi definida também a forma pela qual seria realizada a escolha dos participantes. Tendo realizado todo o planejamento do experimento, foi executado um estudo piloto, para verificação do material elaborado. Por fim, foi elaborado como seria realizada a execução do experimento e a forma de análise dos dados empíricos coletados.

4.2.1 *Questões de pesquisa*

A definição das questões de pesquisa foram definidas através do paradigma *Goal Question Metric (GQM)*. O GQM é utilizado na Engenharia de Software para adequar e integrar metas aos modelos do processos de software, produtos e perspectivas de qualidade de interesse, com base nas necessidades do projeto (SOLINGEN; BERGHOUT, 1999).

Solingen e Berghout (1999) disseram que:

> O resultado da aplicação desse método se tem a especificação de um programa de medição visando um determinado conjunto de questões e um conjunto de regras para a interpretação dos dados de medição. O princípio por trás do método GQM é que a medição deve ser orientada a objetivos.

O *GQM* define a obtenção do objetivo das questões de pesquisa e das métricas. O objetivo é a meta a qual a pesquisa quer alcançar. As questões de pesquisa são declarações sobre o conhecimento que é buscado, ou que é esperado a sua descoberta durante um experimento. A descoberta, ou obtenção desse conhecimento, demostra que o

Objetivo	**Sigla**	**Obter indícios sobre a utilidade da ferramenta *SigniFYIng APIs* em melhorar a testabilidade de uma *API*.**
Questão	Q1	O que foi identificado com a aplicação da ferramenta *SigniFYIng APIs*?
Métrica	M1.1	Listar as CDNs encontradas.
Métrica	M1.2	Listar os tipos de falhas separados em: falha parcial, falha completa e falha total.
Métrica	M1.3	Listar as etiquetas de comunicabilidade encontradas.
Métrica	M1.4	Avaliação subjetiva do que foi encontrado.
Questão	Q2	A aplicação da ferramenta *SigniFYIng APIs* sobrecarrega/dificulta o processo tradicional de testes?
Métrica	M2.1	Avaliação subjetiva sobre as expectativas percebidas que a dificultam.
Métrica	M2.2	Avaliação subjetiva sobre as expectativas percebidas que a sobrecarregam.
Questão	Q3	A solução proposta pelo *SigniFYIng APIs* auxiliou o processo de teste e contribuiu para melhorar a testabilidade da *API* testada?
Métrica	M3.1	Avaliação subjetiva sobre as expectativas percebidas que o auxiliaram.
Métrica	M3.2	Avaliação subjetiva sobre as expectativas percebidas que contribuíram.
Questão	Q4	Em quais situações o *SigniFYIng APIs* obteve melhores resultados?
Métrica	M4.1	Avaliação subjetiva dos efeitos encontrados e os casos de teste escritos comparando com o grupo controle;
Métrica	M4.2	Contabilizar casos de teste mal projetados pelo grupo SAPI.
Métrica	M4.3	Contabilizar casos de teste mal projetados pelo grupo controle.
Métrica	M4.4	Contabilizar tempo gasto na elaboração dos casos de teste no grupo controle.
Métrica	M4.5	Contabilizar tempo gasto na elaboração dos casos de teste no grupo SAPI.
Métrica	M4.6	Avaliação subjetiva da elaboração dos casos de teste.

Tabela 8: Estrutura do GQM.

experimento atingiu o objetivo esperado (RUNESON et al., 2012), assim como as métricas são refinamentos das questões em um processo de medições, de modo a obter as informações suficientes para responder às questões e ao objetivo desejado (SOLINGEN; BERGHOUT, 1999).

Diante desse conceito, foi elaborado o objetivo da pesquisa que é obter indícios sobre a utilidade da ferramenta *SigniFYIng APIs* em melhorar a testabilidade de uma *API*. A partir do objetivo de investigação principal definido, foram elaboradas as questões investigativas e as métricas para cada uma dessas questões estruturadas, utilizando o método do GQM, como visto na Tabela 8.

Cada uma das questões de pesquisa definidas, como vistas na Tabela 8, teve como objetivo identificar e descobrir algo para o experimento. A questão de pesquisa *Q1*

teve como objetivo contabilizar e classificar a quantidade de falhas, as etiquetas de comunicabilidade e as CDNs ao aplicar o *SigniFYIng APIs* . A questão de pesquisa *Q2* verifica se, ao utilizar a ferramenta, os testadores conseguem projetar melhores casos de teste sem gerar grandes dificuldades. A questão de pesquisa *Q3* tem em vista identificar como os grupos se saíram na elaboração dos casos de teste, verificando a qualidade obtida em cada um dos grupos, assim como as expectativas percebidas durante a utilização da aplicação do *SigniFYIng APIs* no experimento. E, por fim, a quarta questão de pesquisa, definida como *Q4*, visa identificar se existe alguma situação na qual o *SigniFYIng APIs* obteve melhores resultados.

4.2.2 Definição do objeto de estudo

A seleção dos cenários para aplicação do estudo considerou encontrar *APIs* com os seguintes quesitos: que continham *bugs* [1] ao executar; que apresentassem documentações contendo ambiguidades e definições vagas da sua utilização; e que abordassem diferentes tipos de falhas conforme categorizado no *SigniFYIng APIs* como: falha do parcial, falha temporária e falha total. A escolha dos cenários foi realizada por duas pesquisadoras da área de teste de software. Uma delas foi a própria autora dessa pesquisa e a outra foi aluna de Iniciação Científica do projeto. Os cenários escolhidos foram validados pela orientadora, que possui mais experiência na área. As profissionais procuraram cenários que apresentassem essas características e para isso foi necessário testar previamente os cenários para verificar se os cenários continham *bugs*, ou uma documentação ambígua, assim aplicaram o *SigniFYIng APIs* de forma a validar se os cenários apresentavam falha parcial, temporária e total.

Para encontrar as *APIs* que seriam utilizadas, foi feita previamente uma busca no Google [2] e em alguns fóruns, como o StackOverflow [3], procurando possíveis problemas comuns em *APIs*. Como não foi possível achar algo que se enquadrasse para esse experimento, decidiu-se selecionar os sistemas Trello [4] e Moodle [5] como objetos de estudo, uma vez que suas *APIs* se adequavam às condições determinadas e também os problemas existentes já eram de conhecimento das pesquisadoras. O Trello permite fazer a gestão de tarefas de forma colaborativa por meio de quadros (*boards*), em que são inseridos cartões (*cards*) móveis que contêm a descrição da atividade a ser realizada, os prazos para a conclusão da

[1] O termo *bug* foi utilizado como sinônimo de falha, de acordo com a nomenclatura utilizada por Delamaro, Jino e Maldonado (2021). A utilização desse termo se deu para evitar confusão com o termo falhas, que é utilizado na ferramenta *SigniFYIng APIs*.

[2] https://google.com

[3] https://stackoverflow.com

[4] https://trello.com

[5] https://moodle.org

mesma e seus objetivos a serem concluídos. Já o Moodle é uma plataforma voltada para a produção de um ambiente de ensino on-line e pode ser configurado de acordo com as necessidades das instituições de ensino. Nesse sistema, é possível que sejam criados, por exemplo, fóruns para troca de informações, como também comentários em uma discussão (MOODLE, 2021).

Para a realização do experimento, foram escolhidos um conjunto de cinco cenários, três do Trello e dois do Moodle. Em relação aos cenários das *APIs* do Trello, todas tinham algum *bug* ou apresentavam uma documentação com pouca descrição. Ao aplicar o *SigniFYIng APIs* nessas *APIs*, foram encontradas falhas parciais e temporárias.

Já nos cenários do Moodle, as *APIs* não apresentavam nenhum *bug*, porém foram escolhidas por apresentar uma documentação contendo muitas ambiguidades nos nomes das chamadas e nas variáveis das *APIs*. Ao aplicar o *SigniFYIng APIs* nessas *APIs*, foram encontradas previamente falhas totais.

4.2.2.1 A aplicação de teste

Os cenários utilizados no experimento realiza chamadas para a *API*. De modo a simplificar o teste destes pelos participantes, foi utilizada uma aplicação desenvolvida para a realização dessa pesquisa, como vista na Figura 9. Ela acessa as *APIs* que devem ser testadas, evitando, assim, a necessidade de instalação prévia de ferramentas externas para realizar as chamadas.

A aplicação foi fornecida aos participantes do estudo na modalidade on-line. Ela apresenta: um tutorial de como utilizar a aplicação, um campo para preencher o nome do participante e os cinco cenários distintos a serem testados. Cada cenário contém sua documentação relacionada para prover o entendimento necessário de como os campos que estão sendo utilizados na *API* funcionam e quais informações são aceitas, possibilitando assim a elaboração dos casos de testes pelos participantes. Ao ser utilizada, a aplicação solicita o nome do participante e registra todas as ações realizadas durante o experimento. Todas essas informações puderam ser consultadas utilizando a exportação do arquivo JSON como visto na seção 4.2.2.1.

4.2.2.2 Descrição dos cenários

Para o entendimento de cada cenário, será descrito o propósito de cada um deles e as principais falhas identificadas previamente, as quais levaram a serem escolhidos para o

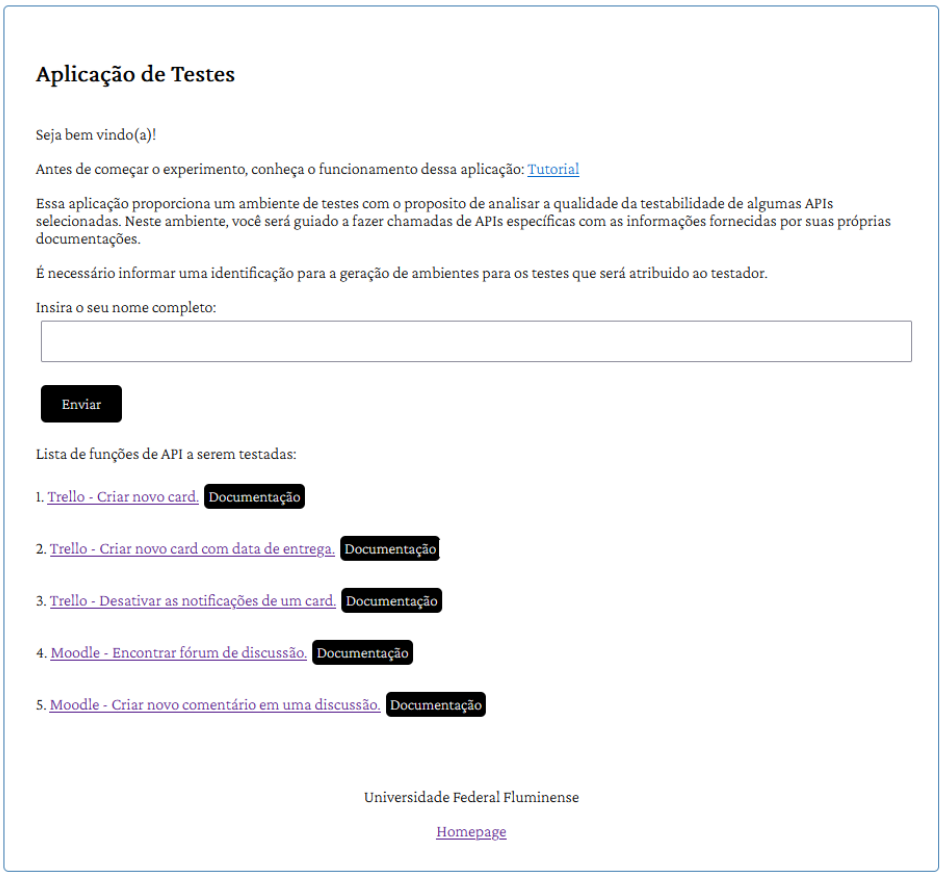

Figura 9: Aplicação de teste.

estudo. Em todos os cenários descritos, os valores de *'token'*, *'chave'* e o *'identificador da lista'* foram preenchidos automaticamente pela aplicação. Sendo necessário apenas o preenchimento dos parâmetros pedidos para completar a *URL* da requisição.

O cenário 1, denominado de "1.Trello — Criar novo card", teve como objetivo inserir um novo cartão em uma lista designada. A aplicação disponibiliza o preenchimento de dois campos para o participante: o campo nome do cartão e posição do cartão. Ou seja, é necessário a definição do nome do cartão e a posição na qual esse é inserida. Como se sabia previamente, o cenário apresentava falha ao tentar inserir um cartão em uma posição escolhida. Essa falha ocorre, em específico, quando já existem mais cartões nessa lista. Portanto, foi provida uma lista já populada por três cartões. O intuito principal é que o participante consiga adicionar um novo elemento na lista entre a primeira e segunda posição, ocupando o segundo lugar mais ao topo da lista.

O cenário 2, denominado de "2.Trello - Criar card com data de entrega", teve como objetivo inserir um novo cartão com uma data de entrega em uma lista designada. Os campos solicitados para preenchimento foram: o nome do cartão, a posição e a data de

entrega. Nesse cenário, foi visto que a documentação [6] provida do Trello não apresentava a informação clara de como o campo data de entrega deveria ser preenchida. Nela, foi informado apenas que era aceito o formato data e não forneceu explicitamente o padrão adotado. Notou-se que, ao inserir diferentes formatos, o cartão era criado com as informações erradas.

O cenário 3, denominado de "3.Trello - Desativar as notificações de um card", teve como objetivo utilizar a chamada da *API* para deixar de seguir, indicado como subscrito na aplicação, um cartão previamente criado. Nesse cenário, os campos dados para preenchimento foram: o nome do cartão e o subscrito. O cartão foi criado na inicialização dos quadros no Trello com o argumento de subscrito definido como ativado, ou seja, com o valor *true*. Esse cenário apresentava a falha quando o argumento do subscrito era desativado, ou seja, alterado para *false*.

O cenário 4, denominado de "4.Moodle - Encontrar um fórum de discussão", teve um foco um pouco diferente dos cenários do Trello. Nesse cenário, foram disponibilizadas três chamadas à *API*: *core course get courses*, *mod forum get forums by courses* e *mod forum get forum discussions*.

A primeira chamada, *core course get courses*, retorna detalhes de um curso. Essa função não requer nenhum parâmetro, mas retorna valores que deverão ser usados em suas próximas duas chamadas.

A segunda chamada, *mod forum get forums by courses*, requer o preenchimento do campo *course id*, o qual deve ser utilizado o *id* que foi obtido na primeira chamada, *mod forum get forums by courses*. Essa chamada retorna todos os fóruns que existem no curso pelo *id* que foi informado.

Na terceira chamada, *mod forum get forum discussions*, é solicitado o preenchimento do campo *forum id* para obter as discussões de um fórum, pois cada fórum possui um número qualquer de discussões. As discussões são o espaço do curso em que usuários podem submeter perguntas e respostas em forma de comentários.

O cenário 4 foi escolhido pois apresentava uma falha na documentação, em específico na escolha do nome da variável *id*. A chamada da *API* retornava a *id* sem deixar claro a quem pertence esse *id*, se era do fórum, do curso, ou outra entidade, já que o nome estava definido apenas como *id*.

O cenário 5, denominado de "5.Moodle — Criar comentário em uma discussão",

[6] https://developer.atlassian.com/cloud/trello/rest/api-group-cards/#api-cards-post

disponibilizou quatro chamadas distintas à *API*: *core comment add comments*, *mod forum add discussion post*, *core message send messages to conversation* e *mod chat send chat message*, em que cada uma delas foi solicitado o preenchimento dos parâmentos que eram necessários para completar a chamada. O intuito desse cenário era adicionar uma resposta a uma discussão previamente criada no cenário 4. Apesar de terem sido fornecidas quatro chamadas, apenas uma, de fato, adicionava um novo comentário a uma discussão, a qual era a chamada *mod forum add discussion post*. Esse cenário foi escolhido tendo em vista a falta de clareza da documentação nos nomes das chamadas das *APIs*. A documentação apresentou várias chamadas com nomes similares e não deixou claro o que cada uma delas fazia. Desse modo, além de precisar compreender objetivo do cenário, o participante teve que, por meio da leitura da documentação, verificar se ele saberia qual das chamadas deveria usar para completar a tarefa estabelecida.

É importante ressaltar que apesar dos cenários apresentarem objetivos, o foco é em testar os cenários e encontrar as falhas por meio dos casos de teste. O objetivo do cenário foi apenas uma forma de delimitar o teste.

4.2.2.3 Mapeamento dos casos de teste

Escolhidos os cenários, elaborou-se um mapeamento dos principais casos de testes. Para isso, utilizou-se o critério de análise de causa e efeito, possibilitando, assim, a construção da *tabela verdade* de cada cenário, como pode ser visto na Figura 10. Esse mapeamento foi realizado para rastrear os cenários de testes possíveis e facilitar a correlação dos testes elaborados pelos participantes, como visto na seção 4.2.7.1. O mapeamento foi elaborado pela aluna mestranda e foi revisado pela orientadora.

Primeiramente, foi elaborada a *tabela verdade* de cada um dos cenários, separando as causas e os efeitos previstos. Como exemplo, a Figura 10 mostra a tabela referente ao cenário 1. Diante da *tabela verdade* construída, foram obtidos os casos de testes de cada um dos cenários, como pode ser visto na Figura 11. Por exemplo, no cenário 1, mapeou-se um conjunto de CTXX, do qual XX é o número que corresponde a uma situação específica. O CT01 representa o teste no qual o testador deixaria o campo 'nome' com o valor igual a *vazio* e selecionaria a 'posição' igual a *top*. Como resultado dessa operação, é esperado que o sistema crie um *card* sem nome na posição *top*.

A primeira linha da Figura 11 mostra os valores válidos para o campo nome: vazio e válido, assim a posição pode assumir os seguintes valores: *top, bottom, positive float, vazia, ou uma opção inválida*. Para cada uma dessas opções, ou seja, as causas, foram

relacionados os possíveis efeitos esperados. Os campos que constam um *hífen* (-) são as situações que não são viáveis, ou que não existem. No Apêndice B, podem ser vistas as *tabelas verdades* e os casos de testes relacionados dos demais cenários.

	Cenário 1 Criar um card em uma lista. Adicionar o card criado entre a primeira e segunda posição.								
Causas	CT01	CT02	CT03	CT04	CT05	CT06	CT07	CT08	CT09
Nome vazio	1	1	1	1	0	0	0	0	-
Nome valido	0	0	0	0	1	1	1	1	-
Posição informada: top	1	0	0	0	1	0	0	0	-
Posição informada: bottom	0	1	0	0	0	1	0	0	-
Posição informada: positive float	0	0	1	0	0	0	1	0	-
Posição vazia	0	0	0	1	0	0	0	1	-
Posição inválida	0	0	0	0	0	0	0	0	1
Efeitos									
Cria um card	1	1	1	1	1	1	1	1	0
Cria um card sem nome	1	1	1	1	0	0	0	0	0
Cria um card com nome	0	0	0	0	1	1	1	1	0
Cria um card no top	1	0	0	0	1	0	0	0	0
Cria um card no bottom	0	1	0	0	0	1	0	0	0
Cria um card na posição float	0	0	1	0	0	0	1	0	0
Cria um card na posição default	0	0	0	1	0	0	0	1	0

Figura 10: *Tabela verdade* construída para o cenário 1.

Cenário 1	**Causa**		**Efeito**
	Vazio *Válido*	*Top* *Bottom* *Positive float* *Vazia* *Inválida*	
	Nome	**Posição**	**Resultado esperado**
CT01	Vazio	Top	Cria um card sem nome na posição top
CT02		Bottom	Cria um card sem nome na posição bottom
CT03		Positive float	Cria um card sem nome na posição informada no float
CT04		Vazia	Cria um card sem nome na posição default
CT05	Válido	Top	Cria um card com nome na posição top
CT06		Bottom	Cria um card com nome na posição bottom
CT07		Positive float	Cria um card com nome na posição informada no float
CT08		Vazia	Cria um card com nome na posição default
CT09	-	Inválida	Não cria um card

Figura 11: Casos de testes relacionados ao cenário 1.

4.2.3 Estudo piloto

O estudo piloto foi feito pela pesquisadora autora juntamente com a aluna de Iniciação Científica do projeto para garantir que: (i) a forma de aplicação é adequada, (ii) os termos usados na pesquisa são claros e compreensíveis e (iii) as respostas às questões de pesquisa são significativas.

A aplicação do estudo piloto foi planejada para ser executada com dois participantes distintos, um para o grupo controle e outro para o grupo do SAPI. Para isso, foi levado em consideração que os dados coletados não foram considerados para a análise desse trabalho.

4.2.4 Seleção de participantes

A seleção dos participantes foi realizada pela pesquisadora autora dessa dissertação, por meio de convites enviados on-line, conforme o modelo Carta Convite (ver Apêndice D). Como a pesquisa é qualitativa, o que permite a captura de detalhes sutis (LEITÃO; PRATES, 2017), optou-se por selecionar uma amostra de participantes reduzida, com no máximo 10 participantes.

A aplicação da pesquisa foi realizada mediante a um grupo de participantes voluntários que aceitaram participar do experimento. Esses participantes, para poderem estar no experimento, tinham que ter conhecimentos, ou atuarem na área de teste de software. Para isso, foi feita a apresentação dos objetivos e a coleta do consentimento dos participantes. Nela, foi explicado de uma forma geral o objetivo da pesquisa com um termo de consentimento para a realização, o qual deveria ser assinado por cada participante (Termo de Consentimento - ver Apêndice C).

A divisão dos participantes nos dois grupos foi feita por meio de um formulário que foi enviado a cada participante, o qual continha perguntas sobre o nível de experiência na área de teste de software. Nele, possuem perguntas abertas e o objetivo foi obter informações acerca do conhecimento do participante na área de teste de software. O preenchimento se deu no máximo de 2 minutos. As perguntas do formulário foram as seguintes:

1. Qual seu nível de experiência em teste de software?
2. Já havia tido alguma experiência utilizando, ou desenvolvendo, alguma *API*?
3. Já se deparou com a situação na qual, ao consumir algum dados de uma *API*, ela não retornou o esperado?

4. Já ocorreu uma situação na qual, ao projetar um caso de teste, percebeu que a resposta divergia do esperado devido à falta de informação na documentação?

5. O quão importante você considera a documentação de uma *API* para realizar testes de software?

6. Utiliza alguma técnica para elaborar casos de teste?

Por meio desse formulário, foi possível definir o perfil do participante, dividindo em uma categoria de participantes que continha o perfil acadêmico com conhecimentos de testes e o outro em que continha participantes com experiência profissional na área. Em vista desses dois grupos, foram distribuídos os participantes que iriam participar do grupo controle e do grupo SAPI. Em cada um dos grupos, equilibrou-se os participantes com o perfil acadêmico e participantes que são profissionais da área.

4.2.5 Preparação dos participantes

A preparação dos participantes foi planejada para ser feita apenas com o grupo SAPI. Como a execução do experimento envolvia a aplicação da ferramenta *SigniFYIng APIs*, a qual não era de conhecimento dos participantes, foram criados vídeos sobre o *SigniFYIng APIs*. A primeira videoaula [7] instrui o participante acerca dos conceitos e da ferramenta, enquanto na segunda, é mostrado um exemplo prático da aplicação do *SigniFYIng APIs* em um sistema e-gov. Nesta apresentação, foram mostrados os objetivos de utilizar uma ferramenta da área de IHC na área de teste de software, assim como foi explicado em detalhes como funciona a ferramenta *SigniFYIng APIs*.

Como o *SigniFYIng APIs* apresenta termos bem específicos e etapas que devem ser seguidas, foram elaborados formulários, um para cada cenário, para serem utilizados durante a execução do experimento. O intuito desse formulário foi de simplificar e guiar o participante para aplicação do *SigniFYIng APIs*. Em adicional, foi criado, também, um manual rápido [7] para ser utilizado durante o experimento e/ou consultar eventuais dúvidas.

4.2.6 Execução do experimento

A execução do experimento foi planejada para ser executada com 10 participantes organizados em dois grupos: o grupo controle e o grupo SAPI. O tempo total estimado

[7] https://github.com/camilamorelli/testabilityInAPIs_with_SemioticEngineering

para o experimento foram quatro horas, podendo acontecer em mais ou menos tempo, dependendo do participante. O experimento foi presencial e realizado individualmente, no qual cada pessoa recebeu as instruções de como realizá-lo via e-mail. As orientações continham informações sobre: o acesso à aplicação; os links dos cinco formulários para aplicação do *SigniFYIng APIs* em cada um dos cinco cenários; o modelo de caso de teste a ser preenchido; e o formulário de *feedbacks* do experimento. Ressaltando que apenas o grupo SAPI recebeu os cinco formulários, pois era para a aplicação da ferramenta.

Cenário 1: Criar novo Card - Trello

Objetivo: O propósito do teste é inserir um novo Card em uma lista designada. A lista provida já estará populada por três cartas. O objetivo é conseguir adicionar um novo elemento na lista entre a primeira e segunda posição, ocupando o segundo lugar mais ao topo da lista. Os valores de token, chave e o identificador da lista já serão preenchidos automaticamente. Insira somente os valores para os parâmetros pedidos para completar a URL informada abaixo.
Data(s)do Teste:
Notas e Resultados:
Tempo gasto:

ETAPAS / RESULTADOS DO SCRIPT DE TESTE				
PASSO	ETAPA / ENTRADA DO TESTE	RESULTADOS ESPERADOS	RESULTADOS REAIS	PASSOU / FALHOU
1				
2				
3				
4				

Figura 12: Modelo de caso de teste para o grupo controle.

O e-mail enviado aos participantes continha um *link* para acessar a aplicação de teste que continha os cinco cenários com suas respectivas documentações, como vista na seção 4.2.2.2. Nesse e-mail, foi informado aos participantes que os casos de testes deveriam usar como base a documentação fornecida [8] [9].

O experimento foi elaborado da seguinte forma: para o grupo controle, foi solicitado que o participante elabore os casos de testes para cada um dos cenários, como visto na Figura 12. Nessa imagem, por exemplo, mostra como, no cenário 1, são apresentados o objetivo do teste e os campos para preenchimento como a data em que o teste foi realizado, assim como notas, resultados (caso o participante tiver algum comentário a ser feito) e o tempo gasto do teste. No campo etapas/entrada do teste, se solicitou que fosse

[8] A documentação do Moodle somente está disponível para usuários logados no sistema.

[9] A documentação do Trello pode ser vista no link https://developer.atlassian.com/cloud/trello/rest/api-group-cards/#api-cards-post.

descrito o passo realizado no teste; em resultados esperados, era para ser informado o que o participante esperava encontrar; enquanto em resultados reais, era para descrever o que de fato foi encontrado no cenário de teste. No campo passou/falhou, o participante deveria informar se o teste passou, ou falhou.

Já para o grupo SAPI, foi fornecido um conjunto de cinco formulários, um para cada cenário, de forma que o participante utilizaria o formulário para aplicar a metodologia do *SigniFYIng APIs* que foi aprendida. Em seguida, deveria ser solicitada a elaboração dos casos de teste, como visto na Figura 13. O documento era semelhante ao do grupo controle, porém com a inclusão dos dados que foram obtidos da execução do *SigniFYIng APIs*. Essa inclusão permitiu que o participante conseguisse verificar as respostas obtidas e, assim, implementar os casos de testes se baseando nelas.

Resultado do SAPI para o Cenário 1				
Análise obtida nas etapas de Efeito (análise da expressão) e Falha (execução do teste)	Frame Metacomunicação	Efeito da Metacomunicação	CDN	Etiqueta de comunicabilidade
Dados obtidos pelo participante	Dados obtidos pelo participante	Dados obtidos pelo participante	Dados obtidos pelo participante	Dados obtidos pelo participante

Cenário 1: Criar novo Card - Trello

Objetivo: O propósito do teste é inserir um novo Card em uma lista designada. A lista provida já estará populada por três cartas. O objetivo é conseguir adicionar um novo elemento na lista entre a primeira e segunda posição, ocupando o segundo lugar mais ao topo da lista. Os valores de token, chave e o identificador da lista já serão preenchidos automaticamente. Insira somente os valores para os parâmetros pedidos para completar a URL informada abaixo.
Data(s)do Teste:
Notas e Resultados:
Tempo gasto:

ETAPAS / RESULTADOS DO SCRIPT DE TESTE				
PASSO	ETAPA / ENTRADA DO TESTE	RESULTADOS ESPERADOS	RESULTADOS REAIS	PASSOU / FALHOU
1				
2				
3				
4				

Figura 13: Modelo de caso de teste para o grupo SAPI.

Após finalizar o experimento, deveria ser preenchido o formulário sobre a avaliação da experiência dos participantes no experimento. O desenho do questionário considerava alguns cuidados para diminuir os riscos na coleta dos dados: linguagem simples (sem termos muito técnicos); questões com apenas uma pergunta; questões sem duplo negativo;

questões que investigam situações recentes. Além disso, incluía questões para coletar opiniões e fatos vividos pelos participantes. As perguntas foram totalmente discursivas para que o participante pudesse explicar com detalhes como foi a realização do experimento. As perguntas que foram abordadas com o grupo controle são:

1. Já conhecia a *API* em questão?
2. Quanto à documentação da *API*, achou de fácil entendimento?
3. O quão fácil foi utilizar a *API*?
4. Qual foi o tempo gasto para a criação de casos de teste?
5. Quantos casos de teste foram possíveis elaborar?
6. Qual técnica foi utilizada para a criação dos casos de teste?

Já para o grupo SAPI, as perguntas foram as seguintes:

1. Já conhecia a *API* em questão?
2. O quão fácil foi aprender a ferramenta?
3. O quão fácil foi utilizar a *API*?
4. O quanto a ferramenta ajudou a elaborar os casos de teste?
5. Caso a decisão de usar a ferramenta fosse exclusivamente sua, você usaria o *SigniFYIng APIs*, ou adotaria outra abordagem?
6. Quanto à documentação da *API*, achou de fácil entendimento?
7. Qual foi o tempo gasto para a criação de casos de teste?
8. Quantos casos de teste foram possíveis elaborar?
9. Qual foi a sua impressão ao utilizar a ferramenta? Descreva o que achou da ferramenta e diga se vê algum ponto positivo, ou negativo, no seu uso para área de teste de software.

4.2.7 Análise de dados

A análise aqui apresentada consiste na forma que os dados coletados (os casos de teste e os *feedbacks*) devem ser preparados para serem analisados. Para uniformizar os casos de testes elaborados pelos participantes, é proposta a uniformização deles utilizando a correlação dos casos de testes. Para a categorização dos *feedbacks* obtidos do experimento, é utilizado o Método de Explicitação do Discurso Subjacente (MEDS) (NICOLACI-DA-COSTA; LEITÃO; ROMÃO-DIAS, 2004).

Todo o material coletado dos participantes, formulários e os casos de testes, consta de forma anonimizada no repositório [10] disponibilizado com os resultados da pesquisa. No repositório, também consta toda a preparação dos dados que foram utilizados para a análise do experimento.

4.2.7.1 Correlação dos casos de teste

A etapa de correlação dos casos de teste foi realizada previamente em 4.2.2, em que utilizou-se o mapeamento dos cenários, como pode ser visto na Figura 14. Essa correlação foi feita da seguinte forma: para cada um dos grupos, foram analisados por participante os casos de testes elaborados, ou seja, os cenários identificados pelo participante. Para cada caso de teste identificado, foi relacionado com o caso de teste presente no mapeamento. Para verificar e entender melhor o caso de teste que o participante realizou, foi consultado o arquivo JSON obtido pela interação do participante com a aplicação. Como o grupo SAPI aplicou primeiramente a ferramenta nos cenários e essa já possibilita o encontro de falhas, além de ter sido feita a identificação dos casos de teste, foi verificado, então, se os participantes descrevem alguma falha levantada apenas durante a aplicação do *SigniFYIng APIs* a qual deveria ter sido incluída na elaboração dos casos de teste. Todo o material foi correlacionado para mapear os casos de teste, encontrados por ambos os grupos.

Vale ressaltar que, para os casos de teste mal projetados pelos participantes, ou seja, que não se enquadravam em testes funcionais, como testes de segurança, os mesmos não devem ser correlacionados. Testes relativos à aplicação disponibilizada, e não à resposta da *API*, também foram desconsiderados.

[10] https://github.com/camilamorelli/testabilityInAPIs_with_SemioticEngineering

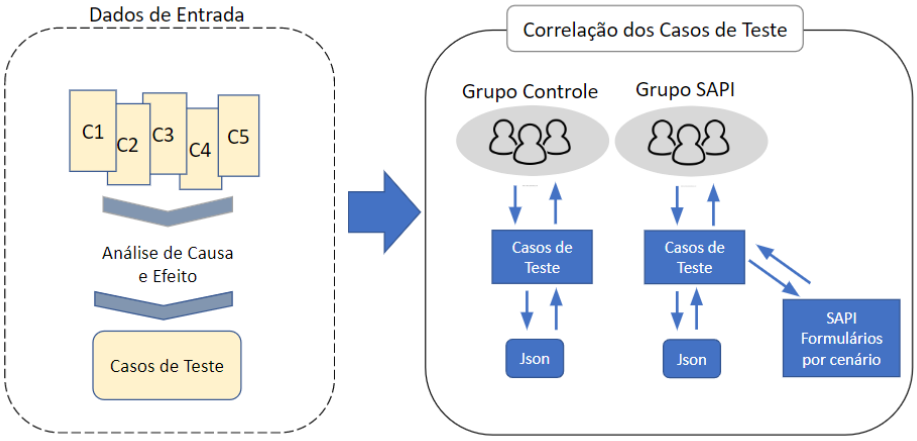

Figura 14: Correlação dos casos de teste.

4.2.7.2 Categorização dos *feedbacks*

A análise dos dados coletados em 4.3 foi realizada qualitativamente e a investigação em profundidade por meio da perspectiva dos participantes. Para poder entender o posicionamento dos participantes, foram mapeadas as respostas comuns para as perguntas, gerando diferentes categorias de significado para a pesquisa. A codificação do material coletado é uma codificação aberta na qual é segmentada por participante, ou seja, esse processo permite analisar, segmentar, extrair, examinar, comparar, conceituar e categorizar os dados (LEITÃO; PRATES, 2017). A técnica de análise de dados foi realizada por meio de uma análise *top-down*, ou *bottom*, já que esse processo permite que o pesquisador categorize e interprete o material obtido de modo que sua reorganização faça emergir novos significados, como definido em (LEITÃO; PRATES, 2017).

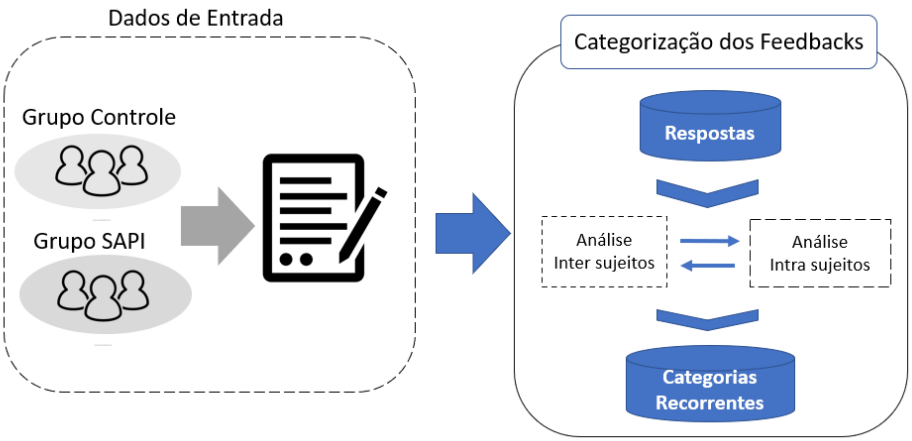

Figura 15: Categorização dos *feedbacks*.

Para a categorização dos *feedbacks* obtidos pelo preenchimento do formulário por cada um dos grupos, como visto na Figura 15, foi utilizada a metodologia do MEDS para analisar as respostas. Todas as respostas obtidas foram analisadas da seguinte forma: primeiramente, uma análise inter-sujeitos e, posteriormente, a análise intra-sujeitos. Ou seja, compararam-se as respostas obtidas por cada participante e em seguida foram verificadas as respostas entre eles. Dessa análise, foi possível a criação das categorias recorrentes encontradas, da qual obtiveram-se tanto os pontos fortes como os pontos fracos da aplicação do *SigniFYIng APIs* no processo tradicional de teste de software.

4.3 Execução do experimento

A realização do experimento foi apenas iniciada após ele ter sofrido algumas modificações identificadas no estudo piloto, como pode ser visto na seção 4.2.3. Com o planejamento do experimento pronto, foram realizados os convites aos participantes e feita a seleção deles para os devidos grupos. Em seguida, foi executado o experimento.

4.3.1 Estudo piloto

Os participantes foram convidados para o estudo, em que foi solicitado, primeiramente, o consentimento para a participação e, em seguida, com a confirmação da participação, foram enviadas as instruções para realizar o estudo piloto. Todo procedimento foi feito on-line. Com a condução do estudo piloto, percebeu-se a necessidade de realizar alterações, além de também ter sido sugerido pelos participantes algumas modificações.

A primeira modificação foi na forma de aplicação do *SigniFYIng APIs*: foi enviado o material para o participante, incluindo vídeos explicativos, porém, percebeu-se que mesmo tendo explicado em detalhes como deveria ser executado o experimento, o participante fez perguntas que estavam presentes no material fornecido, dando a impressão de que o vídeo não foi assistido. Para prevenir esse problema, foi elaborado um conjunto de perguntas simples a serem aplicadas após o participante assistir cada um dos dois vídeos.

A segunda modificação foi em relação ao envio do e-mail contendo muitas informações, que levou o participante a começar pelo que tinha maior interesse, ao invés de seguir a ordem necessária para o experimento. Esse problema foi resolvido mediante a divisão desse e-mail. Inicialmente, continha o link dos vídeos para aprender sobre o *SigniFYIng APIs*; em seguida um formulário para aplicar o *SigniFYIng APIs*; então um arquivo para o participante escrever os casos de testes; e por fim um *feedback* do experimento. Esse

único e-mail foi dividido em 3: sendo o primeiro abordando apenas a explicação sobre o *SigniFYIng APIs*, o segundo com o formulário para aplicar o *SigniFYIng APIs* e o terceiro solicitando a criação de casos de teste com base na etapa anterior e um breve questionário de *feedback* do experimento.

A terceira modificação foi acerca do modelo fornecido para a elaboração dos casos de teste. O participante ficou em dúvida com relação a alguns termos utilizados, como "fluxos alternativos" e "requisitos validados". Essa ameaça foi resolvida com a substituição desses termos por outros mais claros, mantendo, assim, os seguintes campos no relatório de teste a serem preenchidos: etapa/entrada do teste, resultados esperados, resultados reais, passou/falhou.

4.3.2 Seleção dos participantes

A seleção dos participantes nos dois grupos (controle e SAPI) foi determinada tendo em vista o que os participantes relataram nos formulários de experiência. A análise desses formulários foi realizada pela pesquisadora autora dessa dissertação em conjunto com a professora orientadora. As pesquisadoras identificaram, diante da análise das respostas dos participantes, duas categorias: um com experiência profissional e outro com conhecimentos acadêmicos. Ambos possuem conhecimentos na área de teste de software, porém o grupo com experiência profissional contém participantes que trabalham com testes. A realização dessa etapa foi necessária para poder equilibrar os dois grupos formados. Os participantes com conhecimentos acadêmicos foram identificados como P1, P8, P5 e P10. Enquanto os com experiência profissional como P2, P3, P4, P6, P7 e P9. A Tabela 9 mostra em qual grupo cada participante foi alocado. Os participantes do grupo controle são representados do P1 ao P5, enquanto os do grupo SAPI do P6 ao P10.

Grupo controle	**Grupo SAPI**
P1	P6
P2	P7
P3	P8
P4	P9
P5	P10

Tabela 9: Divisão dos Grupos.

Realizada a separação dos grupos, foi considerado se o participante possuía, ou não, conhecimentos nas *APIs* testadas. Na Tabela 10, os participantes que conheciam a *API* estão sinalizados com *Sim* e os que não conheciam com *Não*.

Participante	Perfil	Conhece a *API* Trello	Conhece a *API* Moodle
P1	Possui experiência	Sim	Não
P2	Possui experiência	Não	Não
P3	Acadêmico	Não	Não
P4	Possui experiência	Não	Não
P5	Acadêmico	Não	Sim
P6	Possui experiência	Não	Não
P7	Acadêmico	Não	Não
P8	Acadêmico	Não	Não
P9	Possui experiência	Não	Não
P10	Acadêmico	Não	Não

Tabela 10: Perfil dos participantes.

4.3.3 Preparação dos participantes

Os participantes que fizeram parte do grupo SAPI se prepararam previamente antes de ir para o experimento. Foi instruído que assistissem os vídeos fornecidos e fizessem a leitura do manual rápido do *SigniFYIng APIs*, disponíveis no repositório[11].

4.3.4 Execução do experimento

O experimento foi realizado em dois dias distintos. No primeiro dia, foi realizado apenas com os participantes do grupo SAPI, enquanto no segundo dia, foi feito com o grupo controle. Foram direcionadas aos participantes as instruções do experimento por e-mail e as dúvidas que surgiram, foram tiradas individualmente.

4.4 Considerações do capítulo

Esse capítulo teve como foco apresentar a proposta da pesquisa que é a utilização da ferramenta proveniente da Engenharia Semiótica, *SigniFYIng APIs*, para ser aplicada na elaboração testes funcionais. Como o *SigniFYIng APIs* possibilita encontrar falhas de comunicabilidade, o que foi proposto investigar é se essa ferramenta pode contribuir no aumento da testabilidade, permitindo, assim, que seja desenvolvido melhores casos

[11] https://github.com/camilamorelli/testabilityInAPIs_with_SemioticEngineering#material-grupo-sapi

de testes. A fim de verificar se as falhas que o *SigniFYIng APIs* encontra ajudam na testabilidade do software, foi planejada a aplicação do experimento, como foi descrito nesse capítulo. No próximo capítulo, serão apresentados os resultados obtidos do experimento, assim como uma discussão acerca desses resultados.

5 Resultados e Discussões

Com a execução do experimento descrito previamente na seção 4.3 e a preparação dos dados como descrito na seção 4.2, foi possível responder às questões de pesquisa. Este capítulo aborda o que foi encontrado e as ameaças à validade do experimento, assim como a discussão acerca dos resultados obtidos.

5.1 Questões de pesquisa

As próximas seções estão divididas da seguinte forma: primeiramente, foi feita uma análise de cada uma das quatro questões de pesquisa definidas pelo GQM 4.2. Em seguida, foi elaborada uma discussão dos resultados obtidos. Para a elaboração e análise dos dados, foi considerado o perfil do participante e, também, se ele já conhecia a *API* em questão, como visto na seção 4.3.2.

5.1.1 Análise da Q1: O que foi identificado com a aplicação da ferramenta *SigniFYIng APIs*?

A questão Q1 teve como objetivo identificar, contabilizar e categorizar os dados que se obtêm ao aplicar o *SigniFYIng APIs*, assim como obter uma avaliação subjetiva dos resultados obtidos. Por meio dessa pergunta, foram possíveis identificar as falhas de comunicabilidade presentes em cada um dos cenários. Para mostrar as falhas e uma análise do que foi obtido, foram estabelecidas as seguintes métricas: M1.1 Listar as CDNs encontradas; M1.2 Separar os tipos de falhas em parcial, completa e total; M1.3 Listar as etiquetas de comunicabilidade; e M1.4 Avaliação subjetiva do que foi encontrado com as métricas anteriores.

Cenário	Efeito da Metacomunicação	Quantidade	CDNs	Total de CDNs distintas
C1	E3: Inesperado E5: Mal usado E4: Incompreendido	4 1 2	Consistência - E3, E3 - P9 Expressividade dos papéis - E5 - P10 Visibilidade - E3 -P6 Difusão - E4-P7	4
C2	E1: Sucesso E5: Mal usado E2: Recusa E4: Incompreendido E3: Inesperado E7: Esperado	2 2 1 1 2 1	Propensão a erros - E1 (P8), E4 (P7), E5 (P10), E5(P7), E3(P10) Consistência - E2(P9) Nível de Abstração - E3(P9)	3
C3	E1: Sucesso E3: Inesperado E2: Recusa	2 1 1	Visibilidade - E2(P6) Propensão a erros - E1(P8) Não observado - E3 - apesar de ter visto um efeito, não associou ele a uma CDN (P10), E1 (P9)	2
C4	E1: Sucesso E2: Recusa	3 1	Difusão - E2(P6) Visibilidade - E1(P9) Não observado - E1(P8), E1(P10)	2
C5	E6: Falha E5: Mal usado E1: Sucesso E3: Inesperado E8: Sucesso E4:Incompreendido	2 2 1 1 1 1	Operações mentais difíceis - E4(P6), E8(P8) Nível de abstração - E6(P7), E5(P9) Consistência - E6(P6) Difusão - E5(P9) Expressividade dos papéis - E1(P10)	5

Tabela 11: CDNs distintas.

Cenário	Etiqueta de comunicabilidade	Quantidade	Total de tipos de falhas	Total de Etiquetas distintas
C1	T6: O que aconteceu ? - P6,P9 T1: Desisto - P7,P9 T4: Vai de outro jeito - P6,P7 T9: Epa! - P8 T13: Porque não funciona?-P10 T3: Não,obrigado -P7	2 2 2 1 1 1	Temporária - 4 Total - 2 Parcial - 3	6
C2	T6: O que aconteceu?- P8,P10 T4: Vai de outro jeito - P9 T5: Cadê? - P7 T13: Por que não funciona? - P9 T1: Desisto. - P10 T3: Não, obrigada. - P9	2 1 1 1 1 1	Temporária - 4 Total - 2 Parcial - 2	6
C3	T6: O que aconteceu? - P6,P8 T4: Vai de outro jeito - P10	3 1	Temporária - 3 Parcial - 1	2
C4	T5: Cadê? - P8 T6: O que aconteceu?- P9 T3: Não, obrigado -P6	1 1 1	Temporária - 1 Parcial - 1	2
C5	T1: Desisto - P6, P8, P9 T2: Parece bom para mim - P7 T13: Por que não funciona? - P6 T3: Não, obrigado - P9	3 1 1 1	Temporária -1 Total - 4 Parcial - 1	4

Tabela 12: Etiquetas distintas.

A Tabela 11 foi elaborada a partir das informações dos resultados[1] obtidos dos

[1] https://github.com/camilamorelli/testabilityInAPIs_with_SemioticEngineering

participantes. Nela, são mostrados os **efeitos da metacomunicação** e a quantidade de vezes que o mesmo efeito foi observado em cada um dos cenários. Na coluna de CDNs, é relacionado cada um dos efeitos de metacomunicação com suas respectivas CDNs. Já na última coluna, se tem o número de CDNs distintas vistas em cada cenário.

A Tabela 12 relaciona as **etiquetas de comunicabilidade** identificadas pelo id, as quantidades encontradas de cada etiqueta, o total de tipos de falhas separadas em **falhas totais**, **temporárias** e **parciais**, e a última coluna exibe o total de etiquetas distintas encontradas.

Com os resultados exibidos nas Tabelas 11 e 12, pode-se observar o que se obteve em cada um dos cenários. Nas sessões seguintes é feita uma análise para cada um dos cenários.

5.1.1.1 Análise do cenário 1

Analisando o cenário 1 do Trello, em que era necessário informar o nome do cartão e a posição em que ele deveria ficar e, ao observar a Tabela 11, percebe-se que os participantes do grupo SAPI encontram os seguintes efeitos da metacomunicação: inesperado, mal usado e incompreendido. Esses efeitos refletem a situação que os participantes passaram ao utilizar a *API*, em que eles tentaram colocar o cartão em uma determinada posição e o comando não era executado conforme solicitado. Observa-se que nenhum dos participantes teve sucesso na inclusão do cartão na posição correta. No cenário 1, foram identificadas as CDNs de *consistência*, *expressividade dos papéis*, *visibilidade* e *difusão*. Em outro exemplo, o P9 encontrou uma situação **inesperada** caracterizada pela etiqueta de comunicabilidade "O que aconteceu?", na qual o participante relacionou a CDN de *consistência*. Encontrar a CDN de *consistência* reflete que a semântica da *API* não foi expressa corretamente, como relatado pelo P9:

> *"Inseri um card nomeado "!@#$%^&*"e tive como retorno "!@", com meu novo card sendo nomeado com o mesmo nome."*

A CDN de *expressividade dos papéis*, também encontrada nesse cenário 1, demonstra que o propósito da entidade não é prontamente inferido. O P10, por exemplo, relatou que tentou colocar o cartão na posição escolhida e diante de alguns testes conseguiu colocar, porém não conseguiu entender a lógica de como funciona o sistema. Encontrando o efeito de **mal usado** caracterizada pela etiqueta "Porque não funciona?". Segue o trecho do participante P10:

> *"Ao tentar executar o cenário na primeira vez, notei que a informação disponibilizada pela API sobre o parâmetro "pos"é insuficiente[...] Após algumas tentativas, usando diferentes valores, consegui colocar o card na posição que foi solicitada. Entretanto, continuo sem entender qual é a lógica do parâmetro [...]"*

A CDN de *visibilidade*, também observada nesse cenário, é relativa à se o sistema esconde as informações de forma que diminui, ou reduz, a visibilidade delas. Nesse caso, o participante P6 relacionou à CDN de *visibilidade* diante do relato de que caso o cenário fosse executado mais de uma vez era possível incluir na posição esperada. Assim foi encontrado o efeito de **inesperado**, caracterizado pela etiqueta "Vai de outro jeito". Segue o relato do participante:

> *"No teste anterior, foi informado que o valor do parâmetro "pos"é igual a 2 e, ao submeter a chamada, o sistema incluiu o card na primeira posição. Caso o cenário seja executado mais de uma vez, o sistema inclui o card na posição esperada pelo usuário."*

E por fim, a CDN de *difusão* é aquela que reflete se a *API* apresenta verbosidade de linguagem, ou se é prolixa e mal distribuída. Assim foi encontrado o efeito de incompreendido, caracterizado pela etiqueta "Vai de outro jeito". Um exemplo dessa situação é relatado pelo participante P7, como no trecho a seguir:

> "Após inserir um novo cartão, foi retornada uma quantidade considerável de código que não informa se a inserção funcionou, ou não. Isto também ocorre para os casos de erro em que a descrição de que houve um erro não está clara."

5.1.1.2 Análise do cenário 2

O cenário 2 do Trello tinha como propósito a criação de um cartão com uma data de entrega. Nesse cenário, ao ser executado pelos participantes, observaram-se 6 efeitos da metacomunicação: sucesso, mal usado, recusa, incompreendido, inesperado e esperado. Nesse cenário, obtiveram-se as CDNs de *propensão a erros*, *consistência* e *nível de abstração*. É interessante notar que, por exemplo, o participante P8 relatou um efeito da metacomunicação de **sucesso** e relacionou a CDN de *propensão a erros*. No dado coletado do participante, observou-se que ele identificou que a *API* leva a uma propensão de erro,

porém o participante conseguiu compreender o problema e acabou relacionando um efeito de sucesso. Ficando assim, a CDN de *propensão a erros* relacionada com um efeito de sucesso. Seguem os trechos do participante P8:

> *"A intenção não menciona o formato da data [...] Com alguns fracassos, entretanto, não é difícil inferir possuindo um mínimo de experiência."*

Já a *propensão de erro* está relacionada ao P8 que segue devido ao seguinte relato:

> *"Não saber o formato da data fica claro ao ler a documentação.[...] A data inserira foi 1 de maio de 2022, meia-noite, UTC. O Trello exibe 20 de abril de 2022, 21h, Brasília [...]"*

Como exemplo para a CDN de *consistência* encontrada, tem o relato do participante P9 que obteve o efeito de metacomunicação de **recusa** devido à resposta não ser exibida como o esperado. Como consequência, ele marcou a etiqueta de comunicabilidade "Vai de outro jeito". Segue o relato:

> *"Digitei 11/02/00, na expectativa de mostrar a data 11 de fevereiro (2) de 2000, e fui respondido com 2 de novembro (11) de 2000".*

Outro fato observado pelo P9 nesse cenário foi o efeito de **inesperado** devido à seguinte situação relatada pelo participante: "Não soube qual posição é a inicial, 1 ou 0, tive de testar para descobrir". Diante de não saber qual a posição inicial e ter que testar para descobrir, o participante relacionou a CDN de *nível de abstração*, levando o participante a questionar "Por que não funciona?".

Em relação aos demais participantes que também observaram a CDN de *propensão a erros*, é interessante notar que, apesar da CDN ser a mesma, a Tabela 12 mostra como cada um dos participantes reagiram diante da análise feita por meio das etiquetas de comunicabilidade. Por exemplo, o participante P7 encontrou um efeito de metacomunicação **incompreendido**, relacionando com a etiqueta de comunicabilidade de "Cadê?", mostrando que o participante não conseguiu compreender o que aconteceu e não encontrou uma solução. Essas etiquetas permitem capturar a reação das pessoas no momento que a utilizam.

5.1.1.3 Análise do cenário 3

O cenário 3 do Trello tinha como objetivo desativar as notificações de um cartão. Nesse cenário, ao ser executado pelos participantes, observaram-se três efeitos da metacomunicação: sucesso, inesperado e recusa. As CDNs observadas foram *visibilidade* e *propensão a erros*. Os participantes P9 e P10 não observaram nenhum efeito, visto que tiveram sucesso ao aplicar o cenário e consequentemente não tiveram etiquetas de comunicabilidade associadas.

O participante P6 teve o efeito de **recusa** diante da não alteração do campo *subscribed*, relacionando à etiqueta "O que aconteceu?". Diante dessa situação, o participante associou o efeito à CDN *visibilidade* por conta de o sistema estar escondendo as informações, isso devido a ele ter percebido que apenas fazendo outra chamada à *API* que ela funcionava. Segue o relato do participante:

> *"Embora o nome do card tenha sido alterado, o sistema não alterou o valor do campo subscribed. O campo só sofre alteração caso o usuário envie a chamada novamente."*

O participante P8 apresentou um efeito de **sucesso** devido ao participante ter conseguido realizar a alteração, porém vislumbrou um efeito, no qual relacionou à CDN *propensão a erros*, caracterizado pela etiqueta "O que aconteceu?". Segue o que foi encontrado pelo participante:

> *"Segundo a API, a propriedade /subscribed/ foi alterada pra falsa. O Trello, entretanto, não exibe claramente o valor da propriedade /subscribed/. Isso não era esperado.[...] Alterando o nome do cartão, entretanto, o usuário conclui que de fato consegue alterar a propriedade /subscribed/. O problema não parece ser da API, entretanto. A API expõe suficiente evidência de que a intenção e o efeito estão dentro do esperado.[...]"*

5.1.1.4 Análise do cenário 4

O cenário 4 do Moodle tinha como objetivo encontrar um fórum de discussão, para isso era necessário escolher o fórum a ser acessado e depois acessar uma discussão desse fórum. Nesse cenário, os participantes encontraram efeitos da metacomunicação de **sucesso** e **recusa**. As CDNs encontradas foram *difusão* e *visibilidade*. Os participantes que

não observaram CDNs, como por exemplo P8 e P10, encontraram o efeito de sucesso, conseguindo realizar o que foi pedido sem problemas. Os participantes que tiveram o efeito de metacomunicação relacionados à **recusa** observaram dificuldades na visibilidade das informações e difusão do conteúdo.

O participante P6, por exemplo, encontrou o efeito de metacomunicação de **recusa**, pois apesar do participante ter compreendido a *API*, o mesmo rejeitou o recurso, relacionando à etiqueta de comunibilidade "Não, obrigada". Nessa situação, o participante relacionou à CDN *difusão* devido à linguagem utilizada não ser adequada. Segue a análise feita pelo participante P6:

> *"Após realizar a sequência de chamadas, foi possível identificar o ID da discussão. Entretanto, seria prudente criar uma nova chamada que realizasse a consulta de uma discussão a partir do título da mesma."*

O participante P9 observou o efeito de **sucesso** e relacionou à etiqueta de "O que aconteceu?". Ele encontrou sucesso no cenário, porém relacionou a análise à CDN *visibilidade*, pois o sistema não fornecia, de forma clara, as informações. Segue o trecho do participante P9:

> *"Tive que procurar manualmente pelo resultado, se o usuário está somente usando a API, ele terá que manualmente ler"*

5.1.1.5 Análise do cenário 5

O cenário 5 do Moodle tinha como objetivo criar um comentário em uma discussão. Nesse cenário, os participantes encontraram os seguintes efeitos da metacomunicação: falha, mal usado, sucesso, inesperado e incompreendido. Nesse cenário, não existia um *bug* e a documentação fornecida, assim como as respostas das chamadas à *API*, não deixavam claras as informações dos dados retornados. O participante P10 conseguiu realizar com sucesso o que foi proposto, mas o P8, apesar de ter tido sucesso conforme a Tabela 2, não aceitou o recurso provido pela *API*. Ou seja, por ele ter encontrado *operações mentais difíceis*, optou pela etiqueta de comunicabilidade "Desisto", desistindo por não compreender a *API*. Segue a análise do participante:

> *"A documentação é tão pobre que o usuário conclui que não vale o esforço mental que certamente decorreria"*

Já o participante P7 observou o efeito de metacomunicação de *falha* por não ter conseguido identificar as variáveis que deveriam ser utilizadas. Ele relacionou à CDN *nível de abstração* e à etiqueta de "Parece bom para mim", na qual o usuário está inconsciente do que aconteceu. Segue o trecho do participante:

> *"Para tentar executar este cenário, eu voltei nos resultados do cenário 4 para tentar identificar as variáveis, porém sem sucesso."*

O participante P6 observou o efeito de metacomunicação de **falha**, associando à etiqueta "Desisto", por não conseguir entender qual é o parâmetro que apresentou erro, e à CDN *consistência*. Sua análise pode ser vista no seguinte trecho:

> *"Ao tentar enviar uma requisição através da função 'core comment add comments', a resposta da chamada informa que o valor do parâmetro é inválido, mas não diz qual parâmetro que apresentou a inconsistência."*

O participante P9 encontrou o efeito de metacomunicação de **mal usado** por ter se confundido em qual *API* ele deveria utilizar e associou à etiqueta de "Desisto". A CDN relacionada foi a *difusão*. Segue a análise do participante:

> *"Tive dificuldade em encontrar o endpoint correto, sua nomenclatura não fazia sentido e, para piorar, a nomenclatura de outro endpoint parecia cumprir com o propósito desse."*

5.1.1.6 Análise da aplicação do *SigniFYIng APIs* nos cenários

Visto o que foi encontrado em cada um dos cenários, observa-se que a quantidade e a variedade de CDNs diferem de um cenário para outro. Os cenários que foram encontradas mais falhas de comunicabilidade foram os cenários 5, 1 e 2, nessa sequência.

Um ponto relevante a ser considerado é o fato de que nesses cincos cenários mostrados, nenhum dos participantes tinham conhecimento prévio da *API*, como visto na Tabela 10, e os participantes que tinham experiência profissional na área eram P6 e P9. Apesar dos participantes P6 e P9 terem experiência, não foi possível observar, ou notar, algum padrão das respostas obtidas quando comparados com os demais participantes que tinham apenas conhecimentos acadêmicos.

Analisando os dados da Tabela 11, percebe-se que ao aplicar o *SigniFYIng APIs*, os cenários que tiveram as duas maiores quantidades de **CDNs distintas** foi o cenário 5 e

o cenário 1. No cenário 5, os participantes encontram no total cinco CDNs: *operações mentais difíceis, nível de abstração, consistência, difusão, expressividade dos papéis*. Já no cenário 1 foram encontradas quatro CDNs: *consistência, expressividade dos papéis, visibilidade, difusão*. O cenário que teve mais etiquetas de comunicabilidade distintas foi o cenário 1 e o cenário 2. Os cenários que foram identificadas falhas mais graves, foi, primeiramente, o cenário 5 com 4 **falhas totais**. O outro cenário foi o cenário 1, o qual obteve 2 falhas, sendo que ambas foram **falhas totais**.

Visto o que foi obtido e demostrado, pode-se observar que, ao aplicar o *SigniFYIng APIs*, foi possível identificar que os cinco cenários escolhidos possuem falhas de comunicabilidade, sendo que o cenário que teve maior quantidade observada foi o cenário 5. Além disso, identificaram-se os cenários que tiveram falhas de comunicabilidade graves, ou seja, falhas totais que são falhas das quais o usuário da *API* não entende o porquê de a *API* estar se comportando dessa forma. As falhas totais foram respectivamente no cenário 5, cenário 2 e cenário 1.

Além de ter identificado quais foram as CDNs, efeitos de metacomunicação e etiquetas de comunicabilidade em cada um dos cenários, foi feita uma análise com os relatos dos efeitos e das falhas obtidas durante a aplicação do *SigniFYIng APIs*.

5.1.1.7 Análise dos comentários do grupo SAPI

Ao analisar os efeitos e as falhas descritas por cada participante na Tabela 13, foram possíveis de identificar categorias de situações semelhantes. Esses efeitos são as análises que cada participante fez sobre o cenário, como, por exemplo, o participante P9 relatou o seguinte efeito no cenário 1:

> *"O designer espera que o usuário saiba o significado do parâmetro de retorno 'pos'. Mesmo ele tendo o mesmo nome do parâmetro de envio e um significado completamente diferente".*

Esse comentário foi agrupado na categoria O1, visto na Tabela 13, pois ela reflete a situação relatada pelo P9, em que **"A documentação não deixa claro como funciona o sistema e suas regras de negócio"**.

Após analisar cada efeito, como demonstrado pelo participante P9, foram elaboradas categorias para os efeitos semelhantes, descritos na Tabela 13, com os IDs de O1 à O7. Essas situações revelam os principais pontos observados em cada um dos cenários e, na

coluna participantes, é possível verificar qual participante identificou determinada situação. Por exemplo, o participante P6, diante do cenário 5, relatou o seguinte:

> *"Os parâmetros instanceid e itemid da função core comment add comments podem ser facilmente confundidos. Além disso, os parâmetros component e content também são semelhantes e possuem a mesma descrição na documentação.".*

Esse comentário foi relacionado à situação **O2: Nomenclatura poderia ser mais compreensível**, agrupando comentários em que os nomes das variáveis e os nomes das chamadas da *API* não apresentavam de modo claro e compreensível para os participantes. Essa situação foi identificada nos cenários C1, C3 e C5. Para as outras situações identificadas, foi feito o mesmo procedimento, sempre relacionando o comentário do participante com a categoria identificada.

A situação **O3: A resposta da *API* não deixou claro se funcionou, ou se deu erro**, agrupa comentários nos quais foram identificados que a *API* não deixou claro se a requisição funcionou, isso ao realizar uma requisição para a *API* e validar o seu retorno diante da visualização do cenário diretamente no Trello. Segue o comentário do participante P7 no cenário 1:

> *"...Não ficou muito claro, olhando a documentação, qual seria o retorno do sistema para informar que a inserção do cartão funcionou. O texto que é retornado não diz se funcionou, ou se deu erro. O usuário precisa clicar em lista para saber se a ação ocorreu de forma esperada.".*

A situação **O4: A documentação não informa quais caracteres são aceitos**, agrupou comentários em que a *API* não deixa claro se aceita número, *string*, ou caracteres especiais. O participante P9, diante do cenário 1, fez a seguinte observação:

> *"Nem todos os caracteres do teclado são aceitos, isso causa uma alteração forçada do nome do card. Falta sinalização dos caracteres restritos pelo designer."*

A situação **O5: O formato esperado do parâmetro não estava bem definido**, agrupou comentários que refletem que apesar de a documentação ter definido o parâmetro, este deixou vago o formato que seria aceito pela *API*. Segue o comentário do P10 diante do cenário 2:

> *"Baseado na iteração anterior, após realizar alguns testes com a API, consegui entender qual é o formato correto para determinar a data de entrega de um card. Entretanto, as pós-condições do uso da API ainda não estão claras o suficiente. Ou seja, por que a data informada na requisição está diferente da data do resultado?"*

A situação **O6: A documentação não detalha os conceitos do domínio da aplicação**, agrupa os comentários que identificaram que os termos utilizados na *API* não são detalhados na documentação, exigindo que o usuário entenda como funciona a *API*. Comentário do participante P8 no cenário 5:

> *"...não avisa ao usuário que ele precisa conhecer a organização e nomenclatura interna do sistema. A documentação não explica termos, como "contextLevel", e espera que o usuário saiba exatamente cada uma das definições envolvidas. Nem exemplos são mostrados. A documentação, portanto, serve aos programadores do sistema e não a um usuário externo. A documentação é quase inútil ao aventureiro iniciante".*

A situação **O7: A complexidade para obter um dado específico da *API***, reflete a necessidade de realizar mais de uma requisição em *APIs* diferentes para um dado específico, sendo que poderia ter uma forma mais objetiva. Comentário do participante P6 diante do cenário 4:

> *"... Para encontrar o ID de uma determinada discussão, será necessário buscar todos os cursos ativos através da função "core course getcourses". Em seguida, deve-se buscar todos os fóruns, informando como parâmetro o ID do curso, que foi recuperado na resposta da chamada anterior. Dado que o sistema retornou todos os fóruns de um curso, será necessário realizar outra chamada com a função "mod forum get forum discussions"que terá como parâmetro o ID do Fórum. A resposta desta requisição retornará todas as discussões deste fórum, onde o usuário poderá identificar o ID do fórum desejado após encontrar o registro que possui o campo name: Discussão para Aplicação de Testes".*

ID	Situações observadas	Cenários	Participantes
O1	A documentação não deixa claro como funciona o sistema e suas regras de negócio	C1 C2 C3 C5	P9, P10 P9, P10 P10 P6, P8
O2	A nomenclatura poderia ser mais compreensível	C1 C3 C5	P6 P6 P6, P7, P8, P9
O3	A resposta da API não deixou claro se funcionou, ou se deu erro	C1 C3	P7 P8
O4	A documentação não informa quais caracteres são aceitos	C1	P7, P9
O5	O formato esperado do parâmetro não estava bem definido	C2	P7, P8, P10
O6	A documentação não detalha os conceitos do domínio da aplicação	C5	P7, P8, P9, P10
O7	A complexidade para obter um dado específico da API	C4	P6, P9

Tabela 13: Situações observadas

Com o intuito de entender o que foi identificado com a aplicação do *SigniFYIng APIs* diante da elaboração dos casos de teste, foram analisadas as situações observadas na Tabela 13. As situações identificadas foram analisadas juntamente com o documento no qual foi escrito o caso de teste.

As Tabelas 14 e 15 possuem cada uma das situações observadas, os cenários que foram vistos, o participante e suas respectivas CDNs, etiquetas e efeito de metacomunicação. Na última coluna tem a situação, nela, foram listadas quatro situações identificadas: **Possibilitou criar caso de teste**, **Possibilitou criar caso de teste, porém não criou**, **Melhoria**, **Efeito não relacionado à falha** e **Inconclusivo**. A situação **Possibilitou criar caso de teste** representa que o participante, diante do que foi observado, possibilitou criar um caso de teste. Em **Possibilitou criar caso de teste, porém não criou**, refletem que em situações em que o participante obteve dados para elaborar um caso de teste pelo *SigniFYIng APIs*, ao elaborar o caso de teste, não relatou a falha. A situação de **Melhoria** levou o participante a identificar apenas como a *API* poderia melhorar. Na situação **Inconclusivo**, apresenta que, apesar do participante identificar ter algum *bug*, ele não chegou a nenhuma conclusão, pois o efeito não ajudou a esclarecer a falha. Já em **Efeito não relacionado à falha**, foram situações que o participante relatou um efeito que não era relacionado à falha, logo o caso de teste elaborado não pode ser associado ao efeito descrito.

O1: A documentação não deixa claro como funciona o sistema e suas regras de negócio

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C1	P9	Consistência	T1	E3	Possibilitou criar casos de teste
C2	P10	Expressividade dos papéis	T13	E5	Possibilitou criar casos de teste
C3	P9	Nível de Abstração	T13	E3	Melhoria
	P9	Consistência	T3	E7	Possibilitou criar casos de teste
	P10	Propensão a erros	T6	E5	Possibilitou criar casos de teste
C3	P10	Não teve	T4	E3	Inconclusivo
C5	P6	Consistência Operações mentais difíceis	T6	E6 E4	Melhoria

O2: Nomenclatura poderia ser mais compreensível

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C1	P6	Visibilidade *	T4	E3	Efeito não relacionado à falha
C3	P6	Visibilidade*	T6	E2	Efeito não relacionado à falha
C5	P6	Consistência Operações mentais difíceis	T1 T13	E6 E4	Melhoria
	P7	Nivel de Abstração	T2	E6	Melhoria
	P8	Operações mentais difícies	T1	E8	Melhoria
	P9	Difusão	T1	E5	Melhoria

O3: A resposta da API não deixou claro se funcionou, ou se deu erro

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C1	P7	Difusão	T4	E4	Melhoria
C3	P8	Propensão a erros	T6	E1	Inconclusivo

Tabela 14: Consolidação dos dados - pt.1.

O4: Documentação não informa quais caracteres são aceitos

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C1	P7	Não observado	T1 T3	E4 E4	Melhoria
	P9	Consistência	T6	E3	Possibilitou criar casos de teste, porém não foi criado

O5: O formato esperado do parâmetro não estava bem definido

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C2	P7	Propensão a erros	T5	E4	Inconclusivo
	P8	Propensão a erros	T6	E1	Melhoria
	P9	Consistencia	T4	E2	Possibilitou criar casos de teste
	P10	Propensão a erros	T1	E3	Possibilitou criar casos de teste
C5	P7	Nível de abstração	T2	E6	Melhoria

O6: A documentação não detalha os conceitos do domínio da aplicação

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C5	P8	Operações mentais difíceis	T1	E8	Melhoria
	P9	Nível de abstração	T3	E5	Melhoria
	P10	Expressividade dos papéis	-	E1	Melhoria

O7: Complexidade para obter um dado especifico da API

Cenário	Participante	CDN	Etiqueta	Efeito	Situação
C4	P6	Difusão	T3	E2	Melhoria
	P9	Visibilidade	T6	E1	Melhoria

Tabela 15: Consolidação dos dados - pt.2.

ID	Depoimento sobre a API do Moodle
T1	*"A API do Moodle é um pouco mais complexa, no sentido de que era preciso saber entender e interpretar os resultados das requests das funções para saber se está retornando certo ou não, e não somente verificando visualmente." - P1*
T2	*"A documentação ...ficou um pouco confusa na minha concepção.[...] principalmente na parte de comentário, na documentação tinha um excesso de parâmetros a serem pedidos e nem todos eram claros sobre ao valor que necessitava." - P4*

Tabela 16: Trechos de depoimentos sobre a *API* do Moodle.

ID	Depoimento sobre a API do Trello
T1	*"... a do trello está muito fraca. Pode ajudar descrevendo os tipos e colocando exemplos de resultado" - P2*
T2	*Create a new Card - a documentação é bem confusa e gastei um tempo pra entender que precisava explicitar o ".0"* *Create a new Card com data - ainda muito confusa, não diz o formato da data, e a hora está em um fuso horário não explicitado também* *Desativar as notificações - Essa foi mais confusa ainda, não diz que o nome precisa ser o mesmo, e não cumpre o que está explicitado no documento. - P3*

Tabela 17: Trechos de depoimentos sobre a *API* do Trello.

5.1.1.8 Análise dos comentários do grupo controle

Vale ressaltar que o grupo controle também reuniu *feedbacks* das *APIs* testadas. No fim do experimento, foi solicitado o preenchimento de um questionário para obter as impressões do participante em relação às *APIs* testadas. As perguntas podem ser vistas na seção 4.2.6. As Tabelas 16 e 17 reúnem os trechos ditos pelos participantes sobre as *APIs*. Em relação às *APIs* do Trello, os participante P2 e P3 relataram algumas das dificuldades encontradas e sugeriram pontos de melhorias. O participante P2, no trecho T1, relatou que a *API* é

"fraca" e sugeriu como melhoria a exibição de exemplos dos resultados das chamadas do Trello. Já o P3 no trecho T2, mostra a dificuldade encontrada em cada um dos cenários, por exemplo, no cenário 1 *"create a new card"*, ele demorou para compreender que para informar a posição do card era necessário explicitar a posição com ponto.

Os participantes P4 e P5 relataram que não tiveram problemas na *API*. O participante P4 relatou que:

> *"A documentação do Trello estava extremamente clara, mostrava o que esperava e o que iria ter como resultado."*.

Já o participante P5, disse:

> *"... mesmo sendo diferentes as documentações, em ambas as APIs, conseguiu visualizar o que cada variável esperaria como entrada."*

Nas *APIs* do Moodle, os participantes P1 e P4 relataram pontos a serem melhorados, como por exemplo o P1, diante do trecho T1, diz que o nome das funções poderiam ser mais sugestivas. O participante P4 já relatou no T2 sobre o excesso de parâmetros e que nem todos eram claros. O participante P2 não informou nada sobre o uso das *APIs*. Enquanto isso, os participantes P3 e P5 disseram que a documentação era boa e que não tiveram problemas. O participante P5 disse que não teve problemas com a *API* do Trello. Já o participante P3 relatou o seguinte:

> *"... A documentação dessa é bem melhor, o único ponto confuso é que diz, na do fórum, que o id do curso é opcional, mas ao tentar não fornecer dá um erro. Tirando isso, ela explica bem o formato das coisas. Fazer comentário - de todas, essa é a mais bem explicada, cada uma das funções documentadas explica bem o que precisa ser feito e qual o resultado esperado"*.

É interessante notar que os participantes de ambos os grupos perceberam que as *APIs* precisavam de melhorias. O grupo controle ficou um pouco divido quanto às *APIs*, por exemplo, enquanto alguns participantes disseram que a *API* era muito boa, outros disseram o contrário. Em relação à reflexão do uso das *APIs*, foram perceptíveis as diferenças em relação ao grupo SAPI. A principal diferença notada foi que o grupo SAPI, diante do uso da ferramenta, possibilitou obter maiores reflexões sobre as situações de melhorias. Pode-se notar, também, que, até mesmo os participantes que conseguiram executar sem

problemas, relataram em quais pontos deveriam ser feitas melhorias, como por exemplo o participante P10 diante dos cenário 4 e 5.

Em resposta à Q1: O que foi identificado com a aplicação da ferramenta *SigniFYIng APIs*?

A ferramenta permitiu coletar informações, como os efeitos da metacomunicação, as CDNs e as etiquetas de comunicabilidade de cada cenário testado pelos participantes. Essas informações permitiram entender em quais pontos as *APIs* demonstram fraquezas e possibilidades de melhorias. Além disso, percebeu-se que ao aplicar o *SigniFYIng APIs* em alguns casos os participantes conseguiram elaborar os casos de testes, enquanto em outros casos direcionaram à uma situação de melhoria. Já as situações que o participante identificou que era necessário fazer algo, mas não conseguiu concluir, ou até mesmo entender o que deveria ser feito, foram consideradas como **inconclusivas**.

5.1.2 Análise da Q2: De que forma *SigniFYIng APIs* sobrecarrega e de que forma dificulta o processo tradicional de teste?

A questão Q2 teve como intenção explorar em quais situações do processo tradicional de testes, a utilização da ferramenta *SigniFYIng APIs* traz dificuldades e/ou sobrecarga. Para responder a essa pergunta, além de uma análise subjetiva do que os participantes do *SigniFYIng APIs* relataram, também foi comparado o tempo gasto pelo grupo controle e *SigniFYIng APIs*. Para isso, foram levadas em considerações as seguintes métricas: M2.1 Avaliação subjetiva sobre como foi a utilização e a aplicação da ferramenta; M2.2 Tempo total gasto pelo grupo SAPI em aplicar a ferramenta e elaborar os casos de teste; M2.3 Tempo total gasto pelo grupo controle.

Para identificar quais foram os pontos que sobrecarregam e dificultam, foi feita uma análise intra e inter-participantes para identificar as categorias de significado recorrentes para cada um desses pontos. Essas informações foram extraídas do formulário de avaliação da experiência do experimento visto na seção 4.3.

A categoria **dificuldade** agrupa subcategorias e códigos de significado que exprimem pontos percebidos pelos participantes que podem ser caracterizados como dificuldades para a utilização da ferramenta *SigniFYIng APIs* em um processo de elaboração de testes.

Foram identificadas três subcategorias que dificultam a aplicação: **(1) Necessidade de treinamento**; **(2) Curva de aprendizado acentuada no início**; **(3) Falta de entendimento durante a aplicação da ferramenta**. É possível perceber uma relação

entre essas categorias, pois todas as três, de alguma forma, expressam a dificuldade na utilização do *SigniFYIng APIs*. A subcategoria **Necessidade de treinamento** aponta que a sua utilização precisa de um treinamento prévio o qual o mesmo depende de ensinar alguns conhecimentos originários da Engenharia Semiótica dos quais a ferramenta utiliza. Conforme exemplificado pelo trecho T1 da Tabela 18.

ID	Depoimento
T1	*"[..] é preciso de treinamento prévio para utilizar." - P7*
T2	*"O método é complexo de aprender. . . . . " - P7*
T3	*"O ponto negativo que se pode vislumbrar será compreender o modus operandi da ferramenta." - P8*
T4	*". . . foi um pouco complicado, se não fosse pela assistência contínua, possivelmente eu não entregaria o resultado esperado." - P9*
T5	*"Tive dificuldade para aprender o método no início, ..." - P6*
T6	*"... não estava muito claro como seria possível escrever o frame sem ter visto a documentação da API antes" - P6*
T7	*"Não achei muito fácil... senti dificuldade no processo de aplicação do mesmo" - P10*
T8	*"Muitas vezes,... me senti perdido sobre o que deveria fazer, ou qual era o meu papel em sua aplicação." - P10*
T9	*".....e confesso que estou com a sensação de não ter aprendido." - P7*

Tabela 18: Trechos de depoimentos referentes às dificuldades percebidas.

Em relação à **Curva de aprendizado acentuada no início**, foram exprimidas as dificuldades dos participantes em compreender e saber aplicar a ferramenta mediante aos conceitos envolvidos da área de Engenharia Semiótica. Notou-se que os participantes acharam a ferramenta de aprendizado complexa, o que pode ser visto nos trechos T2, T3, T4 da Tabela 18.

Diante da dificuldade no aprendizado da ferramenta, percebe-se que a subcategoria **Curva de aprendizado acentuada no início** levou os participantes a terem dificuldades, também, na forma de aplicação da ferramenta. Assim identificando a subcategoria **Falta de entendimento durante a aplicação da ferramenta**, em que os participantes relatam que a aplicação da metodologia, a princípio, não é simples e que mesmo tendo aprendido os conceitos, se sentiram perdidos e até mesmo sem saber o seu papel durante a

aplicação. Esses depoimentos podem ser vistos pelos trechos T5, T6, T7 e T9 da Tabela 18.

Já a categoria **sobrecarga** agrupa a subcategoria e o código de significado que impõe uma carga excessiva na utilização da ferramenta durante o processo de teste. Nela, foi identificada a subcategoria: **(4) Necessidade de simplificação da ferramenta**.

A **Necessidade de simplificação da ferramenta** seria a modificação da ferramenta *SigniFYIng APIs* para se tornar mais aplicável e atraente para os usuários. De fato a ferramenta envolve etapas como a criação do *frame* de metacomunicação, identificar CDNs, efeitos de metacomunicação e associação das etiquetas de comunicabilidade, apesar de ser dividida apenas em três etapas: intenção, efeito e falha, como descrito no Capítulo 2, pois a aplicação dela envolve muita análise por parte dos participantes. Outro ponto que envolve a necessidade da simplificação da ferramenta, seria pelo fato de que a utilização dela pode deixar o processo de teste mais burocrático. Esses depoimentos podem ser vistos nos trechos T10, T11, T12 da Tabela 19.

ID	Depoimento
T10	*"Para aplicar o SAPI na pratica, precisa ser mais simples e com menos opções de escolha para o usuário" - P7*
T11	*"Um típico programador de computadores dificilmente se entusiasmaria de aprender a usar o SAPI. Um típico programador gosta de trabalhar com máquinas e sistemas e não com texto, relatórios e análises minuciosas de consistência e precisão." - P8*
T12	*"[...] eu enxergo que o seu uso pode "burocratizar"um pouco o processo de análise da qualidade da API" - P10*

Tabela 19: Trechos de depoimentos referentes a sobrecarga.

Em relação ao tempo gasto pelos participantes durante o experimento, nos gráficos 16 e 17, são mostrados os tempos totais gastos dos participantes em cada um dos cenários. O valor do tempo é mostrado em minutos.

No cenário 1, nota-se que os participantes do grupo controle P2 e o P4 foram os que gastaram menos tempo. Já os demais P1, P3 e P5 mantiveram o tempo em torno de 45 a 50 minutos. Já no grupo SAPI, os participantes que gastaram menos tempo foram o P8 e o P10, em torno de 25 a 47 minutos. Já os que gastaram mais tempo, foram P1,P2 e P4. Sendo que o P1 gastou 2 horas, enquanto que o P2 e o P4 em torno de 1 hora.

No cenário 2, os participantes do grupo controle P1, P2, P3 e P4 ficaram em uma

faixa de tempo parecida variando de 20 minutos a 36 minutos, enquanto o que gastou mais tempo foi o P5, em torno de 50 minutos. Já no grupo SAPI, os participantes P7 e P9, gastaram em torno de 4h, enquanto que os demais ficaram na faixa de 24 a 43 minutos. Em específico, os participantes P7 e P9 passaram mais tempo executando o experimento, pois precisaram fazer um intervalo de 1 hora para almoçar no meio da execução do cenário.

No cenário 3, os participantes do grupo controle P1, P2, P3, P4 ficaram em uma faixa parecida, variando de 20 minutos a 30 minutos, enquanto que o participante P5 levou 50 minutos. No grupo SAPI, os participantes P8 e P10 gastaram menos tempo, em torno de 13 a 18 min e os que levaram mais tempo, P6, P7, P9, variaram entre 26 a 50 minutos.

No cenário 4, o participante P5, do grupo controle, que nos demais cenários apresentou o maior tempo, gastou apenas 14 minutos. Os participantes P1, P2 e P3 permaneceram dentro da faixa de tempo de 17 a 36 min e o P4 teve um o maior tempo gasto de 53 minutos. Já no grupo SAPI, os participantes P8 e P10 apresentaram o menor tempo, variando em 17 a 19 minutos, enquanto que P6 e P7 fizeram em torno de 1 hora, o participante P9 gastou 2 horas.

No cenário 5, o grupo controle apresentou o menor tempo com os participantes P1 e P5, ficando entre 13 a 20 minutos, enquanto que P2, P3 e P4 gastaram em torno de 40 a 55 minutos. Já o grupo SAPI apresentou o menor tempo com os participantes P8 e P10, que gastaram em torno de 13 a 20 minutos, enquanto que P6, P7 e P9 ficaram na faixa de tempo de 35 a 72 minutos.

Ao realizar a diferença de tempo entre os grupos, visto na Tabela 20, observa-se que o grupo SAPI teve uma diferença de tempo grande ao se comparar com o grupo controle nos cenários 1, 2 e 4. Já o grupo SAPI teve, em específico, um pico de tempo gasto no cenário 2, enquanto nos cenários 3 e 5, o tempo entre os grupos foi praticamente igual, com pouca diferença entre eles.

Apesar do experimento oferecer em torno de 4 horas para a execução, alguns participantes do grupo SAPI acabaram gastando mais tempo, como ocorreu no cenário 2.

Outro fato que é relevante considerar foram as quantidades de iterações realizadas ao aplicar a ferramenta *SigniFYIng APIs*. No cenário 2, o participante P7 fez quatro iterações e o participante P2 fez duas. Ao analisar a quantidade de iterações dos participantes, os que foram mais rápidos fizeram apenas 1 iteração, como por exemplo o P8. Essas iterações são refinamentos que a ferramenta faz diante do cenário de teste, para que sejam capturados

ao longo de sua aplicação os efeitos da metacomunicação, etiquetas de comunicabilidade e as CDNs, assim como a análise de efeitos e falhas obtidas da inspeção. A quantidade de iterações a serem realizadas é definido pelo usuário ao longo da inspeção. Na Tabela 21, são vistos os cenários e as iterações feitas por cada participante com a ferramenta *SigniFYIng APIs*. Uma conclusão que se pode ter ao olhar o número de iterações com a quantidade de tempo gasto é que os cenários que mais tiveram iterações, como os cenários 1 e 2, foram os que o grupo SAPI mais teve o aumento no tempo do cenário. Com exceção do cenário 4 que, apesar de não ter tido muitas iterações, se observou que a diferença de tempo foi alta, principalmente com os participantes P6, P7 e P9.

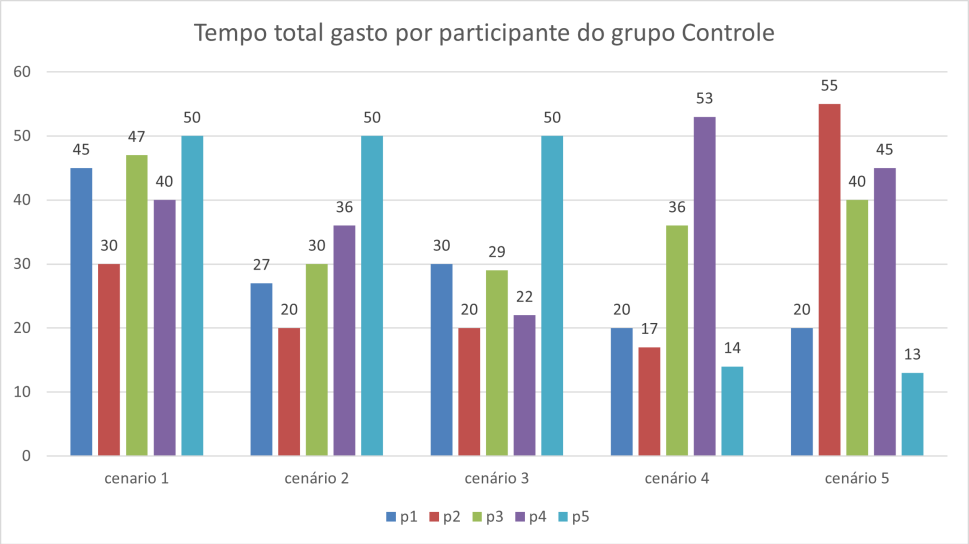

Figura 16: Tempo total do grupo controle.

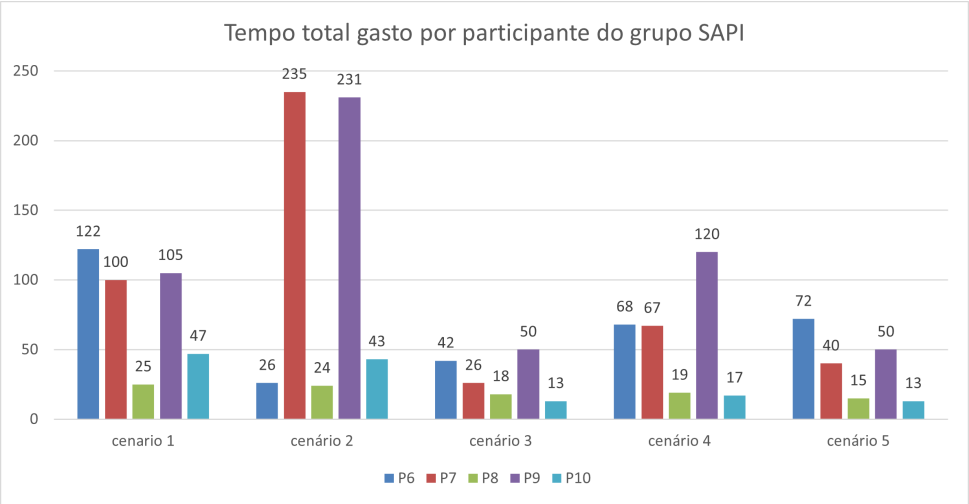

Figura 17: Tempo total do grupo SAPI.

Cenário	Controle	SAPI	Diferença de Tempo
C1	212 min	399 min	187 min
C2	163 min	559 min	396 min
C3	151 min	149 min	2 min
C4	140 min	291 min	151 min
C5	173 min	190 min	17 min

Tabela 20: Diferença do tempo total gasto entre o grupo controle e o SAPI.

	C1	C2	C3	C4	C5
P6	2	1	2	1	2
P7	3	4	1	1	1
P8	1	1	1	1	1
P9	2	2	1	1	2
P10	1	2	1	1	1

Tabela 21: Quantidade de iterações realizadas pelo grupo SAPI.

De forma geral, o tempo total gasto pelo grupo SAPI (aplicação da metodologia + elaboração dos casos de teste), no experimento, sobrecarregou a sua aplicação. Principalmente nos cenários 1, 2 e 4, o grupo SAPI teve picos de tempo gasto, em específico com alguns participantes. Comparando o cenário 3 e 5 de ambos os grupos, observa-se que a ferramenta não sobrecarregou, pois as faixas de tempo estão bem próximas.

Em resposta à Q2: De que forma *SigniFYIng APIs* sobrecarrega e de que forma dificulta o processo tradicional de teste?

O *SigniFYIng APIs* dificulta diante das três subcategorias identificadas pela sua aplicação: (1) Necessidade de treinamento; (2) Curva de aprendizado acentuada no início; (3) Falta de entendimento durante a aplicação da ferramenta.

Já com relação à sobrecarga, foram identificadas duas, a primeira agrupa a seguinte subcategoria identificada: (4) Necessidade de simplificação da ferramenta. Já a segunda é o tempo total gasto pelo grupo SAPI (aplicação da metodologia + elaboração dos casos de teste), no experimento, que sobrecarregou a sua aplicação.

5.1.3 Análise da Q3: De que forma o *SigniFYIng APIs* auxiliou no processo de teste? E como *SigniFYIng APIs* contribui para melhorar a testabilidade da *API* testada?

A questão de pesquisa Q3 teve como objetivo identificar em quais pontos a aplicação do *SigniFYIng APIs* auxiliou o processo de teste e em quais pontos ele contribuiu para melhorar a testabilidade da *API*. Para responder essa pergunta, foram utilizadas as seguintes métricas: M3.1 - Avaliação subjetiva sobre as expectativas percebidas que auxiliaram, M3.2 - Avaliação subjetiva sobre as expectativas percebidas que contribuíram. Essas informações foram extraídas do formulário de avaliação da experiência, aplicado durante experimento, como visto na seção 4.3.

A identificação das categorias que auxiliaram e contribuíram diante das respostas dos participantes permitiu encontrar as categorias respectivas a cada uma delas.

A categoria **auxiliou** agrupa as opiniões dos participantes que expressam em que pontos o *SigniFYIng APIs* auxiliou no processo de teste. Nessa categoria, foram identificadas três subcategorias: **(1) Aplicar o *SigniFYIng APIs* antes da elaboração dos casos de teste ajudou na realização dos testes**;**(2) A ferramenta instrui e incentiva a verificar se a documentação está de acordo com a *API*** e **(3) Aderência na utilização da ferramenta**.

A subcategoria **Aplicar o *SigniFYIng APIs* antes da elaboração dos casos de teste ajudou na realização dos testes** reúne opiniões dos participantes que, ao aplicar o *SigniFYIng APIs* previamente antes da elaboração dos testes, os ajudaram na condução, na familiarização e no detalhamento dos testes das *APIs*. Os comentários feitos pelos participantes que expressam essa opinião podem ser vistos nos trechos T1, T2, T3 da Tabela 22. Para representar essa subcategoria, foi escolhida a seguinte fala do participante P10:

> *"Ter executado o método antes da elaboração formal dos casos de teste me possibilitou ser mais direto ao elaborar os mesmos. Isso se deve ao fato de que durante a execução do método, a API foi explorada em função das tarefas propostas. Com isso, ao pensar nos casos de teste, já estava familiarizado com o que esperar de cada um dos cenários testados."*

Além da utilização do *SigniFYIng APIs* ter fornecido suporte aos participantes para a elaboração dos casos de teste, como visto anteriormente na subcategoria 1, também

notou-se que ela instrui a documentação do processo de busca por falha e incentiva a verificar se a documentação está dentro do esperado da *API*. Com isso, gerou a subcategoria **A ferramenta instrui e incentiva a verificar se a documentação está de acordo com a *API***, vistos nos trechos T4, T5 da Tabela 22, A seguir uma fala do participante P6 para representar a subcategoria:

> "Percebi que é uma boa ferramenta para testar se a documentação está coerente com as respostas das chamadas da *API*"

A subcategoria **Aderência na utilização da ferramenta**, no qual reúne as opiniões dos participantes caso eles tivessem a opção de utilizar o *SigniFYIng APIs* como metodologia para elaboração de testes, se os mesmos utilizariam. A sua utilização foi vista como uma boa proposta e os participantes disseram que a utilizariam. Isso pode visto nos trechos T7 e T8 da Tabela 22. Para representar o grupo, segue a fala do participante P8:

> *"excelente método, adequado a usuários que têm inclinação ao trabalho acadêmico e zela por seu trabalho como um projetista de API."*

ID	Depoimento
T1	*"O método SAPI me ajudou na elaboração dos casos de testes, pois em cada iteração do método é possível refinar ainda mais o frame de metacomunicação, o que auxilia na construção dos casos de testes" - P6*
T2	*"A ferramenta auxilia na condução de testes" - P7*
T3	*"Consideravelmente. Facilitou bastante o detalhamento dos casos de teste." - P9*
T4	*"Talvez a principal vantagem de se utilizar a abordagem SAPI seja a documentação desse processo da busca por falhas" - P10*
T5	*"Percebi que é uma boa ferramenta para testar se a documentação está coerente com as respostas das chamadas da API " - P6*
T6	*"O método se reparte em etapas bem intuitivas e fornece artifícios linguísticos que aceleram a comunicação entre projetistas de API." - P8*
T7	*"Proposta é boa." - P7*
T8	*"Usaria sim." - P9*

Tabela 22: Trechos de depoimentos que auxiliaram nos testes.

ID	Depoimento
T7	*"Relevante para a testagem de APIs e que pode ser utilizada junto com outras abordagens de teste." - P6*
T8	*"Consegue alcançar outros tipos de perspectivas (no caso, seria a comunicabilidade) que muitas outras abordagens não alcançam." - P6*
T9	*"Me lembra técnicas de inspeção de software..... a documentação do processo de inspeção. Essa documentação pode ser útil para os desenvolvedores no futuro para entender as causas de mudanças no sistema (em decorrência do reparo das falhas)." - P10*
T10	*"Entretanto, acredito que em situações reais de desenvolvimento, em que as vezes é necessário retomar documentações antigas, o método se sobressaia sobre uma abordagem ad-hoc." - P10*

Tabela 23: Trechos de depoimentos que contribuíram nos testes.

Após terem sido identificados os pontos que auxiliaram, foram levantados os fatos que contribuíram para a melhoria da testabilidade. A categoria **contribui** agrupa tudo que possibilitou, de alguma forma, o participante a ter alguma percepção diferenciada na utilização do *SigniFYIng APIs*. As subcategorias relacionadas foram: **(4) Pode ser aplicado para complementar outras abordagens**, **(5) Gera novas perspectivas para testar**, **(6) Possibilita identificar melhorias**, **(7) Gera documentação**.

As subcategorias identificadas possibilitaram entender como o *SigniFYIng APIs* pode contribuir quando aplicado em teste de software. Foi possível notar que na subcategoria **Pode ser aplicado para complementar outras abordagens** demonstra que a metodologia pode ser utilizada, não para substituir o processo tradicional de teste de software, mas sim para agregar conhecimento com outras abordagens. Como dito pelo participante P6:

> *"relevante para a testagem de APIs e que pode ser utilizada com outras abordagens de teste."*

Outro fato percebido foi que o *SigniFYIng APIs* possibilitou que os participantes presenciassem outros pontos de vistas durante a utilização da *API*. Esse fato contribui com o testador, pois facilita o encontro de um erro quando se tem perspectivas diferentes, como a de um usuário, que ele, com a visão de um testador, poderia não ter durante o teste. Segue o trecho relatado pelo participante P10:

> *"... durante o experimento, muitas vezes me vi como usuário da API (que está tentando executar alguma tarefa com a mesma), ao invés do projetista, ou testador, da mesma."*

Esse ponto percebido remete à subcategoria **Gera novas perspectivas para testar**, na qual agrupa as perspectivas que a aplicação do *SigniFYIng APIs* gera. Outro ponto identificado é que a metodologia traz a perspectiva da comunicabilidade para o ambiente de teste, o que é um diferencial, pois, como dito pelo participante no trecho T8 da Tabela 23, outras abordagens não possuem essa visão.

A aplicação do *SigniFYIng APIs*gera não apenas as falhas de comunicabilidade, como também possibilita a indicação de melhorias a serem aplicadas na *API*. A subcategoria, **Possibilita identificar melhorias**, foi identificada pelo participante P7 na seguinte frase: *"Ajudou na identificação de melhorias do software sendo testado".*

Outra contribuição notada é a respeito da documentação. Ao utilizar o *SigniFYIng APIs*, a ferramenta pede que todas as etapas, intenção, efeito e falhas, sejam bem detalhadas. Por exemplo, na etapa de efeito é preciso fazer a análise de uma interface de programação e de seus efeitos na comunicação e interação com o usuário. Para isso, deve-se analisar juntamente a expressão, o conteúdo e a intenção do cenário da *API*. Toda essa análise acaba por gerar uma documentação, agrupada na subcategoria **Gera documentação**. Ela agrupa contribuições ao se ter um processo de documentação bem detalhado, como, por exemplo, para conseguir compreender alterações que foram necessárias. Elas podem ser vistas nos trechos T9 e T10 da Tabela 23.

Em resposta à Q3: De que forma o *SigniFYIng APIs* auxiliou no processo de teste? E como *SigniFYIng APIs* contribui para melhorar a testabilidade da *API* testada?

O *SigniFYIng APIs* **auxiliou** no processo de teste nas seguintes subcategorias identificadas: **(1) Aplicar o *SigniFYIng APIs* antes da elaboração dos casos de teste ajudou na realização dos testes**; **(2) A ferramenta instrui e incentiva a verificar se a documentação está de acordo com a *API*** e **(3) Aderência na utilização da ferramenta**.

Já as suas contribuições, para melhorar a testabilidade da *API* testada, foram nas seguintes subcategorias: **(4) Pode ser aplicado de forma a complementar outras abordagens**, **(5) Gera novas perspectivas para testar**, **(6) Possibilita identificar melhorias**, **(7) Gera documentação**.

5.1.4 Análise da Q4: De que forma o *SigniFYIng APIs* obteve melhores resultados?

A questão Q4 teve como descobrir em quais situações a aplicação do *SigniFYIng APIs* teve algum diferencial do grupo controle. Como o objetivo do experimento era a elaboração do documento contendo os casos de teste por ambos grupos, foram procuradas evidências que poderiam indicar quais situações favorecem, ou não, a aplicação da ferramenta.

Para responder essa questão, foram utilizadas as seguintes métricas: M4.1 Avaliação subjetiva dos efeitos encontrados e os casos de teste escritos, comparando com o grupo controle, M4.2 Contabilizar casos de teste mal projetados pelo grupo SAPI, M4.3 Contabilizar casos de teste mal projetados pelo grupo controle e M4.3 Avaliação subjetiva da elaboração dos casos de teste.

O primeiro ponto identificado, no qual o grupo SAPI obteve um melhor resultado, foi percebido durante a análise dos casos de teste que cada um dos grupos elaborou. Identificou-se que os casos de teste do grupo controle tinham várias falhas de projeção, ou seja, o participante descreveu um caso de teste de uma situação que levava a um *bug*, porém esse *bug* não fazia sentido, tendo em vista a documentação da *API*. Diante disso, foram contabilizados os casos de teste mal projetados de ambos os grupos, como pode ser conforme pode ser visto na Figura 18. Nela, é possível observar quais foram os cenários que tiveram maior quantidade de casos de teste mal projetados. No caso, nos cenários 2 e 3, o grupo controle teve em média uma quantidade maior identificada, enquanto nos cenários 1, 4 e 5, obteve-se resultados semelhantes entre os grupos.

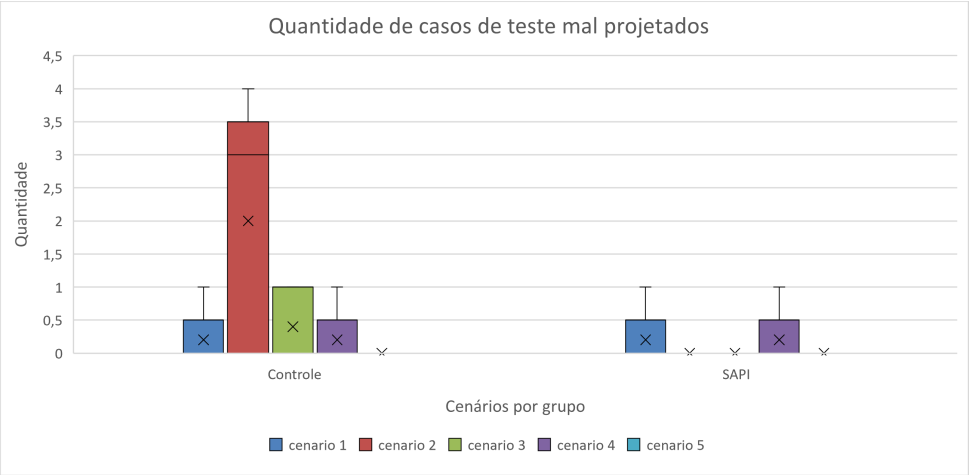

Figura 18: Quantidade de casos de teste mal projetados.

O segundo ponto observado foi em relação ao tempo para elaboração dos casos de teste. Ao verificar quanto tempo cada participante gastou por cenário para elaborar o

caso de teste, é possível perceber, como visto na Figura 19, que o grupo SAPI gastou em média menos tempo que o grupo controle. Como o grupo SAPI aplicou primeiramente a ferramenta nos cenários, ao elaborar os casos de teste, como visto na questão de pesquisa Q3, os participantes já sabiam em quais posições se encontravam as falhas dos cenários. Isso levou os participantes a gastarem menos tempo na etapa em que elaboraram apenas os casos de teste.

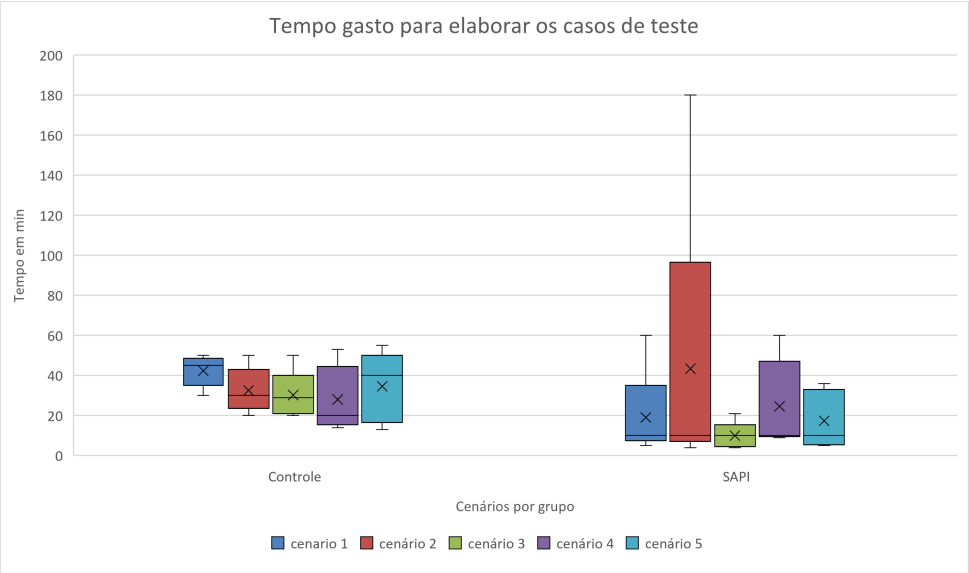

Figura 19: Tempo gasto para elaboração de caso de teste.

O terceiro ponto analisado foi em quais situações os participantes do grupo SAPI conseguiram elaborar mais casos de teste. Essas informações podem ser deduzidas a partir das Tabelas 14 e 15, nas quais pode-se ver que as condições que levaram os participantes a escrevem casos de testes continham CDNs relacionadas a *consistência*, *propensão a erros* e *expressividade dos papéis*. Sendo que a *consistência* foi a que mais levou os participantes a criarem casos de teste. Em relação ao cenário, observou-se que o cenário 2 teve mais situações de criação de casos de teste pelo grupo SAPI.

Tendo identificado em quais situações e em qual cenário que o *SigniFYIng APIs* possibilitou elaborar maior quantidade de casos de teste, foi levantada a quantidade de *bugs* que cada um dos grupos obteve em cada cenário, visto na Figura 21.

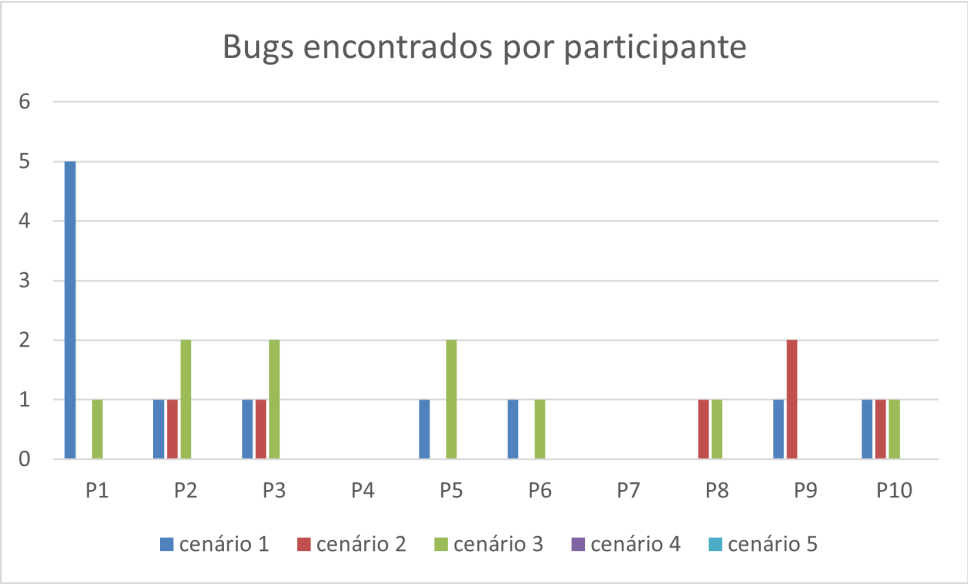

Figura 20: Quantidade de *bugs* encontrados por participante.

No cenário 1, observa-se que o grupo controle encontrou no máximo 5 *bugs*, enquanto que no grupo SAPI, o máximo foi de 1 *bug*. Vale ressaltar que, no grupo controle, essa variação de 5 bugs, visto na Figura 20, foi encontrada apenas pelo participante P1. Como pode ser visto na Tabela 10, nota-se que o participante P1 já tinha experiência na área e conhecia a *API* do Trello, o que pode ter dado uma vantagem para encontrar mais *bugs*. Ao comparar a mediana dos grupos, percebe-se que ambos ficaram com uma média de 1 *bug* encontrado.

No cenário 2, ao comparar a mediana do grupo controle com o grupo SAPI, nota-se que ambos encontraram 1 *bug*. Porém, se comparar a quantidade máxima de bugs encontrados, percebe-se que o grupo SAPI encontrou 2 *bugs*, enquanto o grupo controle encontrou 1 *bug*. Nesse cenário em específico, o grupo SAPI elaborou mais casos de teste, como também encontrou mais *bugs*. Vale ressaltar que no grupo controle os participantes que encontram *bugs* foram: o P2 e P3, enquanto que no grupo SAPI foram três: P8, P9 e P10, sendo que P9 encontrou 2 bugs. Ao analisar o perfil dos participantes, percebe-se que, no grupo do SAPI, o P9 possuía experiência profissional na área, enquanto que o P8 e P10 eram acadêmicos. Quanto ao perfil do grupo controle, o participante P2 possuía experiência profissional e o P3 era acadêmico. Nesse cenário, conclui-se que o perfil não interferiu no grupo controle. Já no grupo SAPI, pode-se notar que o participante P9 explorou bem os casos de teste, encontrando mais *bugs* sendo um participante com experiência profissional.

No cenário 3, os participantes do grupo controle que encontraram *bugs* foram P1, P2, P3 e P5, enquanto que no grupo SAPI, foram P6, P8 e P10. Nesse cenário, o grupo

controle encontrou mais *bugs*, em específico os participantes P2, P3 e P5 que identificaram 2 *bugs*. Já no grupo SAPI, todos os três participantes encontraram apenas 1 *bug*. Nesse cenário, não foi possível observar que os perfis dos participantes interferiram nas respostas.

Ao analisar a Figura 20, percebe-se que os participantes P4 e P7 não identificaram nenhum *bug*. Isso ocorreu porque eles acabaram elaborando casos de teste mal projetados e, em específico, o participante P4 acabou fazendo outros tipos de testes, como por exemplo de segurança que foi desconsiderado.

Nos cenários 4 e 5, não foram encontrados *bugs* por nenhum participante, o que já era de se esperar, pois esses cenários realmente não tinham *bugs*. O foco desses cenários eram propor uma análise da documentação e investigar se os participantes conseguiriam utilizar e testar a *API* com o que tinha sido fornecido na documentação.

No cenário 5, apenas o participante P2 chegou a testar os casos de testes. Alguns participantes como o P4 e P7 não conseguiram entender a *API*, impossibilitando de executar o cenário, pois desistiram. Já os demais participantes tiveram sucesso em utilizar, ou seja, entenderam a *API*, porém não elaboraram casos de testes.

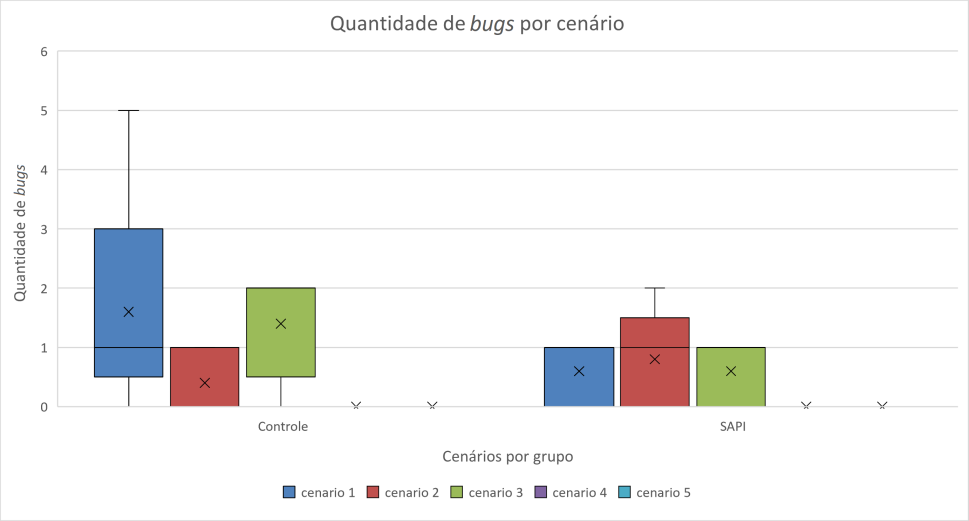

Figura 21: Quantidade de *bugs* encontrados.

O quarto ponto identificado, foram as situações de melhorias. As situações de melhorias, como visto previamente na questão Q1, podem ser identificadas nas Tabelas 14 e 15. Por meio delas, é possível relacionar quais cenários precisam de melhorias e quais tipos de melhorias. Os tipos de melhorias podem ser vistos nas situações O1 a O7, observadas na Tabela 13. Os cenários com a situação de **Melhoria** tiveram, respectivamente, a maior quantidade de CDNs relacionadas ao *Nível de Abstração* e *Operações Mentais difíceis*.

As etiquetas mais vistas foram T6 e T1, as quais se relacionam a falhas temporárias e a falhas totais. Em relação ao cenário que mais necessita de melhoria, é o cenário 5. A aplicação do *SigniFYIng APIs* permitiu identificar que as melhorias que o cenário 5 precisa se relacionam às situações O1, O2 e O6 observadas. Esses relatos são importantes, pois possibilitam a geração de documentação pelos participantes com pontos relevantes e necessários a serem ajustados, ajudando tanto na melhoria da documentação, como interferindo na utilização da *API*.

Em resposta à Q4: De que forma o *SigniFYIng APIs* obteve melhores resultados?

O *SigniFYIng APIs* obteve melhores resultados ao projetar casos de teste; no tempo médio para a elaboração dos casos de teste (desconsiderando a aplicação do *SigniFYIng APIs*); ao identificar as situações de melhorias, possibilitando a identificação e agrupamento dos tipos de melhorias, assim como alterações que são relevantes a serem realizadas nas *APIs*. Percebeu-se também que os participantes conseguiram elaborar mais casos de testes diante das situações em que foram identificadas CDNs de *consistência*, *propensão a erros* e *expressividade dos papéis*.

5.2 Discussão dos Resultados

Nessa seção, são analisados os resultados encontrados em cada uma das subquestões da pesquisa realizada na seção 5.1. Apresentado o resultado obtido por meio da aplicação do GQM, esses resultados direcionam a resposta da questão principal da pesquisa a qual é: Será que a identificação de problemas de comunicabilidade pode nos levar a elementos para a melhoria dos casos de testes e a apoiar a testabilidade?

A abordagem de utilização do *SigniFYIng APIs*, com o intuito de apoiar o processo tradicional de casos de teste, trouxe aspectos que foram identificados como dificuldades e que sobrecarregaram a sua utilização. Entretanto, revelou aspectos que geraram auxílio para o processo de elaboração de casos de teste, assim como contribuíram para a testabilidade da *API* testada. Além disso, foi realizada uma comparação entre os grupos a qual pode-se concluir em quais pontos o grupo SAPI teve melhores resultados diante do experimento.

Sobre as dificuldades que a utilização do *SigniFYIng APIs* gerou, foram identificadas as seguintes categorias de significado: (1) Necessidade de treinamento; (2) Curva de aprendizado acentuada no início; (3) Falta de entendimento durante a aplicação da ferramenta e (4) Necessidade de simplificação da ferramenta. As dificuldades dos participantes são as

categorias (1), (2) e (3), as quais se concentram no aprendizado e nas dificuldades iniciais que se tem ao conhecer a metodologia. Como a metodologia não é de conhecimento prévio na área de teste, existe a necessidade de treinamento. Os conceitos do *SigniFYIng APIs* envolvem conhecimento prévio em alguns conceitos da Engenharia Semiótica o que justifica os participantes terem dificuldades no início do aprendizado. Sobre a categoria (3) Falta de entendimento durante a aplicação da ferramenta, os participantes relataram que mesmo tendo sido fornecidas aulas gravadas com um exemplo prático e um manual de como aplicar a ferramenta, eles se sentiram perdidos em como aplicar e entender como seriam os resultados obtidos pelo *SigniFYIng APIs*. A sobrecarga identificada foi a categoria de significado (4) Necessidade de simplificação da ferramenta, em que essa necessidade foi relatada como um fator que sobrecarrega a utilização do *SigniFYIng APIs* em um processo de teste de software. Como relatado pelos participantes, a metodologia possui muitas escolhas por parte deles, o que implicaria em um aumento excessivo de esforço durante o teste.

Quanto à pergunta **Será que a identificação de problemas de comunicabilidade pode levar o testador a encontrar elementos para a melhoria dos projetos de casos de testes e a apoiar a testabilidade?**, foi possível identificar que o *SigniFYIng APIs* auxiliou os seguintes aspectos: (1) Aplicar o *SigniFYIng APIs* antes da elaboração dos casos de teste ajudou na realização dos testes; (2) A ferramenta instrui e incentiva a verificar se a documentação está de acordo com a *API* e (3) Aderência na utilização da ferramenta. As categorias (1) e (2) mostram que a ferramenta leva o testador a, primeiramente, conhecer e entender melhor os cenários dos quais ele iria escrever os casos de teste, assim como também instruiu e incentivou os participantes a verificarem se a documentação estava funcionado conforme o esperado. A categoria (3) mostra que a proposta foi bem aceita pelos participantes os quais disseram que utilizariam o *SigniFYIng APIs* para auxiliar no processo tradicional de teste.

Quanto às suas contribuições para melhorar a testabilidade das *APIs* que estão sendo testadas, foi identificado que: (4) Pode ser aplicado de forma a complementar outras abordagens, (5) Gera novas perspectivas para testar, (6) Possibilita identificar melhorias e (7) Gera documentação. Essas quatro contribuições identificadas agregam valor ao aplicar o *SigniFYIng APIs* em um processo de teste e acaba contribuindo para a elaboração de casos de teste mais adequados, impactando na testabilidade da *API*.

Sobre essas contribuições, percebe-se uma relação entre os itens (4) e (5). Da mesma forma que o *SigniFYIng APIs* não precisa ser a única abordagem aplicada, ele pode ser

utilizado para complementar outras técnicas já existentes de verificação e validação, como inspeções (FAGAN, 2001), por meio da geração de novas perspectivas no momento que ele é aplicado. A perspectiva, no momento de teste, é algo que contribui até mesmo para a identificação de falhas na *API*, pois conseguindo tirar a visão comum de um testador e olhar com um olhar de um usuário final, por exemplo, contribui para identificar pontos que normalmente poderiam não terem sido vistos. Nessa pesquisa, percebeu-se que, principalmente essas novas perspectivas vistas pelos participantes, geraram relatos de melhorias nas *APIs* testadas, o que leva à contribuição (6).

Em relação às situações identificadas pelos participantes como *"inconclusivas"*, pode-se fazer uma convergência com o que foi observado no estudo Eler, Bertolino e Masiero (2013). Nesse estudo, os autores apresentaram a dificuldade de teste diante ambientes de caixas pretas, onde que não possuem acesso aos dados internos dos serviços de terceiros. Nessas situações, os autores relataram que os casos de teste precisam ser definidos baseados exclusivamente na interface publicada e na documentação. Em convergência com a literatura, foi observado no experimento que, em algumas situações, os participantes, apesar de terem feito o teste, não tinham como prosseguir, pois chegaram a situações inconclusivas. Ou seja, apenas com a documentação fornecida e com a *API*, não era possível identificar como o erro acontecia. Também não era possível saber se o problema dependia de alguma interação com serviços de terceiros que não possibilitaram o sucesso da requisição realizada.

Quanto às melhorias identificadas, foram categorizadas pelos *ids* O1 a O7 na Tabela 12, os quais destacam problemas de comunicabilidade em comum encontrados em cada um dos cenários. As melhorias que são necessárias envolvem os seguintes pontos:

- O1: A documentação não deixa claro como funciona o sistema e suas regras de negócio.
- O2: A nomenclatura poderia ser mais compreensível.
- O3: A resposta da *API* não deixou claro se funcionou, ou se deu erro.
- O4: A documentação não informa quais caracteres são aceitos.
- O5: O formato esperado do parâmetro não estava bem definido.
- O6: A documentação não detalha os conceitos do domínio da aplicação.
- O7: A complexidade para obter um dado específico da *API*.

Nota-se que as categorias O1 a O7 com melhorias identificadas levam a sugestões das quais mostram como a documentação das *APIs* testadas é falha ou imprecisa. O *SigniFYIng APIs* possibilitou a geração de uma documentação contendo essas melhorias descritas em detalhes por cada participante, o que leva a contribuição (6) à geração de documentação.

Ao analisar de que forma as melhorias contribuíram para a elaboração de casos de teste, pode-se perceber que, dentre as melhorias listadas, algumas ficaram apenas como um relato de melhoria da documentação. O outro grupo de melhorias possibilitou que os participantes elaborassem casos de teste, as quais foram as seguintes: O1: A documentação não deixa claro como funciona o sistema e suas regras de negócio; O4: Documentação não informa quais caracteres são aceitos; O5: O formato esperado do parâmetro não estava bem definido. Observando O1, O4 e O5, nota-se que estão presentes nessas três situações as CDNs *consistência*, *propensão a erros* e *expressividade dos papéis*. Sendo que a *consistência* foi a que mais proporcionou a elaboração de casos de teste. Ou seja, diante dos cinco cenários propostos, observou-se que, quando identificadas falhas na consistência, a ferramenta ajudou o grupo SAPI na elaboração dos casos de teste.

Ao realizar uma comparação entre os dois grupos diante do experimento aplicado, identificaram-se situações em que o *SigniFYIng APIs* obteve melhores resultados. A primeira foi em relação à elaboração de casos de teste. Quanto a isso, o grupo SAPI elaborou casos de teste mais objetivos e diretos nas falhas encontradas diante da aplicação do *SigniFYIng APIs*. Como também teve menos casos de teste mal projetados, pode-se relacionar esse resultado à categoria de auxílio vista anteriormente, a categoria (2), no qual o *SigniFYIng APIs*, ou seja, a ferramenta não apenas instruiu o participante a olhar mais atentamente a documentação, como também teve como resultado casos de teste melhores projetados.

A segunda situação foi em relação ao tempo gasto pelos participantes na elaboração de casos de teste de cada um dos grupos. O tempo do grupo SAPI foi menor, o que se relaciona com a categoria (1), já vista, de pontos identificados nos quais o *SigniFYIng APIs* auxiliou. Ou seja, o participante, ao aplicar a ferramenta previamente, já tinha noções de como funcionava cada cenário e, diante disso, já pode ser mais direto na escrita de casos de teste.

A terceira foi em relação a quais situações levaram o grupo SAPI a encontrar mais falhas. Sobre isso, concluiu-se que o cenário que teve as CDNs relacionadas a *consistências* levou o grupo SAPI a elaborar mais casos de teste, o que ajudou a encontrar mais falhas

no cenário testado.

Por último, tiveram as melhorias relatadas diante da execução do *SigniFYIng APIs*. Essas melhorias, como já dito anteriormente, foi um grande diferencial, pois utilizando os relatos dos participantes, pode-se identificar em quais quesitos a documentação precisa melhorar.

5.3 Limitações do *SigniFYIng APIs*

Como é de conhecimento, o objetivo da ferramenta, diante do seu uso, é promover a reflexão do analista durante a inspeção de uma *API*. O estudo prévio que foi feito por Morelli, Neves e Salgado (2021) com o *SigniFYIng APIs*, assim como o experimento realizado nesta dissertação, permitiram verificar a possibilidade de utilizar a ferramenta dentro do processo de teste de software.

Todavia, a aplicação da ferramenta *SigniFYIng APIs*, diante do experimento, permitiu identificar algumas limitações com o seu uso dentro do processo de investigação da testabilidade.

A primeira é que ela precisa de uma adaptação para tornar seu uso mais simples, pois diante do que foi observado no experimento, os participantes ficaram na dúvida de como eles utilizariam os conceitos aprendidos da ferramenta *SigniFYIng APIs* diante um ambiente de teste, como também acharam complexos os termos utilizados. É importante ressaltar que a ferramenta tem uma curva de aprendizagem acentuada no início, pois envolve diversos termos e conceitos da Engenharia Semiótica.

Outro ponto é que essa ferramenta não foi desenvolvida para ser aplicada na área de testes, então foi necessário fazer adaptações para simplificar o seu uso pelos participantes. A aplicação do *SigniFYIng APIs* diante os cenários de teste, nesse experimento, foi realizada por meio do preenchimento de formulários que guiavam o participante a aplicar o *SigniFYIng APIs* na *API* testada. Durante a aplicação do experimento, observou-se que os participantes tiveram dúvidas de como utilizariam o formulário para aplicar o *SigniFYIng APIs* nos testes. A principal dúvida dos participantes foi de como seria realizada as diversas iterações que a ferramenta propõem para refinar o entendimento da *APIs*. Essa limitação pode ter sido gerada pela própria ferramenta *SigniFYIng APIs*, pois ela não especifica claramente como isso deve ser feito, apenas diz que o refinamento e o entendimento são feitos diante as iterações. Essa limitação também pode ter sido ocasionada pelo treinamento fornecido, pois não houve um treinamento prévio simulando

como seria a aplicação do *SigniFYIng APIs* nos cenários propostos.

A segunda limitação é que, pelo menos nessa pesquisa, não foi possível identificar se a ferramenta pode identificar, em quantidade, mais *bugs* do que um grupo que não a utilize. Apesar de no cenário 2 ter sido encontrado mais *bugs* pelo grupo SAPI, precisaria de mais cenários além dos cinco que foram utilizados nessa pesquisa para comprovar esse fato.

A terceira limitação observada é que nem todas as CDNs que a ferramenta *SigniFYIng APIs* fornece possibilitaram, de fato, a criação de casos de teste.

5.4 Limitações da aplicação do experimento

Além das limitações identificadas pela ferramenta, foram listadas algumas que podem ter sido ocasionadas pela aplicação do experimento.

Uma delas foi a forma que foi passado para os participantes aplicarem a ferramenta. Notou-se que o grupo SAPI focou tanto na aplicação da ferramenta que, no momento de elaborar os casos de teste, eles foram mais objetivos e elaboraram os casos de teste apenas com o que foi obtido no *SigniFYIng APIs*. Isso, apesar de ter sido um ponto positivo da aplicação da ferramenta, também limitou o grupo de explorar mais os cenários de casos de teste, como os que foram levantados no mapeamento. O ideal era que eles identificassem mais cenários além dos que a ferramenta *SigniFYIng APIs* permitiu eles identificarem. Essa limitação identificada pode ter sido ocasionada pela forma que a aplicação da ferramenta foi sugerida.

Como a aplicação da ferramenta já requer uma análise, e pensar nos cenários de testes também, percebeu-se que o ideal seria aplicar o *SigniFYIng APIs* de uma forma diferente do realizado. Uma sugestão seria, por exemplo, ter duas equipes: uma equipe A para aplicar o *SigniFYIng APIs* e uma equipe B que receberia a análise do grupo A e iria realizar os casos de testes.

Outra possível limitação foi a forma que o experimento foi conduzido, pois foi solicitado que os participantes deveriam aplicar o *SigniFYIng APIs* em todos os cenários, e somente após a conclusão seriam elaborados os casos de teste. Em observação ao que foi elaborado pelos participantes, percebeu-se que talvez teria sido melhor que a elaboração dos casos de testes fossem feitos logo após a aplicação do *SigniFYIng APIs*, antes de ir para o próximo o cenário. Ao realizar cada cenário por vez, o participante exploraria mais os casos de teste da *API* que ele acabou de aplicar o *SigniFYIng APIs*. Essa abordagem de aplicar

o *SigniFYIng APIs* primeiramente em todos os cenários, e somente depois elaborar os casos de testes, foi pensado para que não quebrasse o raciocino do participante quanto a aplicação da ferramenta.

5.5 Ameaças à validade

Nesse experimento, foram considerados quatro principais ameaças que representam risco em ter gerado uma interpretação imprópria dos resultados: (1) tamanho da amostra, (2) cenários de teste com número pequeno de falhas, (3) cenário com contexto limitado, (4) fadiga dos participantes.

Em relação à ameaça (1), a pesquisa teve um tamanho de amostra reduzido, limitado a 10 participantes, no qual o foco foi na análise qualitativa dos dados gerados de cada um dos participantes. Um experimento com um número maior de participantes pode apresentar resultados distintos, comparado à amostra reduzida.

Em relação à ameaça (2), apesar de terem sido escolhidos cinco cenários distintos, nos quais três apresentavam *bugs* e dois apresentavam documentações complexas, consideramos que cenários com maiores números de *bugs* podem gerar algum resultado diferente. Por exemplo, é provável que se encontre mais CDNs que possibilitem a criação de casos de teste, ou até mesmo possibilitar uma análise dos tipos de falhas existentes do *SigniFYIng APIs*: a falha parcial; temporária e completa diante da criação de casos de teste.

Em relação a ameaça (3), a aplicação do experimento foi limitada a três cenários de *APIs* do Trello e a duas *APIs* do Moodle.

Em relação a ameaça (4), a fadiga dos participantes, apesar do tempo do experimento ter sido limitado a 4 horas, alguns participantes gastaram um pouco mais de tempo do que o esperado, portanto, é compreensível que o participante possa ter ficado cansado, principalmente nos últimos cenários vistos. Outro caso que pode ter tido algum impacto foi em relação aos cenários do Moodle. Como as *APIs* do Moodle foram as últimas a serem executadas, e elas eram bem diferentes dos cenários do Trello, os resultados obtido nesses cenários podem ter sido comprometidos pela possível fadiga dos participantes.

5.6 Considerações do capítulo

Este capítulo teve como objetivo apresentar os resultados obtidos durante o experimento, possibilitando, assim, responder às questões de pesquisa definidas no GQM. Com as questões de pesquisas respondidas, foi possível responder à questão de pesquisa principal: Será que a identificação de problemas de comunicabilidade pode levar o testador a encontrar elementos para a melhoria dos projetos de casos de testes e a apoiar a testabilidade? No próximo capítulo, seguem as conclusões desta pesquisa.

6 Conclusões

Em vista do aumento da complexidade que os sistemas têm tomado nas diversas esferas da sociedade, observa-se a exigência de serem feitas integrações entre serviços que se comunicam por meio de *APIs*. O desafio de construir uma *API* bem projetada e que seja bem compreendida pelo consumidor do serviço ainda é de difícil abordagem, visto que as formas de documentações de *APIs* não são universais, prejudicando as atividades de garantia de qualidade. Diante disso, nota-se a importância de se ter um método, ou soluções sólidas, que garantam a qualidade de desenvolvimento do software que utiliza um serviço.

Como primeira contribuição, este trabalho investiga o uso da ferramenta *SigniFYIng APIs* como método para investigar a capacidade da ferramenta em identificar, na documentação de *APIs*, os fatores que possam alterar a testabilidade e, com isso, possibilitar uma melhora na qualidade do software produzido. Com o intuito de verificar se a ferramenta atingia esse propósito, foi realizado um experimento com duas *APIs* de código aberto: do Trello e a do Moodle.

Diante da aplicação do experimento, foi possível observar que o conhecimento adquirido da documentação da *API*, a partir do uso do *SigniFYIng APIs*, possibilitou quatro principais contribuições. A primeira foi que em situações em que foram encontradas falhas de comunicabilidade relacionadas às CDNs de *consistência*, *propensão a erros* e *expressividade dos papéis* direcionaram os participantes a elaborarem casos de teste. Em específico, a CDN de *consistência*, se comparada com as demais encontradas, foi a que mais proporcionou a elaboração. A segunda é, em relação à primeira em que o *SigniFYIng APIs* mais levou os participantes a escreverem casos de teste, foi o cenário que possibilitou o grupo identificar mais falhas comparando com o grupo controle.

A terceira contribuição é relacionada à forma de elaborar os casos de teste, em que percebeu-se que o *SigniFYIng APIs* levou os participantes a criarem casos de testes mais concisos e objetivos, como também melhor projetados. A projeção de casos de teste que não

correspondem ao esperado pela *API* são normalmente ocasionados pela mal compreensão da documentação da *API*. Nesse sentido, pode-se dizer que o *SigniFYIng APIs* instrui o participante a ficar atento à documentação, bem como o seu entendimento. A quarta contribuição foi em relação ao tempo que o grupo demorou para construir os casos de teste, obteve-se que em média o grupo SAPI levou menos tempo que o grupo controle. A vantagem de aplicar o *SigniFYIng APIs* antes de ir para a elaboração dos casos de teste fez com que os participantes conhecessem previamente as *APIs* e os próprios resultados obtidos ao aplicar a ferramenta os quais foram utilizados para elaborar os casos de teste.

Consideramos neste trabalho que o processo proposto é utilizado pela equipe de desenvolvimento e de testes do software que consome a *API*. No entanto, os resultados obtidos por meio do processo também poderiam ser utilizados pela equipe que disponibilizou a *API*. No resultado do experimento, tiveram situações que, apesar de o grupo SAPI ter identificado a falha, eles não conseguiram elaborar um caso de teste, pois não era possível identificar o porquê de o sistema estar se comportando daquela forma. Essas situações foram denominadas nas Tabelas como situações inconclusivas. Nesse caso, uma vez identificada esse tipo de situação, essa informação pode ser repassada para o produtor do serviço a fim de melhorar a documentação para melhorar a testabilidade do serviço.

De modo geral, foram identificados pontos favoráveis e desfavoráveis para a utilização da metodologia proposta. Os pontos, nos quais a aplicação foi desfavorável, relacionam-se principalmente: à dificuldade dos participantes em relação ao aprendizado e entendimento da ferramenta e à sobrecarga que a ferramenta, diante da sua complexidade de aplicação, gera durante o teste de software. Os resultados favoráveis obtidos mostram que o conhecimento adquirido da documentação da *API*, a partir do uso do *SigniFYIng APIs*, possibilitou: a elaboração de casos de teste diante das falhas de comunicabilidade, a identificação das principais melhorias a serem realizadas nas *APIs* testadas, a escrita de casos de testes mais concisos e melhores projetados conforme a documentação fornecida.

Considerando os resultados da pesquisa, temos passos para trabalhos futuros a serem explorados:

- Aplicação do *SigniFYIng APIs* de forma diferente da adotada nesse experimento. Já que executar a ferramenta demanda tempo e testes, uma forma que pode ser útil é aplicar ele separadamente da elaboração dos casos de teste. Ou seja, ter duas equipes em que a primeira aplica a metodologia e gera o relatório com as principais falhas de comunicabilidade da *API* e a segunda recebe esses dados para guiar e orientar a elaboração dos casos de teste, permitindo que o usuário não se restrinja apenas ao

que foi encontrado.

- Adaptação da ferramenta *SigniFYIng APIs* para uma aplicação mais simples e objetiva. Uma sugestão seria incluir passos do BDD [1] (SMART, 2014), juntamente à etapa de intenção, incentivando e guiando o usuário a identificar cenários de teste.

- Aplicação do *SigniFYIng APIs* por uma equipe que realiza inspeção de artefatos. A inspeção de artefatos envolve analisar os artefatos de software para ver se os mesmos possuem algum tipo de defeito. Como o *SigniFYIng APIs* instrui a ler a documentação em busca da compreensão da intenção, efeitos e falhas, essa técnica poderia ser adaptada para ser aplicada à etapa de inspeção de artefatos. Dessa forma já se identificaria falhas de comunicabilidade, antes mesmo das novas funcionalidades serem desenvolvidas, possibilitando que a documentação fosse ajustada antes mesmo de ir para a etapa de teste.

- Utilização do *SigniFYIng APIs* como um processo de metrificação de teste de usabilidade. Como o *SigniFYIng APIs* obtém a padronização dos efeitos de um usuário reagindo com a interface de uma *API*, se considerarmos as etiquetas e efeitos obtidos, é possível identificar, por exemplo, quais situações o usuário não entendeu, ou não compreendeu.

[1] Desenvolvimento Orientado a Comportamento: do inglês, "Behavior Driven Development"

REFERÊNCIAS

AFONSO, Luiz Marques. **Communicative dimensions of application programming interfaces (APIs)**. 2015. Tese (Doutorado) – Programa de Pós-Graduaçao em Informática, Pontícia Universidade Católica do Rio De Janeiro, Rio de Janeiro, Brazil.

AFONSO, Luiz Marques; CERQUEIRA, Renato F de G; DE SOUZA, Clarisse Sieckenius. Evaluating application programming interfaces as communication artefacts. **System**, v. 100, p. 8–31, 2012.

BINDER, Robert V. Design for Testability in Object-Oriented Systems. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 37, n. 9, p. 87–101, set. 1994. ISSN 0001-0782. DOI: 10.1145/182987.184077. Disponível em: <https://doi.org/10.1145/182987.184077>.

BLACKWELL, Alan F et al. Cognitive dimensions of notations: Design tools for cognitive technology. In: SPRINGER. INTERNATIONAL conference on cognitive technology. [S. l.: s. n.], 2001. P. 325–341.

DE SOUZA, Clarisse Sieckenius. **The semiotic engineering of human-computer interaction**. [S. l.]: MIT press, 2005.

DE SOUZA, Clarisse Sieckenius et al. **Software Developers as Users: Semiotic Investigations in Human-Centered Software Development**. [S. l.]: Springer, 2016.

DELAMARO, Marcio; JINO, Mario; MALDONADO, Jose. **Introdução ao teste de software**. [S. l.]: Elsevier Brasil, 2021.

ED-DOUIBI, Hamza; IZQUIERDO, Javier Luis Cánovas; CABOT, Jordi. Automatic generation of test cases for REST APIs: A specification-based approach. In: IEEE. 2018 IEEE 22nd international enterprise distributed object computing conference (EDOC). [S. l.: s. n.], 2018. P. 181–190.

ECO, U. **A theory of Semiotics**. [S. l.]: Indiana University Press, 1976.

EILERTSEN, Anna Maria; BAGGE, Anya Helene. Exploring API: Client Co-Evolution. In: PROCEEDINGS of the 2nd International Workshop on API Usage and Evolution. Gothenburg, Sweden: Association for Computing Machinery, 2018. (WAPI '18), p. 10–13. ISBN 9781450357548. DOI: 10.1145/3194793.3194799.

ELER, Marcelo Medeiros; BERTOLINO, Antonia; MASIERO, Paulo Cesar. Applying Structural Testing to Services Using Testing Interfaces and Metadata. **Int. J. Software and Informatics**, v. 7, n. 2, p. 239–271, 2013.

FAGAN, Michael E. Advances in Software Inspections. In: **Pioneers and Their Contributions to Software Engineering: sd&m Conference on Software Pioneers, Bonn, June 28/29, 2001, Original Historic Contributions**. Edição: Manfred Broy e Ernst Denert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. P. 335–360. ISBN 978-3-642-48354-7. DOI: 10.1007/978-3-642-48354-7_14. Disponível em: <https://doi.org/10.1007/978-3-642-48354-7_14>.

FAROOQ, Umer; ZIRKLER, Dieter. API peer reviews: a method for evaluating usability of application programming interfaces. In: PROCEEDINGS of the 2010 ACM conference on Computer supported cooperative work. [S. l.: s. n.], 2010. P. 207–210.

GONZÁLEZ, A.; PIEL, É.; GROSS, H. A Model for the Measurement of the Runtime Testability of Component-Based Systems. In: 2009 International Conference on Software Testing, Verification, and Validation Workshops. [S. l.: s. n.], 2009. P. 19–28. DOI: 10.1109/ICSTW.2009.9.

HOOPES, J; PEIRCE, C.S. **Peirce on Signs: Writings on Semiotic by Charles Sanders Peirce**. [S. l.]: The University of North Carolina Pres, 1991.

IEEE. IEEE Standard Glossary of Software Engineering Terminology. **IEEE Std 610.12-1990**, p. 1–84, 1990. DOI: 10.1109/IEEESTD.1990.101064.

JAKOBSON, R. **Linguistics and poetics. In: Style in language**. [S. l.]: MIT Press, 1960.

KITCHENHAM, Barbara; CHARTERS, Stuart. Guidelines for performing systematic literature reviews in software engineering. Citeseer, 2007.

LEITÃO, Carla Faria; PRATES, Raquel Oliveira. A aplicação de métodos qualitativos em computação. **Jornadas de Atualização em Informática**, v. 2017, p. 43–90, 2017.

LOPES, Adriana; CONTE, Tayana; SOUZA, Clarisse Sieckenius de. Directives of Communicability for Software Models as Boundary Objects in Software Development, 2018.

LOPES, Adriana; OLIVEIRA, Edson et al. Directives of communicability: towards better communication through software models. In: IEEE. 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). [S. l.: s. n.], 2019. P. 45–48.

LOURENÇO MARCOS, Emanuel; PUCCINELLI DE OLIVEIRA, Raphael. A Framework for Guidance of API Governance: A Design Science Approach, 2019.

MALLA, Prakash; GURUNG, Bhupendra. Adaptation of Software Testability Concept for Test Suite Generation: A systematic review, 2012.

MOODLE. **Moodle**. [S. l.: s. n.], 2021. https://docs.moodle.org/all/pt_br/Sobre_o_Moodle. Accessed: 2023-02-02.

MORELLI, Camila Pardo Garcia; NEVES, Vânia de Oliveira; SALGADO, Luciana. Investigando Comunicabilidade e Testabilidade com a ferramenta Signifying APIs. **Anais do Computer on the Beach**, v. 12, p. 443–450, 2021.

MOURÃO, Erica et al. On the performance of hybrid search strategies for systematic literature reviews in software engineering. **Information and Software Technology**, Elsevier, v. 123, p. 106294, 2020.

NICOLACI-DA-COSTA, Ana Maria; LEITÃO, Carla Faria; ROMÃO-DIAS, Daniela. Como conhecer usuários através do Método de Explicitação do Discurso Subjacente (MEDS). **VI Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais, IHC**, p. 47–56, 2004.

O'BRIEN, L.; MERSON, P.; BASS, L. Quality Attributes for Service-Oriented Architectures. In: INTERNATIONAL Workshop on Systems Development in SOA Environments (SDSOA'07: ICSE Workshops 2007). [S. l.: s. n.], 2007. P. 3–3. DOI: 10.1109/SDSOA.2007.10.

ROBILLARD, Martin P; DELINE, Robert. A field study of API learning obstacles. **Empirical Software Engineering**, Springer, v. 16, n. 6, p. 703–732, 2011.

RUNESON, Per et al. **Case Study Research in Software Engineering: Guidelines and Examples**. [S. l.]: John Wiley & Sons, 2012.

SENIVONGSE, Twittie; PUAPOLTHEP, Assawin. A maintainability assessment model for service-oriented systems. In: PROCEEDINGS of the World Congress on Engineering and Computer Science. [S. l.: s. n.], 2015. v. 1, p. 139–144.

SMART, John. **BDD in Action: Behavior-driven development for the whole software lifecycle**. [S. l.]: Simon e Schuster, 2014.

SOLINGEN, Rini van; BERGHOUT, Egon. **The Goal/Question/Metric Method - Apractical guide for quality improvement of Software Development**. [S. l.]: Mc Graw Hill, 1999.

SOMMERVILLE, Ian. **Software Engineering**. 9th. USA: Addison-Wesley Publishing Company, 2010. ISBN 0137035152.

SOUZA, Clarisse Sieckenius et al. Can inspection methods generate valid new knowledge in HCI? The case of semiotic inspection. **International Journal of Human-Computer Studies**, v. 68, n. 1, p. 22–40, 2010. ISSN 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2009.08.006. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1071581909001128>.

THUNG, Ferdian. API recommendation system for software development. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). [S. l.: s. n.], 2016. P. 896–899.

TSAI, Wei-Tek et al. Testability of Software in Service-Oriented Architecture. In: 30TH Annual International Computer Software and Applications Conference (COMPSAC'06). [S. l.: s. n.], 2006. v. 2, p. 163–170. DOI: 10.1109/COMPSAC.2006.167.

WATSON, Robert et al. API documentation and software community values: a survey of open-source API documentation. In: PROCEEDINGS of the 31st ACM international conference on Design of communication. [S. l.: s. n.], 2013. P. 165–174.

WOHLIN, Claes. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: PROCEEDINGS of the 18th international conference on evaluation and assessment in software engineering. [S. l.: s. n.], 2014. P. 1–10.

APÊNDICE A – Etiquetas de comunicabilidade

	Característica distintiva	**Sintomas ilustrativos**	**Etiqueta de SigniFYIng APIs**	**Etiqueta de comunicabilidade**	**IDs**
Falha total					
Usuário não entende a comunicação do designer e falha em alcançar o objetivo esperado	Usuário está ciente da falha	Usuário acredita que não consegue atingir seu objetivo e interrompe a interação	*[Falha consciente da tarefa]*	"Desisto"	T1
	Usuário não está ciente da falha	Usuário acredita que alcançou seu objetivo, embora não tenha	*[Falha inconsciente da tarefa]*	"Parece bom para mim"	T2
Falha parcial					
Usuário pode não entender ou rejeitar a interação que o designer espera. No entanto, o usuário ainda atinge seu objetivo fazendo outra coisa	Usuário entende a solução do designer, mas prefere seguir um caminho diferente	O usuário escolhe deliberadamente comunicar sua intenção com sinais secundários ou inesperados, embora tenha entendido as soluções preferíveis promovidas pelo designer	*[Declinação deliberada do recurso de design]*	"Não, obrigado"	T3
	Usuário não entende a solução do designer e segue um caminho diferente	O usuário comunica sua intenção com sinais inesperados porque não consegue ver ou entender o que o sistema está dizendo sobre melhores soluções para atingir seu objetivo	*[Declinação insuspeita do recurso de design]*	"Vai de outro jeito"	T4

Tabela 24: Etiquetas de comunicabilidade, parte 1.

	Característica distintiva	**Sintomas ilustrativos**	**Etiqueta de SigniFYIng APIs**	**Etiqueta de comunicabilidade**	**IDs**
Falha temporária					
A compreensão do usuário é temporariamente interrompida (mas ele se recupera do colapso em seguida)	Usuário não consegue encontrar a expressão apropriada para comunicar o que deseja fazer	O usuário sabe o que está tentando fazer, mas não consegue encontrar um elemento de interface que diga ao sistema para fazê-lo. Ele pode navegar por menus, abrir e fechar caixas de diálogo, etc., procurando o sinal específico que fará isso	*[Falta temporária de vocabulário]*	"Cadê?"	T5
	O usuário não percebe momentaneamente a comunicação do designer	O usuário não entende a resposta do sistema para o que ele disse para fazer. Muitas vezes, ele repete a operação cujo efeito está ausente ou não é percebido	*[Falta temporária de percepção]*	"O que aconteceu?"	T6
	Usuário não consegue encontrar uma estratégia apropriada para interação	O usuário não sabe o que fazer em seguida. Ele vagueia pela interface em busca de pistas para restaurar a comunicação produtiva com o sistema. Ele pode inspecionar menus, caixas de diálogo, etc., sem saber exatamente o que deseja encontrar ou fazer	*[Falta temporária de intenção]*	"E agora?"	T7
O usuário percebe que a interação pretendida está errada	O usuário está comunicando uma mensagem sensata, mas no contexto ou modo errado	O usuário está dizendo ao sistema coisas que seriam apropriadas em outro contexto. Ele pode tentar selecionar objetos que não estão ativos ou interagir com sinais que são apenas saídas	*[Precisa mudar de contexto]*	"Onde estou?"	T8
	O usuário comete um erro ao se expressar, mas percebe imediatamente	O usuário comete um erro instantâneo, mas o corrige imediatamente. Usar a operação "desfazer"para cancelar a ação e depois seguir outro caminho é um exemplo desse colapso comunicativo	*[Precisa mudar a expressão]*	"Epa!"	T9
	O usuário abandona uma conversa de muitas etapas que não alcançou o efeito desejado atualmente	O usuário está envolvido em uma longa sequência de operações, mas de repente percebe que esta não é a correta. Assim, abandona essa sequência e tenta outra. Esse colapso envolve uma longa sequência de ações.	*[Precisa mudar de estratégia]*	"Eu não posso fazer assim"	T10

Tabela 25: Etiquetas de comunicabilidade, parte 2.

	Característica distintiva	**Sintomas ilustrativos**	**Etiqueta de SigniFYIng APIs**	**Etiqueta de comunicabilidade**	**IDs**
Falha temporária					
O usuário procura esclarecer o significado pretendido pelo designer	O usuário solicita implicitamente informações	O usuário tenta obter uma dica rápida sobre algum sinal cujo significado não entende	*[Precisa saber]*	"O que é isso?"	T11
	O usuário solicita explicitamente informações	O usuário busca por informações detalhadas sobre algum signo cujo significado não entende (completamente). Uma interação típica é consultar a documentação ou pedir ajuda	*[Precisa aprender]*	"Socorro!"	T12
	O usuário tenta produzir explicações por conta própria	O usuário procura elaborar o significado de um signo que não entende (completamente). Isso envolve vários testes interativos atrás de uma explicação lógica para o comportamento do sistema	*[Precisa descobrir]*	"Por que não funciona?"	T13

Tabela 26: Etiquetas de comunicabilidade, parte 3.

APÊNDICE B – Mapeamento dos casos de testes

	Cenário 2 Criar um card com uma data de entrega																	
Causas	CT01	CT02	CT03	CT04	CT05	CT06	CT07	CT08	CT09	CT10	CT11	CT12	CT13	CT14	CT15	CT16	CT17	CT18
Nome vazio	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	-	-
Nome válido	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	-	-
Posição informada: top	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	-	-
Posição informada: bottom	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	-	-
Posição informada: positive float	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	-	-
Posição vazia	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	-	-
Posição inválida	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-
Data válida	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	-	-
Data vazia	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	-	-
Data inválida	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	1
Efeitos																		
Cria um card	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Cria um card sem nome	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Cria um card com nome	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
Cria um card no top	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Cria um card no bottom	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Cria um card na posição float	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
Cria um card na posição default	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0
Cria um card com data de entrega	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0
Cria um card sem data de entrega	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0

Figura 22: Tabela *verdade* construída para o cenário 2

Cenário 2	**Causa**			**Efeito**
	Vazio *Válido*	*Top* *Bottom* *Positive float*	*Válida* *Vazia* *Inválida*	
	Nome	**Posição**	**Data**	**Resultado esperado**
CT11	Válido	Bottom	Válida	Cria um card com nome na posição bottom com data de entrega
CT12			Vazia	Cria um card com nome na posição bottom sem data de entrega
CT13		Positive float	Válida	Cria um card com nome na posição informada pelo float com data de entrega
CT14			Vazia	Cria um card com nome na posição informada pelo float sem data de entrega
CT15		Vazia	Válida	Cria um card com nome na posição default com data de entrega
CT16			Vazia	Cria um card com nome na posição default sem data de entrega
CT17	-	Inválida	-	Não cria o card
CT18	-	-	Inválida	Não cria o card

Figura 23: Casos de testes relacionados ao cenário 2

	Cenário 3 Atualizar um card para deixar de seguir um card previamente criado que gera notificações de suas mudanças.						
Causas	CT01	CT02	CT03	CT04	CT05	CT06	CT07
Nome valido	0	0	0	1	1	1	-
Subscribed: true/1	1	0	0	1	0	0	-
Subscribed: false/0	0	1	0	0	1	0	-
Subscribed vazio	0	0	1	0	0	1	-
Subscribed inválido	0	0	0	0	0	0	1
Efeitos							
Atualiza o card	1	1	1	1	1	1	0
Card fica sem nome	1	1	1	0	0	0	0
Card fica com nome	0	0	0	1	1	1	0
Usuário fica inscrito no card	1	0	0	1	0	0	0
Usuário não fica inscrito no card	0	1	0	0	1	0	0
Não altera a inscrição no card	0	0	1	0	0	1	1

Figura 24: Tabela *verdade* construída para o cenário 3

Cenário 3	**Causas**		**Efeitos**
	Vazio *Válido*	*True/1* *False/0*	
	Nome	**Subscribed**	**Resultado esperado**
CT01	Vazio	True/1	Atualiza o card, ficando sem nome, e o usuário fica inscrito no card
CT02		*False/0*	Atualiza o card, ficando sem nome, e o usuário não fica inscrito no card
CT03		Vazio	Atualiza o cardo, ficando sem nome, e não altera a inscrição no card
CT04	Válido	True/1	Atualiza o card, ficando com o nome informado, e o usuário fica inscrito no card
CT05		*False/0*	Atualiza o card, ficando com o nome informado, e o usuário não fica inscrito no card
CT06		Vazio	Atualiza o cardo, ficando com o nome informado, e não altera a inscrição no card
CT07	-	Inválido	Não atualiza o card e não altera a inscrição

Figura 25: Casos de testes relacionados ao cenário 3

	Cenário 4 Encontrar o identificador da discussão.				
Causas	CT01	CT02	CT03	CT04	CT05
Course ID válido	1	1	1	-	-
Course ID vazio	0	0	0	1	-
Course ID inválido	0	0	0	-	1
Forum ID válido	1	0	0	-	-
Forum ID vazio	0	1	0	-	-
Forum ID inválido	0	0	1	-	-
Efeitos					
Retorna a lista de fóruns do curso solicitado	1	1	1	0	0
Não retorna nenhum fórum	0	0	0	1	1
Retorna a discussão solicitada	1	0	0	0	0
Não retorna discussão	0	1	1	0	0

Figura 26: Tabela *verdade* construída para o cenário 4

Cenário 4	**Causas**		**Efeitos**
	Válido *Vazio* *Inválido*	*Válido* *Vazio* *Inválido*	
	Course ID	**Forum ID**	**Resultado esperado**
CT01	Válido	Válido	Retorna a lista de fóruns do curso solicitado, e em seguida retorna a discussão solicitada
CT02		*Vazio*	Retorna a lista de fóruns do curso solicitado, e em seguida não retorna a discussão
CT03		Inválido	Retorna a lista de fóruns do curso solicitado, e em seguida não retorna a discussão
CT04	Vazio	-	A API retorna mensagem dizendo que o course ID está vazio, e não retorna nenhum fórum
CT05	Inválido	-	A API retorna mensagem informando course id inválido, e não retorna nenhum fórum

Figura 27: Casos de testes relacionados ao cenário 4

	Cenário 5 Criar comentário em uma discussão						
Causas	CT01	CT02	CT03	CT04	CT05	CT06	CT07
Discussão ID válido	1	1	1	1	-	0	0
Discussão ID vazio	0	0	0	0	-	1	0
Discussão ID inválido	0	0	0	0	-	0	1
Título da mensagem válido	1	1	1	1	0	-	-
Título da mensagem vazio	0	0	0	0	1	-	-
Corpo da mensagem string válida	1	0	0	0	-	-	-
Corpo da mensagem html válida	0	1	0	0	-	-	-
Corpo da mensagem html inválida	0	0	1	0	-	-	-
Corpo da mensagem vazio	0	0	0	1	-	-	-
Efeitos							
Postagem foi adicionada com mensagem em string	1	0	0	0	0	0	0
Postagem foi adicionada com mensagem em html	0	1	0	0	0	0	0
Postagem não foi adicionada	0	0	1	1	1	1	1

Figura 28: Tabela *verdade* construída para o cenário 5

Cenário 5	**Causas**			**Efeitos**
	Válido *Vazio* *Inválido*	*Válido* *Vazio*	*String válida* *HTML válido* *HTML inválido* *Vazio*	
	Discussão ID	**Título da mensagem**	**Corpo de Mensagem**	**Resultado esperado**
CT01	Válido	Válido	String válida	Postagem será adicionada na discussão desejada, com o título informado e a mensagem como String
CT02			HTML válido	Postagem será adicionada na discussão desejada, com o título informado e a mensagem como HTML
CT03			HTML inválido	API deverá retornar erro devido à formatação inválida do HTML, e a postagem não será adicionada
CT04			Vazio	API deverá retornar erro devido à mensagem estar vazia, e a postagem não será adicionada
CT05	-	*Vazio*	-	API deverá retornar erro devido ao título da mensagem estar vazio, e a postagem não será adicionada
CT06	Vazio	-	-	API deverá retornar erro devido ao ID da discussão estar vazio, e a postagem não será adicionada
CT07	Inválido	-	-	API deverá retornar erro devido ao ID da discussão ser inválido, e a postagem não será adicionada

Figura 29: Casos de testes relacionados ao cenário 5

APÊNDICE C – Termo de Consentimento Livre e Esclarecido (TCLE)

Aplicando técnica de investigação semiótica para melhorar a testabilidade de desenvolvimento de software.

Pesquisador Responsável: Camila Pardo Garcia Morelli

Instituição a que pertence o Pesquisador Responsável: UFF

Email: camila.morelli@id.uff.br

Nome do Voluntário:

O(A) Sr.(ª) está sendo convidado(a) a participar do projeto de pesquisa "Aplicando técnicas de investigação semiótica para melhorar a testabilidade de desenvolvimento de software.", de responsabilidade da pesquisadora Camila Pardo Garcia Morelli Esta pesquisa tem como objetivo propor uma metodologia para a equipe de testadores de software. O presente estudo de caso tem como o objetivo obter indícios sobre a utilidade da ferramenta SigniFYing API em melhorar a testabilidade de uma API. Assim sendo, o(a) sr.(ª) participará de um experimento onde poderá responder ou não perguntas que serão levantadas pelo pesquisador principal da pesquisa. Todas as perguntas e todo o assunto tratado no experimento será sobre testes e criação de testes em APIs investigando a aplicação da ferramenta SigniFYing API nesse processo e sobre a sua experiência com este tema. Este experimento será realizado de forma assíncrona no qual será disponibilizado uma apresentação em formato de vídeo explicando a metodologia a ser utilizada. O participante será incluído no grupo controle ou no grupo aplicando a metodologia em questão. Será aberto um espaço para perguntas do experimento via email e por videoconferência. Ao final será requisitado todo o seu conteúdo obtido no experimento em formato de formulário, arquivo de dados, porém nenhuma informação visual, seja em formato de vídeo, foto ou qualquer outro desta forma, será gravado em nenhum momento do experimento. Ao final, os questionamentos recebidos por email, durante as sessões de dúvidas, serão transcritos

pelo pesquisador principal para que este possa analisar os resultados do experimento. Durante o experimento, se o(a) sr.(ª) se sentir desconfortável por qualquer motivo que seja, poderá não responder ou até mesmo cancelar a participação no experimento no mesmo momento sem nenhuma necessidade de explicação a este pesquisador ou a qualquer outra parte relacionada a esta pesquisa. Caso ocorra a interrupção do experimento, todo dado coletado relacionado a ela será completamente apagado e não entrará para a análise dos dados da pesquisa.

De forma mitigar esses riscos desconfortos como cansaço, aborrecimento ou outros sentimentos durante a resposta ao questionário, utilizamos um conjunto de perguntas fechadas e abertas a fim de otimizar o tempo dos participantes, focando em questões relevantes ao nosso estudo. Além disso, todas são de cunho opcional, podendo o participante responder ou não caso não se sinta confortável. Já para falhas ou lentidão de conexão, e dúvidas sobre o armazenamento das respostas o questionário é salvo automaticamente durante o período de respostas e o envio do mesmo é feito ao pressionar o botão final de submissão (cabe ressaltar que o participante pode, também, voltar às seções anteriores e rever as respostas dadas durante o processo e, também, abandonar o questionário a qualquer momento, caso decida por não continuar respondendo); ressalta-se, porém, que se o computador for desligado ou a página do questionário for fechada antes de concluí-lo, as informações serão perdidas. Por fim, além do questionário ser respondido de forma anônima, as informações obtidas serão armazenadas de forma segura por um período de 1 ano após esta data ou até a conclusão do trabalho escrito. Toda informação coletada garantirá o anonimato dos participantes, os nomes estarão codificados na base de dados. É importante conhecer que não há benefícios diretos a curto prazo para os participantes dessa pesquisa, contudo, ao término desse estudo é esperada a seguinte contribuição: Auxílio na validação da metodologia proposta. Permitindo que equipe de testadores possam utilizam a ferramenta SAPI em processos que visem melhorar a testabilidade de um software.Para o grupo que vai utilizar a ferramenta vão ter a vantagem de aprender a metodologia e poder utilizá-la em outros cenários. Por fim, nós garantimos a confidencialidade e privacidade do(a) sr.(ª). Não iremos incluir, sob nenhuma hipótese ou circunstância, o nome ou outras informações pessoais, de qualquer participante ao qual tivemos contato para a realização desta pesquisa, mesmo que o participante tenha recusado ou desistido da entrevista. Como esta pesquisa é de participação voluntária, sem nenhum custo para o participante, seu consentimento poderá ser retirado a qualquer tempo, sem nenhuma espécie de prejuízo ou qualquer outra penalização. Além disso, esta pesquisa também não irá fornecer nenhum pagamento, em nenhuma forma, para aqueles que participarem do experimento. Portanto,

nenhum gasto do voluntário, como transporte ou alimentação, computador, serviço de internet, impressão, será provido ou ressarcido. Para sanar qualquer dúvida referente aos procedimentos, riscos, benefícios e outros assuntos relacionados com a pesquisa, basta entrar em contato com o pesquisador responsável pela forma desejada presente no topo deste termo.

Os Comitês de Ética em Pesquisa (CEPs) são compostos por pessoas que trabalham para que todos os projetos de pesquisa envolvendo seres humanos sejam aprovados de acordo com as normas éticas elaboradas pelo Ministério da Saúde. A avaliação dos CEPs leva em consideração os benefícios e riscos, procurando minimizá-los e busca garantir que os participantes tenham acesso a todos os direitos assegurados pelas agências regulatórias. Assim, os CEPs procuram defender a dignidade e os interesses dos participantes, incentivando sua autonomia e participação voluntária. Procure saber se este projeto foi aprovado pelo CEP desta instituição. Em caso de dúvidas, ou querendo outras informações, entre em contato com o Comitê de Ética da Faculdade de Medicina da Universidade Federal Fluminense (CEP FM/UFF), por e.mail ou telefone, de segunda à sexta, das 08:00 às 17:00 horas: e.mail: etica.ret@id.uff.br, tel/fax: (21) 26299189

Eu,....., declaro ter sido informado e concordo em ser participante, do projeto de pesquisa acima descrito.

Ou

Eu, ..., responsável legal por ....., declaro ter sido informado e concordo com a sua participação, no projeto de pesquisa acima descrito.

Niterói,.... ... de ..... de .....

.....,

(nome e assinatura do participante ou responsável legal)

.....

(nome e assinatura do responsável por obter o consentimento)

.....

(nome e assinatura da testemunha 1, quando for o caso)

.....

(nome e assinatura da testemunha 2, quando for o caso)

APÊNDICE D – Carta Convite

Assunto: Convite para estudo

Me chamo Camila, sou pesquisadora e estudante de mestrado na UFF e estou fazendo minha dissertação com a supervisão da professora Vânia Neves (UFF) e da professora Luciana Salgado (UFF). Venho por meio deste email convidá-lo para a participação voluntária no meu estudo de caso que é a "Aplicação técnicas de investigação semiótica para melhorar a testabilidade de desenvolvimento de software." O estudo de caso presente tem como fim garantir a validação da minha pesquisa de dissertação.

Nesse sentido: Informo que como definido por meio do termo de consentimento livre e esclarecido (TCLE), em anexo a esse email, que: A qualquer momento o participante poderá desistir da participação, tendo em visto qualquer desconforto ou impossibilidade de continuar com o experimento. A pesquisa não contém fins lucrativos, a colaboração aqui recebida é puramente de caráter acadêmico. os cuidados éticos serão garantidos aos participantes da pesquisa Para o experimento Os participantes serão divididos em 2 grupos: Ao participante alocado no grupo 1 - constituído dos participantes do grupo controle irão realizar o experimento sem o conhecimento prévio da ferramenta SigniFYing API. O participante será alocado em um dos dois grupos definidos. Aos que forem alocados no grupo 2- que será constituído dos participantes que irão aplicar a técnica da ferramenta SigniFYing API, possuirão uma breve contextualização sobre a ferramenta. Será solicitado ao participante: Do grupo 1 - responder um questionário on-line, planilha com o preenchimento das informações obtidas do método e os casos de testes elaborados. Do grupo 2 - responder o questionário on-line e os casos de teste elaborados.

Caso possa participar pedimos que indique o melhor dia/horário para sua participação. Fico à disposição para esclarecimentos.

Atenciosamente, Camila Pardo Garcia Morelli