CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Gabriel Felipe Figueiredo Vieira

RECOMENDAÇÕES DE MELHORIAS NO SOFTWARE ANATOME-AT A PARTIR DE TESTES ORIENTADOS PELA NORMA ISO/IEC/IEEE 29.119-4:2021

Timóteo

2024

Gabriel Felipe Figueiredo Vieira

RECOMENDAÇÕES DE MELHORIAS NO SOFTWARE ANATOME-AT A PARTIR DE TESTES ORIENTADOS PELA NORMA ISO/IEC/IEEE 29.119-4:2021

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientadora: Profª. Dra. Márcia Valéria Rodrigues Ferreira
Coorientador: Prof. Thiago de Sousa Goveia

Timóteo

2024

GABRIEL FELIPE FIGUEIREDO VIEIRA

RECOMENDAÇÕES DE MELHORIAS NO SOFTWARE ANATOME-AT A PARTIR DE TESTES ORIENTADOS PELA NORMA ISO/IEC/IEEE 29.119-4:2021

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 13 de fevereiro de 2025:

Documento assinado digitalmente
gov.br
MARCIA VALERIA RODRIGUES FERREIRA
Data: 21/02/2025 10:46:31-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Márcia Valéria Rodrigues Ferreira
Orientadora

Documento assinado digitalmente
gov.br
THIAGO DE SOUSA GOVEIA
Data: 21/02/2025 10:54:55-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Thiago de Sousa Goveia
Coorientador

Documento assinado digitalmente
gov.br
LUCIANO NASCIMENTO MOREIRA
Data: 21/02/2025 21:56:48-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Luciano Nascimento Moreira
Professor Convidado

Prof. Me. Douglas Nunes de Oliveira
Professor Convidado

Timóteo 2024

CÓPIA DE FOLHA DE ASSINATURAS Nº 1/2025 - DECOMTM (11.63.11)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 22/02/2025 11:35)
DOUGLAS NUNES DE OLIVEIRA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOMTM (11.63.11)
Matrícula: ###212#8

Visualize o documento original em https://sig.cefetmg.br/documentos/ informando seu número: ***1****, ano:* ***2025****, tipo:* ***CÓPIA DE FOLHA DE ASSINATURAS****, data de emissão:* ***22/02/2025*** *e o código de verificação:* ***c4d1a133f8***

Dedico este trabalho à minha mãe, Rosilene.
Sua força, amor e dedicação me inspiram
a ser a melhor versão de mim mesmo todos os dias.
Este Trabalho de Conclusão de Curso é fruto do seu apoio incondicional.

Agradecimentos

Em primeiro lugar, gostaria de agradecer a Deus, pois sem Ele nada seria possível. Foi pela Sua graça e força que consegui superar os desafios e chegar até aqui.

À minha mãe, Rosilene, o meu mais profundo agradecimento. O seu apoio incondicional, a sua crença em mim e o seu esforço incansável foram fundamentais para que este dia se tornasse realidade. Obrigado por sempre estar ao meu lado, nos momentos bons e difíceis.

À minha irmã, Isabela, por sempre acreditar em mim e por me ver como uma inspiração. O seu apoio e palavras de incentivo foram essenciais para que eu continuasse mesmo quando tudo parecia difícil.

À minha avó paterna, Maria Aparecida, que, mesmo não estando mais entre nós, sempre acreditou que este momento chegaria. As suas expectativas e o seu amor me deram forças para seguir em frente e honrar a sua memória.

Ao meu namorado, Mateus, por todo o apoio, compreensão e companheirismo. Obrigado por estar ao meu lado durante as madrugadas de dedicação a este trabalho e por acreditar em mim mesmo quando eu duvidava de mim mesmo.

Aos meus amigos e colegas, Pedro Oliveira e José Geraldo, que tornaram essa jornada mais leve e divertida. Obrigado por estarem presentes e por me ajudarem a superar os desafios ao longo do caminho.

Aos meus incríveis orientadores, Márcia e Thiago, o meu mais sincero agradecimento. Sem as suas orientações, paciência e dedicação, este trabalho não teria sido possível. Obrigado por estarem sempre disponíveis para me ajudar, mesmo diante das dificuldades que surgiram durante o processo.

Por fim, gostaria de agradecer a todas as pessoas que, de alguma forma, contribuíram para que eu chegasse até aqui. Cada gesto de apoio, cada palavra de incentivo e cada momento compartilhado foram importantes para a conclusão deste trabalho. Este momento é também de todos vocês.

Muito obrigado!

Resumo

O Anatome-AT (Authoring Tool) é um sistema web de autoria voltado ao apoio na elaboração de conteúdos por professores para o ensino de anatomia. Por ainda ser e ter recebido contribuições de diversos desenvolvedores, ajuste e melhorias no código fazem parte do processo para aprimorar a robustez do sistema. Antes de implementar essas melhorias, um passo a ser desenvolvido é a realização de testes para identificar defeitos presentes no sistema. O objetivo deste trabalho é a recomendação de melhorias para o Anatome-AT com base nos resultados de testes automatizados de robustez realizados. A identificação dos defeitos foi feita com base na Técnica de Suposição de Erros, conforme descrito na norma ISO/IEC/IEEE 29119-4, a partir da qual foram elaborados um modelo de teste e a definição dos itens de cobertura de testes, seguidos pela definição e execução dos testes. A ferramenta Cypress foi utilizada para a automação dos testes, com o objetivo de ter maior eficiência e precisão na detecção dos defeitos. Os resultados obtidos demonstraram a eficácia dos testes na identificação dos defeitos no sistema, possibilitando a elaboração de recomendações de ajustes mais assertivos, baseados em um conjunto real de testes, ainda que com um número reduzido. A combinação da Técnica de Suposição de Erros com a automação dos testes utilizando Cypress se mostrou satisfatória para atingir os resultados esperados.

Palavras-chave: testes de software, suposição de erros, ISO 29119, Cypress, Anatome-AT.

Abstract

The Anatome-AT (Authoring Tool) is a web-based authoring system designed to support teachers in creating content for anatomy education. Since it is still a prototype and has received contributions from various developers, adjustments and improvements to the code are essential to enhance the system's robustness. Before implementing these improvements, an important step is conducting tests to identify existing defects in the system. The objective of this study is to recommend improvements for Anatome-AT based on the results of automated robustness tests. The defect identification process was based on the Error Guessing Technique, as described in the ISO/IEC/IEEE 29119-4 standard, from which a test model and test coverage items were defined, followed by the design and execution of tests. The Cypress tool was used for test automation, aiming for greater efficiency and accuracy in defect detection. The results obtained demonstrated the effectiveness of the tests in identifying system defects, enabling the development of more precise adjustment recommendations based on a real set of tests, even with a limited number of cases. The combination of the Error Guessing Technique with test automation using Cypress proved to be satisfactory in achieving the expected results.

Keywords: software testing, error guessing, ISO 29119, Cypress, Anatome-AT.

Lista de ilustrações

Lista de tabelas

Lista de Códigos

Lista de abreviaturas e siglas

AT	Ferramenta de Autoria
IEC	International Electrotechnical Commission (Comissão Eletrotécnica Internacional)
IEEE	Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Eletricistas e Eletrônicos)
ISO	International Organization for Standardization (Organização Internacional de Normalização)
JSON	JavaScript Object Notation

Sumário

1 Introdução

O Anatome-AT é um sistema para apoiar professores na criação de conteúdos didáticos para o ensino de Anatomia (FERREIRA, 2019). O sistema foi desenvolvido como um protótipo para validar o modelo de ferramenta de autoria para conteúdos didáticos acessíveis, desenvolvido por Ferreira (2019). Sendo assim, o desenvolvimento do Anatome-AT não priorizou testes de software para validar a robustez da aplicação.

A palavra "teste"que teve a sua origem no termo *testum* (em Latim), que se refere a um pote de barro que era utilizado em experimentos com metais com o objetivo de medir a proporção dos seus componentes. Dessa forma, a expressão "colocar algo em teste", do inglês *to put to the test*, remete a essa prática antiga. Em outras palavras, ao aplicar testes a um software, o objetivo é identificar e analisar possíveis defeitos (HIRAMA, 2011).

Ao realizar testes em um software, o objetivo principal deve ser agregar valor ao sistema, o que se traduz em aumento de qualidade e confiabilidade. Para alcançar esse objetivo, é fundamental identificar e corrigir falhas no programa. Nesse sentido o teste não deve se limitar apenas em confirmar que o software está funcionando corretamente, mas sim partir da premissa de que ele pode conter erros - uma suposição válida para quase todos os sistemas. Assim, os testes devem ser projetados para identificar e corrigir o maior número possível de falhas (MYERS; SANDLER; BADGETT, 2012).

O processo de teste tem dois objetivos principais: primeiro, garantir que o software atenda aos requisitos especificados, seja para softwares customizados ou para softwares genéricos. Em segundo lugar, o objetivo é identificar falhas no comportamento do software, ou seja, situações em que ele funcione de maneira incorreta ou indesejável, ou ainda de forma diferente do que foi especificado (SOMMERVILLE, 2011).

Quando a documentação de requisitos é inexistente ou imprecisa, a aplicação de técnicas formais de teste pode se tornar inviável. Nesse contexto, a técnica de suposição de erros surge como uma abordagem alternativa, baseada na experiência do testador e no conhecimento de falhas comuns em sistemas semelhantes (KHAN, 2022).

A técnica de suposição de erros utiliza um modelo de teste, representado por uma lista, com possíveis defeitos para orientar os testes, identificando entradas que possam causar falhas no sistema. Essa lista pode ser baseada em erros conhecidos, registros de incidentes, experiência do testador ou conhecimento de stakeholders, como usuários e desenvolvedores (ISO/IEC/IEEE, 2021).

Diante do contexto apresentado, este trabalho tem como objetivo responder a seguinte pergunta de pesquisa: Quais as melhorias podem ser identificas no sistema Anatome-AT com a aplicação de testes usando a Técnica de Suposição de erros para orientar ajustes futuros?

1.1 Justificativa

A realização de testes, como passo necessário no processo de desenvolvimento e aprimoramento de sistemas é a justificativa base deste trabalho. Os testes são um passo importante para identificar defeitos, avaliar o comportamento do sistema e fornecer informações detalhadas que orientem futuras melhorias. Essa abordagem possibilita que as melhorias que serão realizadas, sejam fundamentadas em dados de testes reais, permitindo a correção das falhas e a evolução do sistema para uma versão mais robusta.

Os testes são uma ferramenta de controle de qualidade importante, especialmente para o sistemas que surgiram como protótipo e que para migrar para um produto precisam passar por esse processo de verificação de qualidade. E durante esse processo os testes vão permitir validar se o sistema se comporta conforme foi projetado para funcionar, além de identificarem possíveis inconsistências que possam comprometer seu funcionamento e usabilidade.

O Anatome-AT foi o sistema escolhido como item de teste, pela sua relevância social e o potencial de beneficiar os usuários quando esta tecnologia estiver disponível para eles. Essa relevância não se dá apenas de forma teórica, mas também foi validada de forma prática através de avaliações e feedbacks de potenciais usuários.

De acordo com Ferreira (2019) os modelos e requisitos elaborados e o protótipo dos sistemas foram avaliados positivamente. E que as tecnologias acessíveis que foram identificadas e modeladas são adequadas para apoiar a aprendizagem autônoma dos alunos que possuem alguma deficiência ou necessidades específicas, e também se adequam na elaboração personalizada de conteúdo pelos professores para o ensino das diversas disciplinas de Anatomia nos diferentes cursos.

Já segundo Santana (2022), em avaliações dos protótipos realizadas com participantes colaboradores do projeto, os mesmo avaliaram a solução proposta como adequada para favorecer a autonomia do professor e dos alunos no ensino remoto de Anatomia.

1.2 Objetivos

O objetivo geral deste trabalho é elaborar recomendações para orientar ajustes futuros, com base nos resultados de testes utilizando a técnica de Suposição de Erros (ISO/IEC/IEEE, 2021).

Também, objetivam-se mais especificamente:

- Fornecer uma documentação com os erros identificados no Anatome-AT, com base na técnica de Suposição de Erros (ISO/IEC/IEEE, 2022).
- Disponibilizar os casos de teste automatizados, incluindo a especificação de entradas, ações e resultados esperados.
- Fornecer uma análise dos resultados obtidos nos testes e propor recomendações para futuras melhorias com o objetivo de minimizar os erros do Anatome-AT.

2 Revisão de Literatura

Neste capítulo serão abordados assuntos e definições importantes na elaboração do desenvolvimento deste trabalho. Desta forma, a Seção 2.1 aborda os conceitos fundamentais deste trabalho, como qualidade de software, definição de erro, defeito e falha, e na Seção 2.2 são apresentados trabalhos que abordam a utilização de testes no desenvolvimento de um sistema.

2.1 Fundamentação Teórica

Nesta seção são discutidas as definições essenciais para a elaboração do desenvolvimento desta pesquisa.

2.1.1 Qualidade de Software

A gestão de qualidade abrange tanto a garantia de qualidade (QA) quanto o controle da qualidade (QC). A garantia da qualidade etá voltada para a correta aplicação dos processos e sua melhoria contínua, quanto o controle de qualidade se concentra em atividades que asseguram a obtenção dos níveis adequados de qualidade (ISO/IEC/IEEE, 2022).

De acordo com Pressman (2016), a qualidade de software pode ser entendida como a aplicação eficaz de práticas de gerenciamento de qualidade, com o objetivo de desenvolver um produto que seja útil e proporcione um valor mensurável tanto para seus desenvolvedores, quanto para seus usuários.

O padrão ISO 9216[1] foi criado com o objetivo de definir os principais atributos de qualidade para software. Segundo (PRESSMAN, 2016), o padrão estabelece seis características essenciais:

- **Funcionalidade:** mede o quanto o software atende aos requisitos especificados, considerando aspectos como adequação, precisão, interoperabilidade, conformidade e segurança.
- **Confiabilidade:** avalia a disponibilidade do software para o uso, levando em conta fatores como maturidade, tolerância a falhas e capacidade de recuperação.
- **Usabilidade:** refere-se a facilidade de uso do sistema, englobando compreensão, aprendizado e operabilidade.
- **Eficiência:** está relacionada com a otimização dos recursos do sistema pelo software, analisando o desempenho em termos de tempo de resposta e uso de recursos.
- **Manutenibilidade:** indica o quão simples é modificar ou corrigir o software, considerando a facilidade de análise, implementação de mudanças, estabilidade e testabilidade.

[1] Foi substituída pela norma ISO 25010 de 2011 (BRITTON, 2021).

- **Portabilidade:** representa a capacidade do software de ser transferido para diferentes ambientes, avaliando sua adaptabilidade, facilidade de instalação, conformidade e substituibilidade.

De acordo com Pressman (2016) a qualidade de software é avaliada por dimensões como usabilidade, intuição, eficiência, robustez e riqueza, que servem como indicadores gerais do desempenho de uma aplicação. Para uma análise prática, esses fatores são desdobrados em atributos mensuráveis. Onde:

- A **usabilidade** engloba facilidade de compreensão, aprendizagem e operabilidade.
- A **intuição** verifica se a interface segue padrões esperados, sendo acessível até para iniciantes.
- A **robustez** avalia como o software lida com erros e entradas incorretas.
- Já a **riqueza** analisa se a interface oferece recursos personalizáveis e avançados.

2.1.2 Erro (Engano), Defeito e Falha

De acordo com Sommerville (2011), os conceitos de engano, defeito, erro de sistema e falha, são definidos como sendo:

- **Erro humano ou engano:** Ações humanas que resultam na introdução de defeitos em um sistema. Por exemplo, em um sistema meteorológico para o deserto, um desenvolvedor pode decidir calcular o horário da próxima transmissão simplesmente somando uma hora ao horário atual. Esse método funciona corretamente, exceto quando a transmissão ocorre entre 23:00 e meia-noite (00:00 no formato de 24 horas).
- **Defeito no sistema:** Uma característica presente no software que pode ocasionar um erro no sistema. No caso mencionado, no item anterior, o defeito ocorre porque o código foi implementado para adicionar uma hora ao horário da última transmissão sem verificar se o horário ultrapassou às 23h.
- **Erro de sistema:** Um estado incorreto do sistema que pode gerar um comportamento inesperado para seus usuários. Quando o código com defeito é executado, o valor de horário de transmissão é definido de forma errada (ficando com 24:XX em vez de 00:XX).
- **Falha no sistema:** Um evento que ocorre quando o sistema deixa de fornecer um serviço conforme esperado pelos usuários. No exemplo citado, nenhuma informação meteorológica é transmitida devido à invalidez do horário.

2.1.3 Testes de Software

De acordo com Myers, Sandler e Badgett (2012), ao testar um programa, o objetivo principal deve ser agregar valor ao software. Esse valor é alcançado através do aumento da sua qualidade e confiabilidade, o que exige a identificação e a eliminação de erros. Nesse

contexto, o foco do teste não deve ser apenas mostrar que o programa funciona, mas partir da premissa que ele possui falhas - uma hipótese que é válida para praticamente qualquer sistema. A partir disso, os testes devem ser direcionados para identificar o maior número de erros possível.

Dessa forma, para Myers, Sandler e Badgett (2012), uma definição mais adequada para testes de software seria: "Testar é o processo de executar um programa com o propósito de identificar erros.". Ainda enfatiza que testes que identifiquem erros devem ser reconhecidos como elementos essenciais para a evolução do software, pois permitem correções e melhorias. Considerar um teste como bem-sucedido apenas por não detectar falhas demostra uma visão equivocada do processo. Um teste é verdadeiramente eficaz quando encontra erros passíveis de correção ou quando confirma, após uma análise adequada, a ausência de novos erros. Dessa forma, o insucesso está em testes que não avaliem adequadamente o programa, uma vez que a ausência de erros é uma premissa irreal em qualquer software.

O processo de teste, segundo Sommerville (2011), tem dois objetivos principais. O primeiro é garantir ao desenvolvedor e ao cliente que o software cumpre os requisitos estabelecidos. No caos de sistemas personalizados, isso exige pelo menos um teste para cada requisito documentado. Para sistema genéricos, os testes devem abranger todas as funcionalidades e combinações previstas para o produto final. O segundo objetivo é identificar comportamentos inadequados, como erros, falhas, interações indesejadas com outros sistemas, processamentos incorretos e corrupção de dados. Esses problemas geralmente são consequências de defeitos no software e devem ser eliminados.

Sommerville (2011) destaca que esses tipos de testes não são totalmente independentes: durante os testes de validação, defeitos podem ser identificados, enquanto os testes de defeitos também podem demonstrar que o sistema atende aos requisitos.

2.1.3.1 Teste de Sistema

De acordo com Naik e Tripathy (2008), o teste em nível de sistema, conhecido também como teste de sistema, tem como finalidade avaliar se a implementação atende aos requisitos definidos pelos clientes.

Segundo Hirama (2011), o teste de sistema deve ser conduzido para verificar se a especificação do sistema, incluindo hardware, sofotware, banco de dados e usuários, está sendo seguida. Esse tipo de teste é geralmenet executado no ambiente do usuário.

2.1.3.2 Testes Exploratórios

Segundo Soeiro (2023), exploratórios consistem em uma abordagem na qual o testador assume um papel ativo na investigação e avaliação do software, agindo como um "explorador"do sistema. Para isso, ele utiliza sua experiência, conhecimento do domínio e habilidades técnicas para identificar possíveis falhas e comportamentos inesperados.

De acordo com Parmar (2024), o teste exploratório é adequado para cenários específicos, como situações em que é necessário compreender rapidamente um produto ou aplicação

e fornecer um retorno ágil. Além disso, essa abordagem contribui para a avaliação da qualidade do produto sob a ótica do usuário.

2.1.4 Norma ISO/IEC/IEEE 29119

A norma ISO/IEC/IEEE 29119 define um conjunto de padrões internacionais para testes de software, sendo aplicável a organizações de diferentes portes, metologias de desenvolvimento e tipos de sistema. Ela foi desenvolvida com o objetivo de fornecer diretrizes abrangentes para gerenciar e executar testes de software em qualquer contexto (ISO/IEC/IEEE, 2022).

A norma também detalha processos relacionados ao planejamento e execução de testes, incluindo a criação de documentos de testes, a definição de métricas o gerenciamento de configuração e o suporte a gerenciamento (ISO/IEC/IEEE, 2022).

A série ISO/IEC/IEEE (2022) é composta por quatro partes principais, cada uma com um foco específico:

- **ISO/IEC/IEEE 29119-1:** Apresenta conceitos gerais sobre os quais a série é baseada, fornecendo uma introdução aos testes de software explicando sua relação com a qualidade, verificação e validação.
- **ISO/IEC/IEEE 29119-2:** Estabelece os processos de teste em diferentes níveis: organizacional, gerencial e dinâmico. Esse documento detalha o modelo de processo de teste adotado pela norma, incluindo atividades como planejamento, execução e monitoramento de testes, com foco na mitigação de riscos.
- **ISO/IEC/IEEE 29119-3:** Define modelos e exemplos de documentação de teste, como planos de teste, casos de teste e relatórios.
- **ISO/IEC/IEEE 29119-4:** Descreve técnicas de projeto e execução de testes e fornece orientações sobre como aplicar essas técnicas em diferentes cenários, ajudando a maximizar a eficiência dos testes.

Para a utilização e compreensão da técnica, ainda são apresentadas algumas definições do modelo de teste, dos itens de cobertura e dos casos de teste (ISO/IEC/IEEE, 2021).

- **Modelo de Teste:** é elaborada a partir de uma análise de possíveis erros que podem ocorrer no item de teste.
- **Itens de Cobertura de Teste:** referem-se as falhas identificas na lista de verificação, representando condições específicas que devem ser avaliadas durante os testes.
- **Casos de Teste:** são formulados com base nos itens de cobertura, selecionando um defeito específico a ser testado, determinando as entradas que o desencadeiam, definindo o resultado esperado e repetindo até que todos os itens de cobertura tenham sido verificados.

A ISO/IEC/IEEE 29119, em suma, é uma referência abrangente para a padronização de práticas de teste de software, pois fornece diretrizes que auxiliam na realização eficiente de testes, adaptando-se a diferentes contextos e promovendo a qualidade dos sistemas desenvolvidos ISO/IEC/IEEE (2022).

2.1.5 Técnica de Suposição de Erros

A técnica de suposição de erros, é uma abordagem de projeto de casos de teste baseada na experiência e intuição do testador, frequentemente complementada por listas de verificação de defeitos potenciais. Segundo Myers, Sandler e Badgett (2012), esta técnica se baseia na capacidade de identificar situações propensas a falhas em um programa, explorando erros que poderiam ser introduzidos devido a suposições equivocadas feitas pelos desenvolvedores ou ao descuido com casos específicos. Apesar de ser uma abordagem frequentemente intuitiva, a suposição de erros pode ser sistematizada para aumentar sua eficácia e abrangência.

De acordo com a ISO 29119-4 (ISO/IEC/IEEE, 2021), a aplicação da técnica deve ser guiada por um modelo de teste estruturado em listas de verificação de defeitos potenciais. Essas listas podem ser elaboradas a partir de diferentes fontes, como taxonomias de erros conhecidos, registros em sistemas de gestão de incidentes, conhecimento adquirido em testes anteriores ou informações fornecidas por stakeholders, como usuários finais e desenvolvedores. A lista de verificação serve como base para identificar entradas que podem provocar falhas no sistema, caso os defeitos estejam presentes.

O processo de derivação de casos de teste na suposição de erros pode ser descrito em etapas:

1. **Identificação do tipo de defeito:** seleciona-se um tipo de falha a ser investigado, com base na lista de verificação.
2. **Definição das entradas de teste:** determinam-se valores de entrada que poderiam desencadear o defeito escolhido.
3. **Especificação do resultado esperado:** é estabelecido o comportamento correto esperado do sistema para os valores de entradas escolhidos.
4. **Iteração:** o processo é repetido para cobrir diferentes itens da lista de verificação, garantido maior cobertura possível.

Embora a técnica de suposição de erros não possua métricas específicas para medir sua cobertura, sua eficácia depende diretamente da qualidade da lista de verificação utilizada e da experiência do testador. Com isso, a suposição de erros se apresenta como uma ferramenta flexível para identificar falhas em sistemas, sendo aplicável em diferentes contextos e níveis de testes. Sua integração com outras técnicas pode ampliar significamente a abrangência e a profundidade dos testes realizados (ISO/IEC/IEEE, 2021).

2.1.6 Testes Automatizados e o Cypress

Em geral, o teste de software demanda um alto volume de trabalho, pois muitos casos de teste são criados manualmente e, frequentemente, também são executados e analisados sem o auxílio da automação. No entanto, a adoção de ferramentas apropriadas pode reduzir significativamente o tempo necessário para essas tarefas, pois ajudam a tornar o processo de teste mais eficiente e preciso (NAIK; TRIPATHY, 2008).

A automação e execução dos testes diminui o tempo total necessário para sua realização, pois os mesmos casos de teste podem ser processados de maneira autônoma, otimizando o uso de diversas plataformas e configurações de hardware. Contudo, após a finalização dos testes, é fundamental examinar os resultados para identificar quais foram bem sucedidos ou apresentaram falhas, além de investigar as causas dos erros (NAIK; TRIPATHY, 2008).

O Cypress (CYPRESS, 2024) é uma ferramenta de automação de testes projetada para facilitar a verificação da funcionalidade de aplicações web. Ele permite a execução de testes de ponta a ponta, de componentes e de acessibilidade, oferecendo uma abordagem abrangente para garantir a qualidade das aplicações.

Uma característica fundamental do Cypress é a capacidade de execução no mesmo ciclo de eventos que a aplicação, diferentemente de outras aplicações que operam fora do navegador. Essa arquitetura permite acesso direto a elementos como DOM (Document Object Model), funções e temporizadores, facilitando o controle e a depuração dos testes (CYPRESS, 2024).

O Cypress utiliza JavaScript para a criação de testes, o que facilita sua integração com aplicações web, dado que essa linguagem é amplamente empregada no desenvolvimento de interfaces front-end. Essa abordagem permite que os desenvolvedores utilizem a mesma linguagem tanto para o desenvolvimento da aplicação quanto para a elaboração dos testes, simplificando e tornando mais eficiente o processo de produção e validação do software (SANTOS, 2024).

2.1.7 Anatome-AT

O Anatome é um projeto desenvolvido para apoiar o ensino e a aprendizagem de anatomia com foco na acessibilidade para todos os estudantes, com ou sem deficiência, com ou sem necessidades específicas. Conforme descrito por Ferreira (2019), o sistema surgiu da necessidade de prover autonomia aos estudantes que não tinham conteúdos e materiais acessíveis a eles, ficando dependentes do auxílio de outras pessoas para estudar, principalmente nos horários extra-classe. Com esse objetivo, foram modeladas tecnologias acessíveis aos estudantes com alguma necessidade específica, utilizando os mesmos recursos tecnológicos que os demais alunos. Duas dessas tecnologias são sistemas de software: o Anatome-AT, para professores, e o Anatome, sistema que tem o mesmo nome do Projeto, para estudantes.

O Anatome-AT é uma ferramenta de autoria para apoiar a elaboração de conteúdos personalizados pelos professores. Já o Anatome foi modelado para os estudantes acessarem os conteúdos elaborados usando o Anatome-AT. No contexto do trabalho de Ferreira (2019),

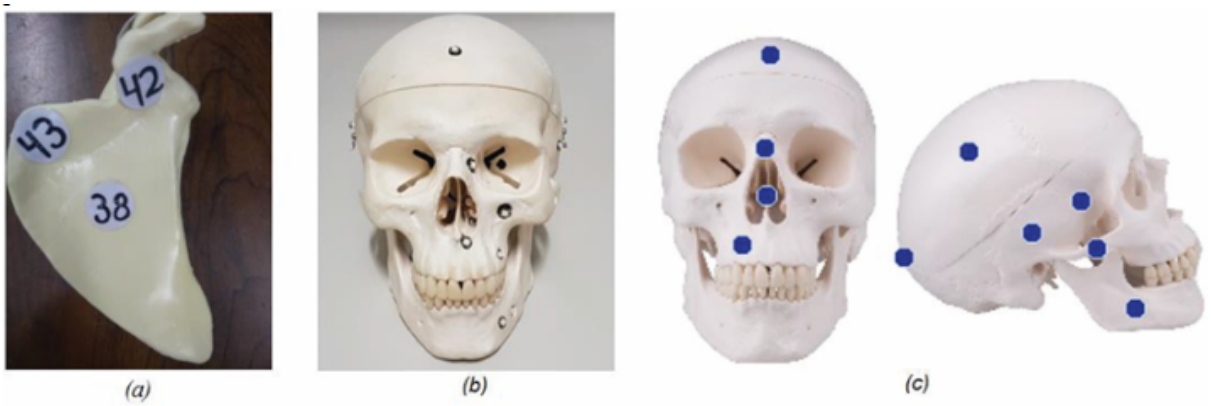
esses dois sistemas foram desenvolvidos como protótipos, para permitir aos professores e alunos de Anatomia avaliarem os modelos propostos de uma forma mais concreta. Sendo assim, o desenvolvimento dos sistemas não priorizou testes de software para validar a robustez da aplicação. Entendido o contexto do projeto Anatome, com os respectivos sistemas que o compõem, serão apresentadas algumas definições e informações sobre peças anatômicas. Na sequência, o sistema Anatome-AT, escolhido como item de teste desse trabalho, será detalhado.

As definições apresentadas abaixo foram elaboradas e apresentadas por Ferreira (2019) para facilitar a compreensão do projeto Anatome:

- **Parte anatômica**: região, estrutura ou acidente anatômico que deve ser identificado nas peças anatômicas.
- **Conteúdo prático**: nome de uma parte anatômica.
- **Conteúdo teórico**: conjunto de conhecimentos teóricos associado a uma parte anatômica.
- **Setar**: posicionar um identificador de localização em uma parte da peça anatômica.
- **Identificação anatômica**: processo de mapeamento entre o nome de uma parte anatômica e sua localização na peça.

Nas aulas de Anatomia, diferentes peças podem ser utilizadas como referência de localização das partes anatômicas que os alunos de Anatomia precisam aprender. A Figura 1 *(a)* apresenta uma peça física com etiquetas NFC indicando a localização das partes anatômicas. A Figura 1 *(b)* apresenta outra peça física com *push buttons* indicando localização das partes anatômicas. Já a Figura 1 *(c)* apresenta uma peça digital em imagem, com duas vistas (frontal e lateral), com a localização das partes anatômicas indicadas por um círculo azul.

Figura 1 – Exemplos de peças com a localização partes anatômicas identificadas



Fonte: Ferreira (2019).

O Anatome-AT, desenvolvido no contexto do trabalho de Ferreira (2019), focava em apoiar a aprendizagem de Anatomia por estudantes cegos, com baixa visão e sem deficiência. Sendo assim, as peças anatômicas utilizadas como referência de localização das partes

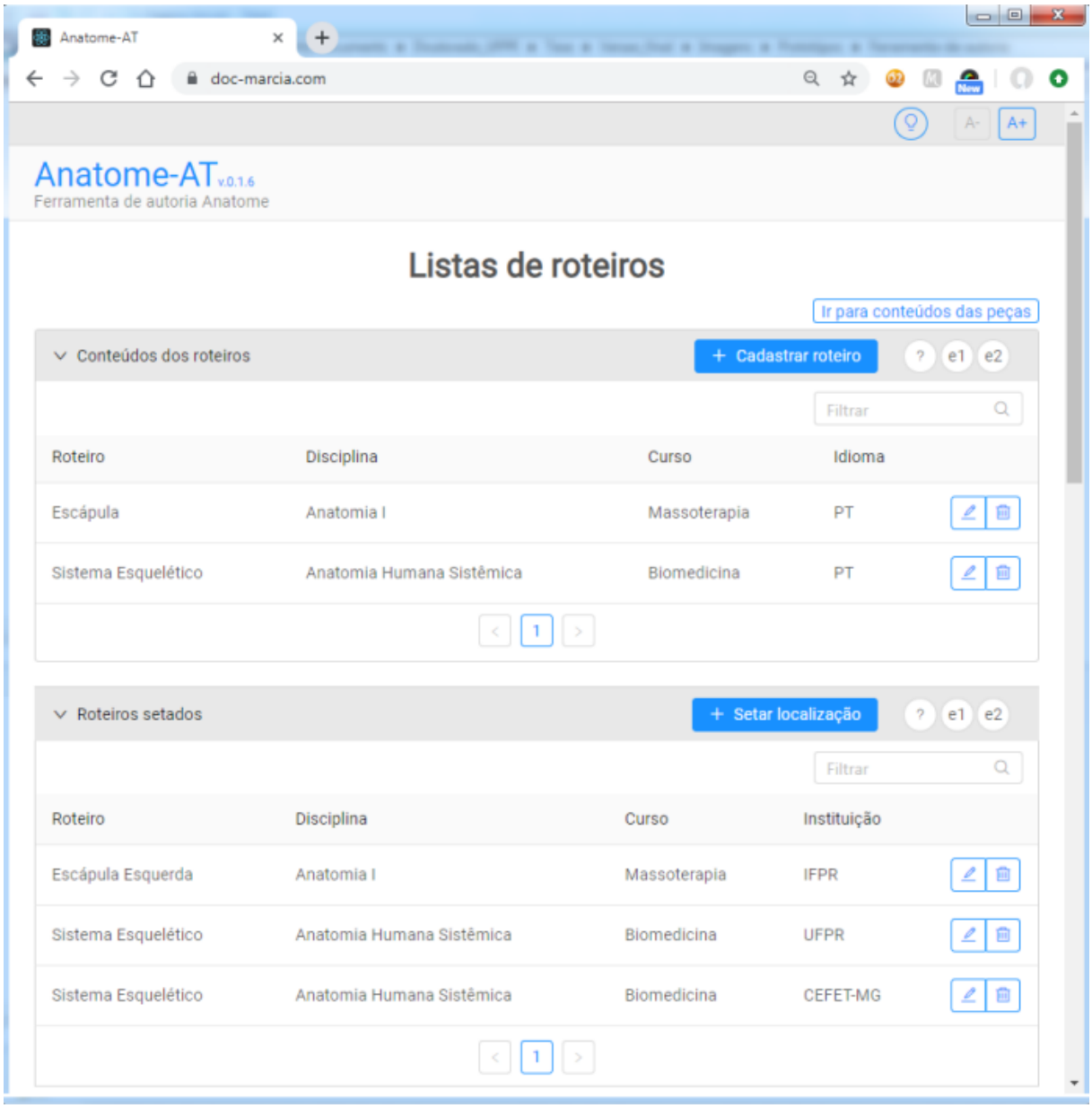
anatômicas eram peças físicas, que podem ser exploradas de forma tátil. Os identificadores das partes anatômicas eram etiquetas NFC, com o respectivo número escrito também em tinta, como a peça apresentada na Figura 1 *(a)*, e em braile. Para a numeração em braile, em cima da etiqueta NFC, coloca-se uma etiqueta adesiva transparente com o respectivo número em braile.

No trabalho de Santana (2022), o Anatome-AT foi aprimorado com foco na viabilização de atividades de ensino remoto, um contexto em que o acesso a laboratórios de Anatomia e a peças físicas pode ser limitado. O protótipo teve suas funcionalidades ampliadas por Santana (2022), permitindo a adaptação do sistema para carregar imagens digitais de peças anatômicas, como a apresentada na Figura 1 *(c)*. Desta forma, permitindo que os alunos acessem os conteúdos elaborados pelos professores mesmo sem acesso ao laboratório de Anatomia.

Nas figuras abaixo são apresentadas algumas telas dos módulos de Peça, Conteúdo do roteiro e Roteiro Setado do Anatome-AT.

A Figura 2 exibe a interface inicial do protótipo do Anatome-AT. Nessa tela, são apresentadas duas seções principais: Conteúdo dos roteiros e Roteiros setados. A seção Conteúdo dos roteiros lista os Roteiro Digitais cadastrados. Um Roteiro digital contém o conteúdo didático referente a uma unidade de ensino da disciplina Anatomia, que inclui as partes anatômicas das peças a serem estudadas, com seus respectivos nomes e conteúdos teóricos. A seção Roteiros setados lista os Roteiros de Aprendizagem Anatome. Um Roteiros de Aprendizagem Anatome contém o conteúdo de um Roteiro Digital com cada parte anatômica associada a um número posicionado em uma peça de referência de localização (FERREIRA, 2019).

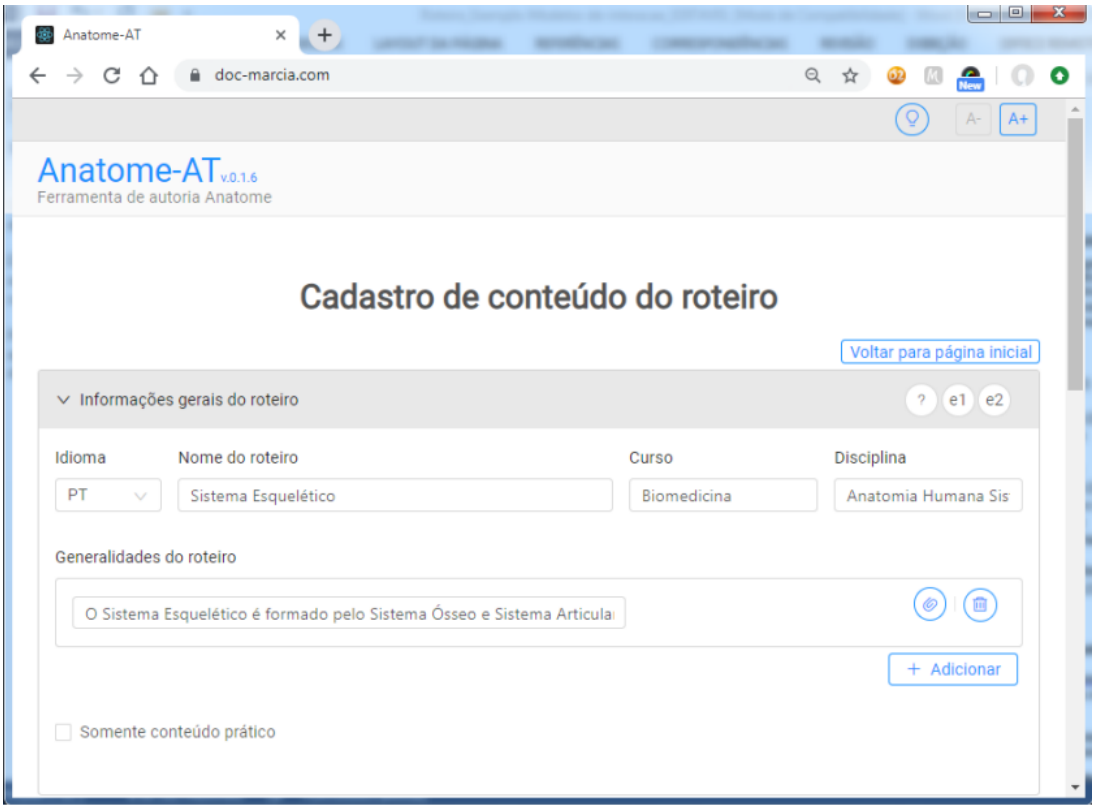
Figura 2 – Tela inicial do Anatome-AT



Fonte: Ferreira (2019).

Nas Figuras 3 a 5, são exibidas as interfaces do protótipo do Anatome-AT relacionadas à funcionalidade de "Criar Roteiro Digital". A Figura 3 apresenta os campos destinados à inserção de informações gerais sobre o Roteiro Digital, também chamadas de "Generalidades". Abaixo desses campos, há a opção "Somente conteúdo prático", que, quando ativada, permite a criação de um Roteiro Digital sem a inclusão de conhecimentos teóricos. Caso essa opção seja selecionada, os campos exibidos na Figura 5 são automaticamente desabilitados, simplificando o processo de criação para focar apenas no conteúdo prático (FERREIRA, 2019).

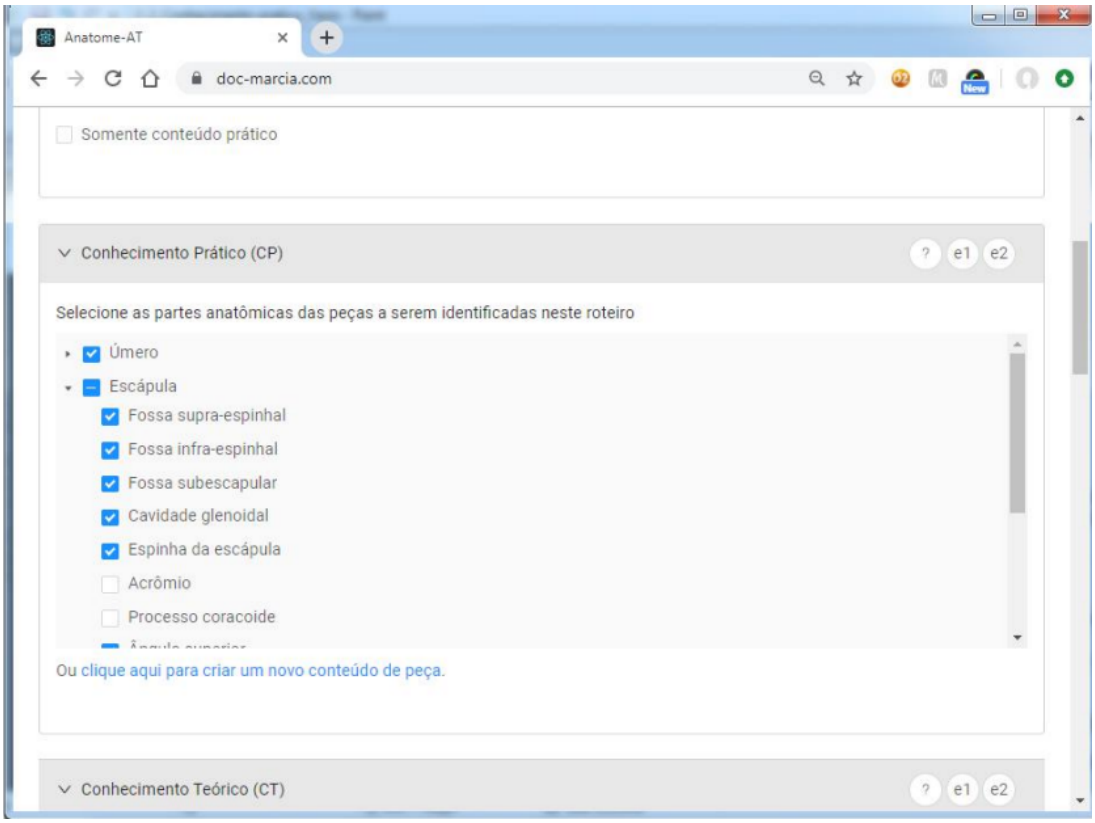
Figura 3 – Tela de Cadastro de Roteiro - Seção para inserir conhecimentos do roteiro.



Fonte: Ferreira (2019).

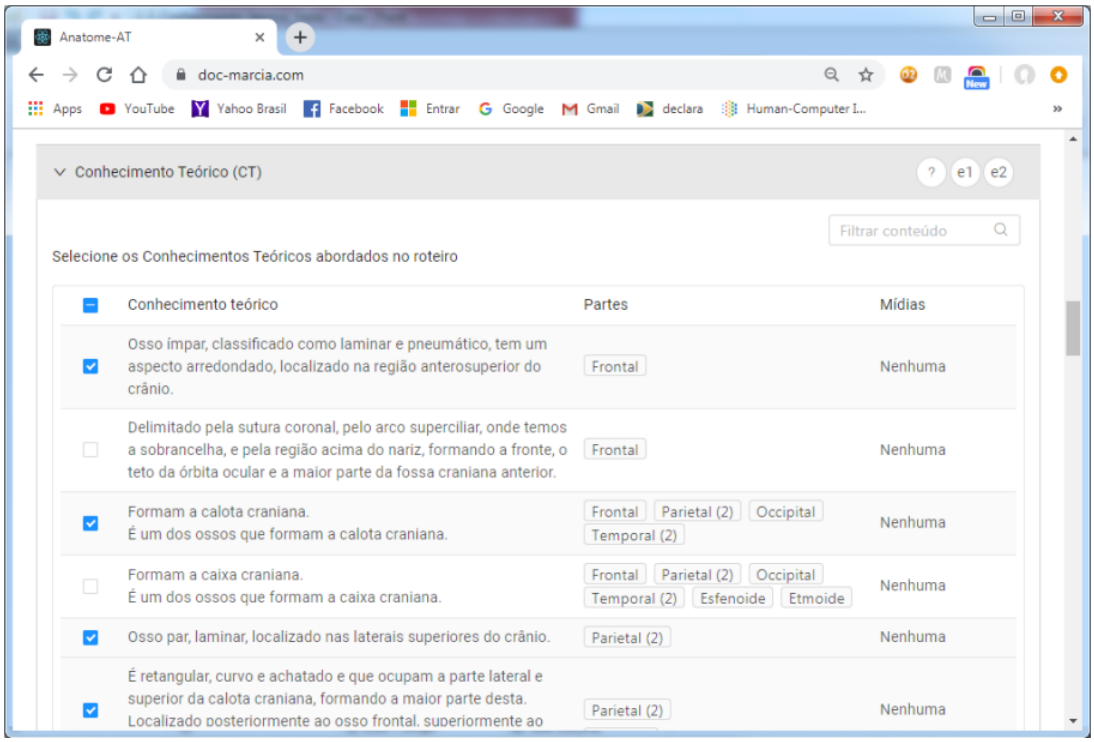
Durante a etapa de seleção das peças que farão parte do Roteiro Digital, bem como das partes anatômicas que serão identificadas em cada uma delas, é possível incluir apenas as estruturas que são relevantes para o estudo, as quais podem variar dependendo do curso. Como ilustrado na Figura 4, algumas partes da Escápula, por exemplo, foram omitidas nesse Roteiro Digital específico. Da mesma forma, na Figura 5, é possível selecionar apenas os conteúdos teóricos que serão abordados, permitindo o reaproveitamento de conhecimentos já cadastrados anteriormente. Essa flexibilidade facilita a criação de roteiros personalizados para diferentes cursos, otimizando o uso de materiais e informações já disponíveis no sistema (FERREIRA, 2019).

Figura 4 – Tela de Cadastro de Roteiro - Seleção de peças e nomes das partes anatômicas.



Fonte: Ferreira (2019).

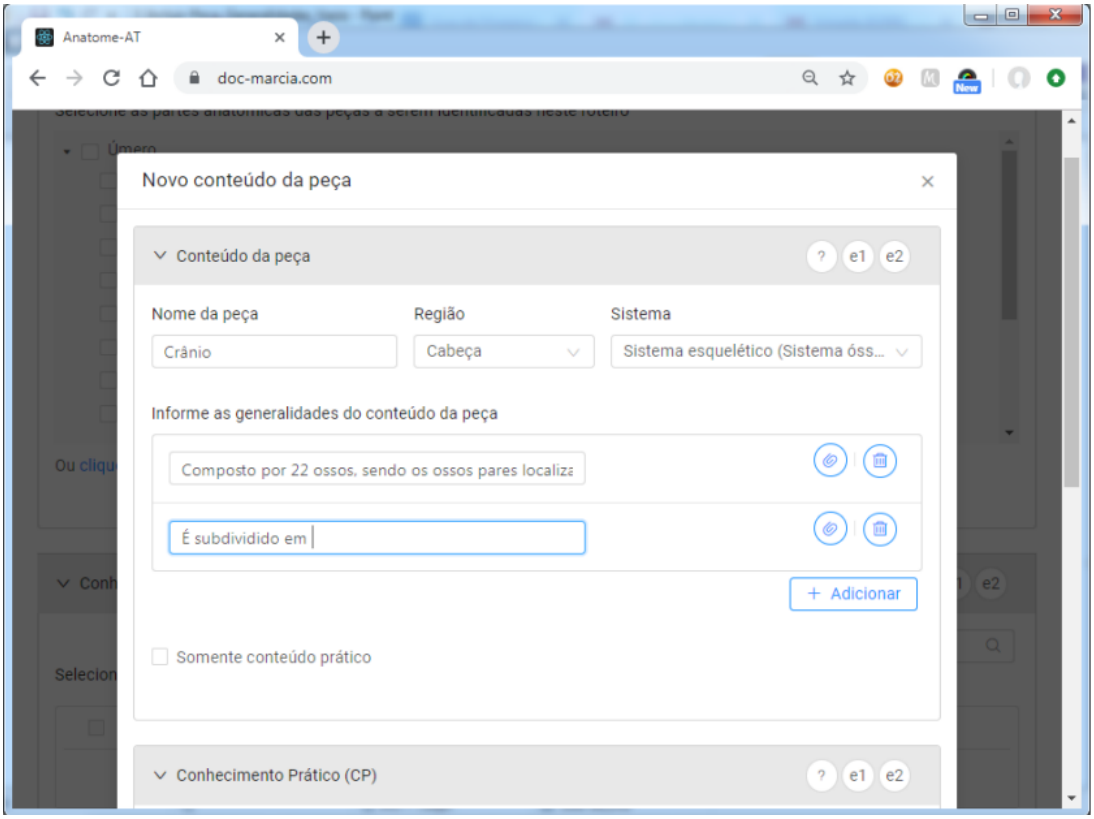
Figura 5 – Tela de Cadastro de Roteiro - Selecionar Conhecimentos Teóricos associados às partes.



Fonte: Ferreira (2019).

Os procedimentos necessários para o cadastro de uma nova peça anatômica são detalhados nas Figuras 6 a 8. A Figura 6 exibe os campos destinados à inserção das Generalidades, ou seja, informações básicas sobre a peça anatômica em questão (FERREIRA, 2019).

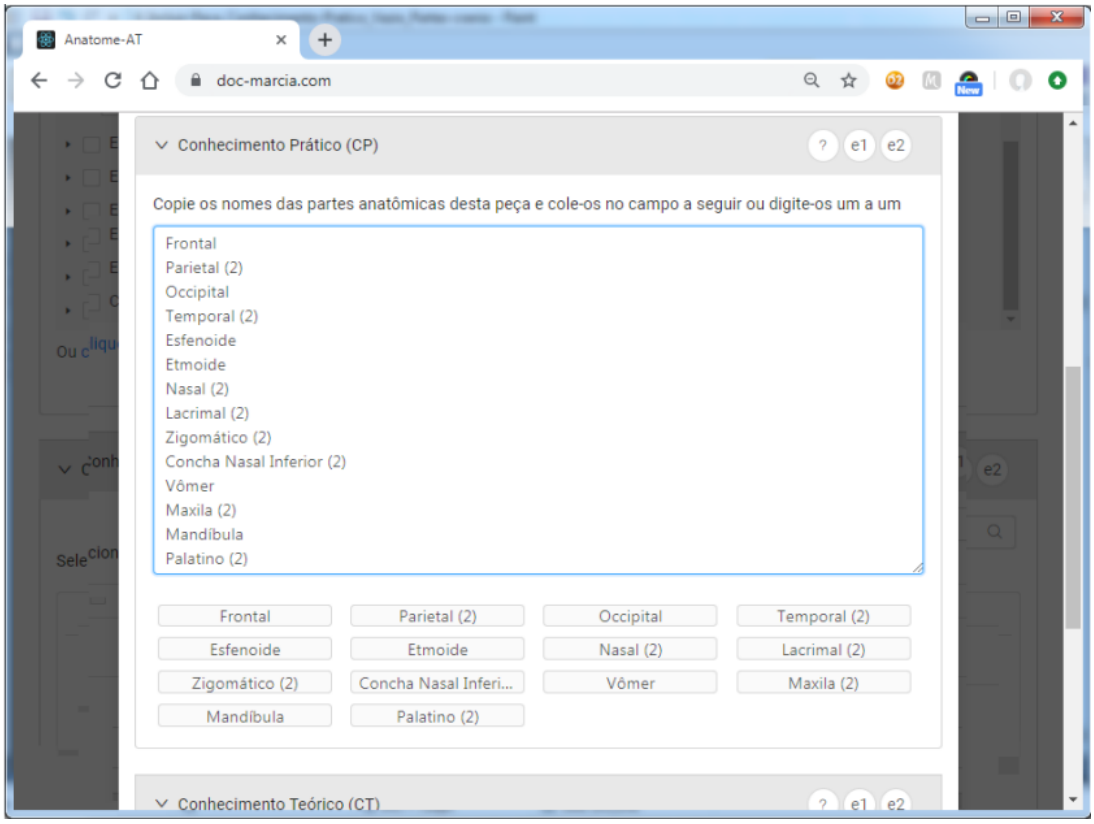
Figura 6 – Cadastro de Peça - Inserir conhecimentos da peça.



Fonte: Ferreira (2019).

Na Figura 7 é exibido um campo de texto destinado à inserção dos nomes das partes anatômicas que serão vinculadas à peça em criação. O usuário pode digitar cada nome em uma linha separada, ou copiar e colar a lista diretamente de um documento de texto, como o roteiro usado pelo professor, o que torna o processo mais ágil e eficiente. Após a inclusão dos nomes, a lista completa das partes anatômicas é exibida logo abaixo do campo de texto, permitindo uma visualização das informações cadastradas (FERREIRA, 2019).

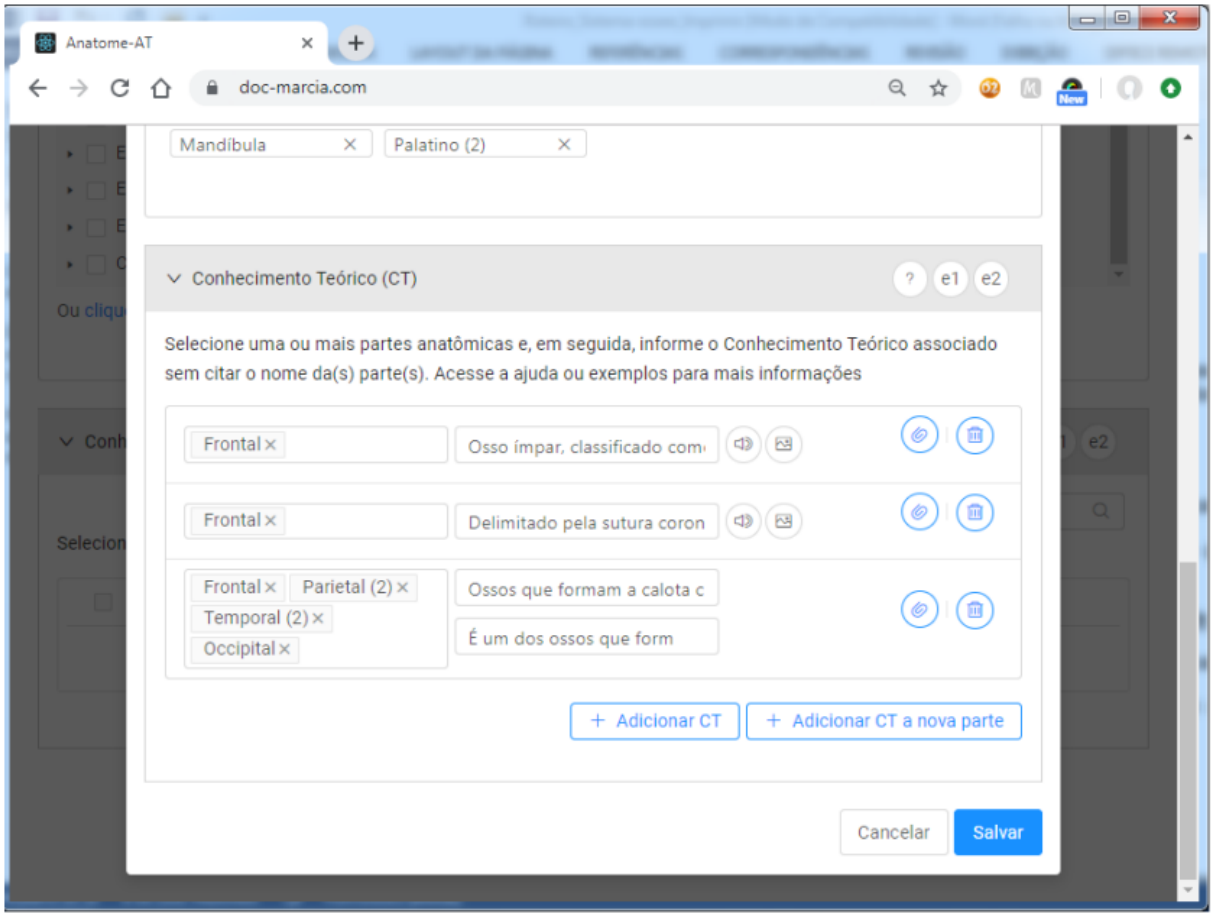
Figura 7 – Cadastro de Peça - Incluir nome das partes anatômicas.



Fonte: Ferreira (2019).

Na Figura 8, é realizada a inserção dos conhecimentos teóricos a uma parte anatômica ou a várias partes anatômicas. Observe que cada conteúdo teórico pode ser inserido de diferentes formas: por meio de texto (utilizando o campo de texto) ou anexando mídias, como áudios, vídeos e imagens, usando o ícone "Clipe", localizado ao lado do campo de texto. No exemplo ilustrado na Figura 8, os ícones exibidos após os dois primeiros campos de textos indicam que os conhecimentos teóricos relacionados às partes anatômicas estão disponíveis não apenas em formato textual, mas também em áudio e imagem, permitindo que os alunos acessem o conteúdo mais acessível de acordo com as suas necessidades específicas (FERREIRA, 2019).

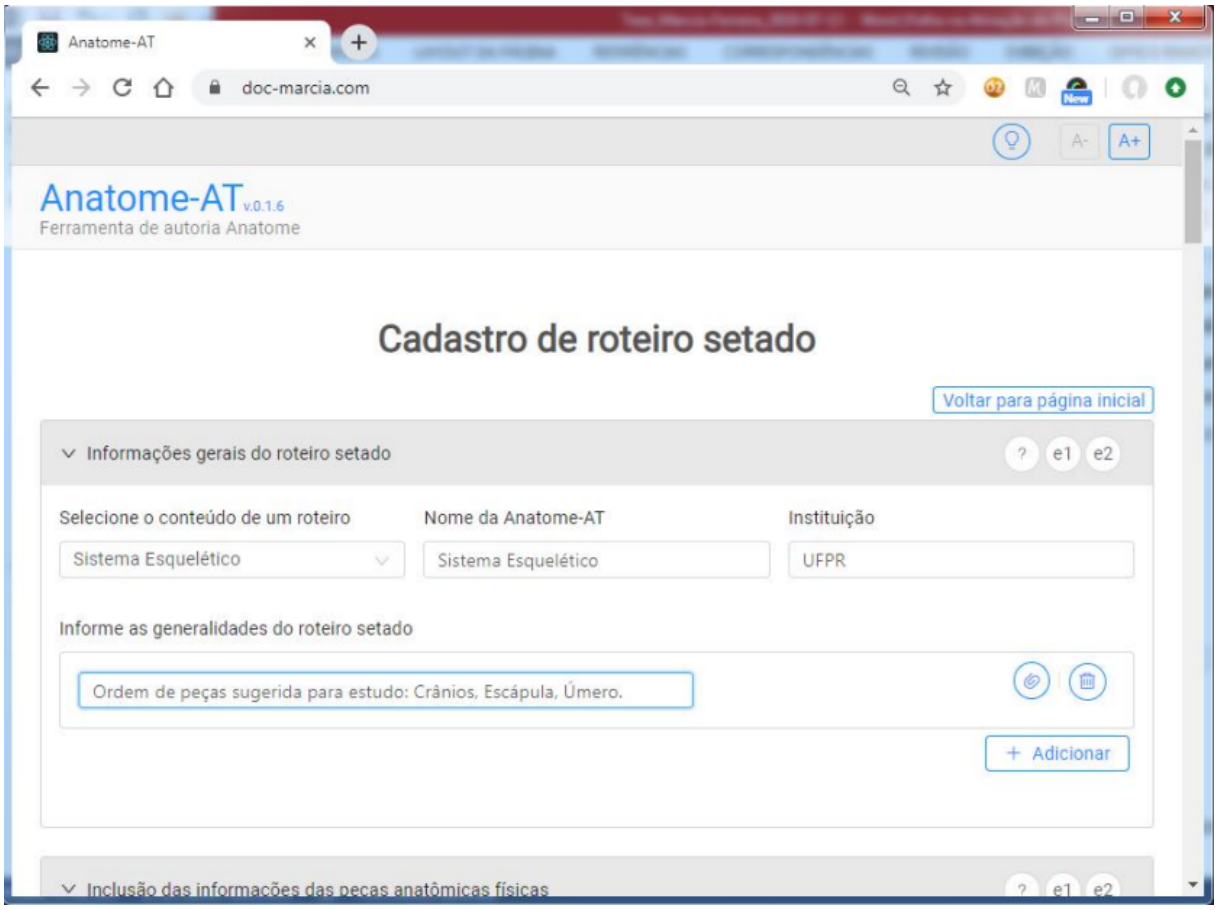
Figura 8 – Cadastro de Peça - Incluir os Conhecimentos Teóricos associados as partes.



Fonte: Ferreira (2019).

A Figura 9 exibe os campos destinados à inserção das informações gerais sobre o Roteiro de Aprendizagem Anatome, também chamadas de "Generalidades"(FERREIRA, 2019).

Figura 9 – Cadastro de Roteiro Setado - Selecionar o roteiro e inserir conhecimentos do roteiro de aprendizagem.



Fonte: Ferreira (2019).

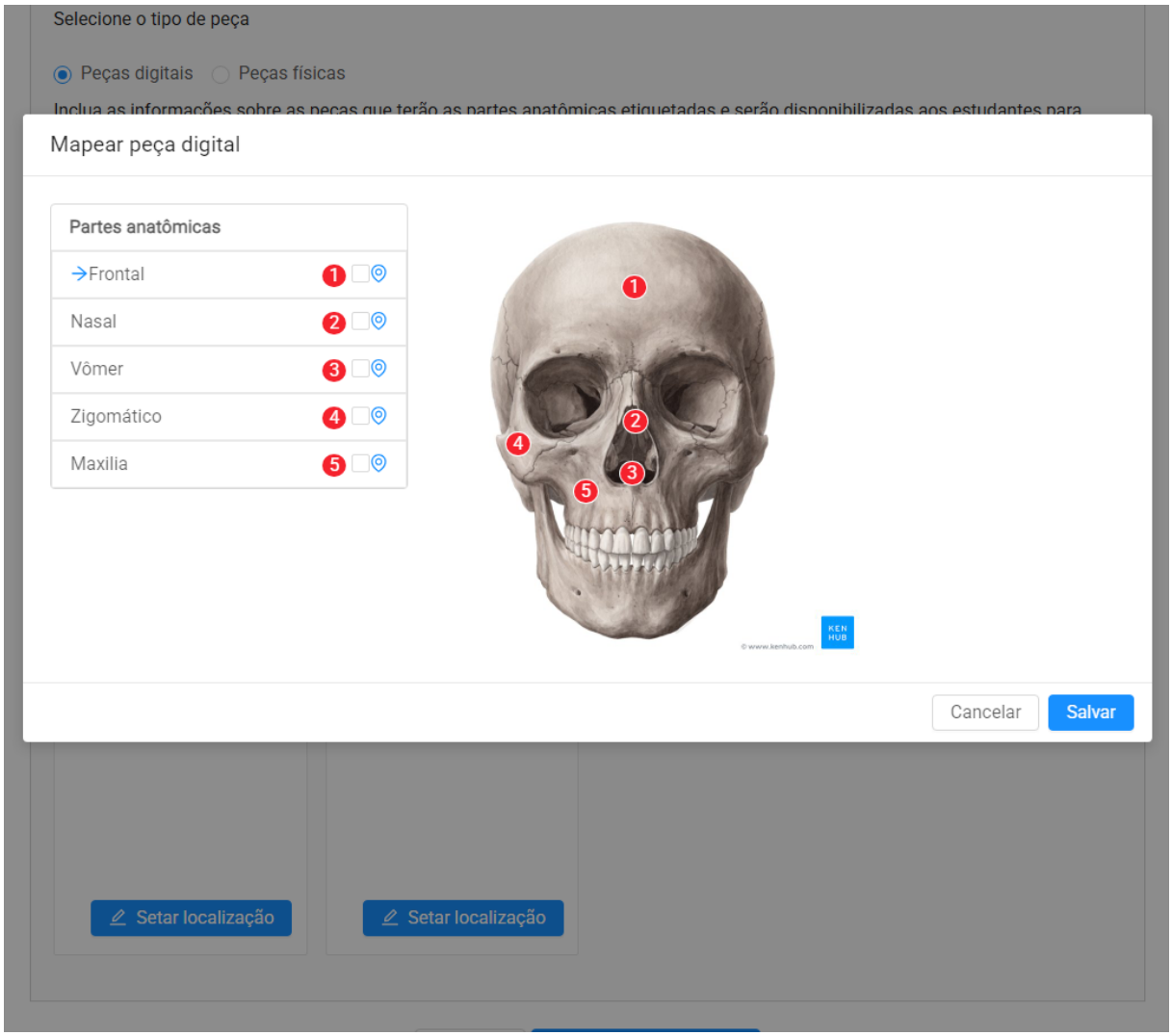
Na Figura 10 são exibidos os campos para inclusão do "Nome da Peça"física ou digital que será usada como referencia de localização (para colocar os números para indicar a localização da parte anatômica na peça); o campo "Conteúdo da peça correspondente", para seleção da peça do Roteiro digital que tem o nome das partes a serem associados aos números; e a descrição da peça para referência de localização. Quando a peça para referência de localização for digital, utiliza-se o ícone de "Clipe"à direita para carregar a imagem da peça.

Figura 10 – Cadastro de Roteiro Setado - Selecionar peças para referência de localização.

Fonte: O Autor.

Na Figura 11 pode-se observar a funcionalidade de mapear uma peça digital. O professor pode mapear o nome das partes anatômicas da peça do roteiro digital com a imagem da peça para referência de localização carregada por ele. Cada parte mapeada é identificada por uma etiqueta vermelha contendo um número. Também é visível um checkbox para cada parte anatômica, que quando selecionado permite fazer o mapeamento de uma parte anatômica de forma referenciada (Figura 12).

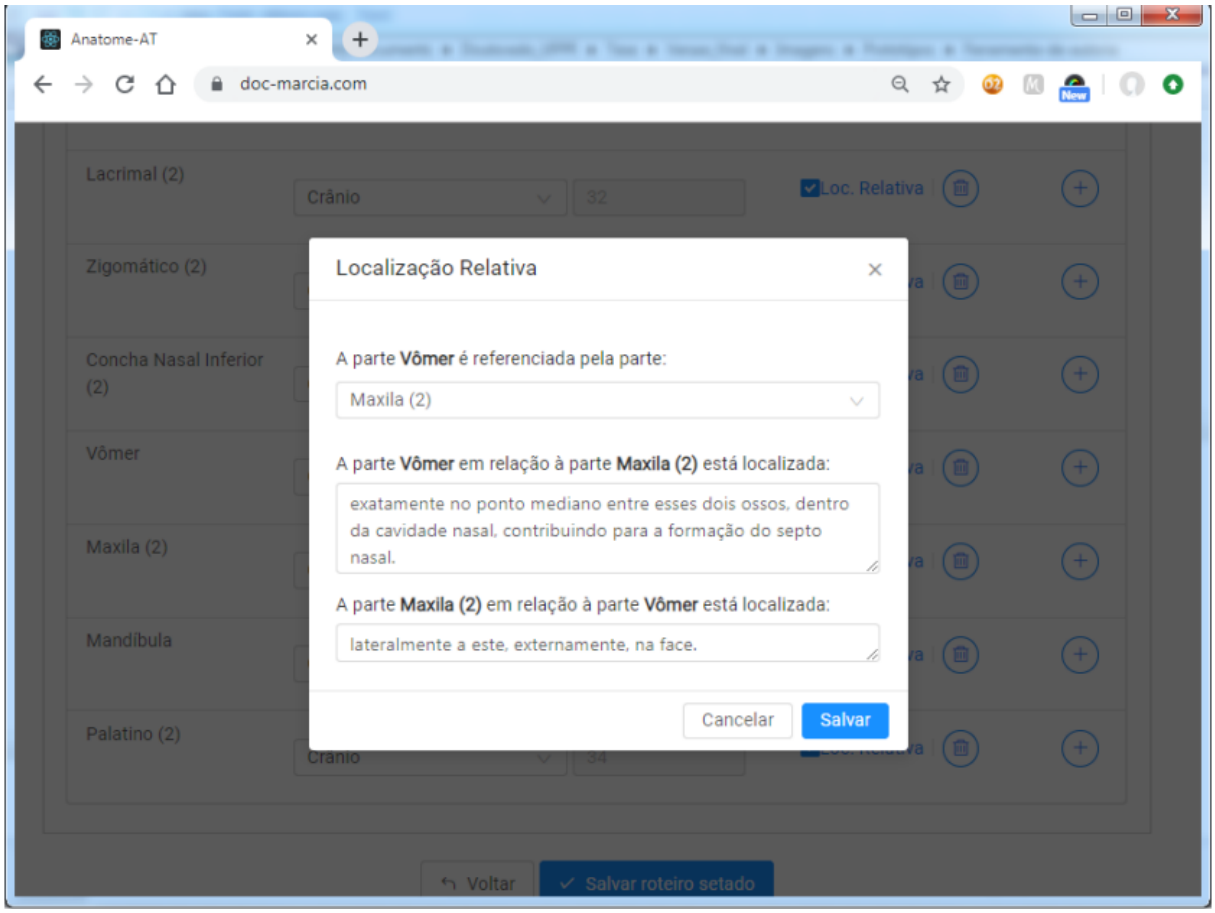
Figura 11 – Cadastro de Roteiro Setado - Associação entre Nome e Localização das partes anatômicas em uma peça digital.



Fonte: O Autor.

Para realizar o mapeamento das Partes Referenciadas, é necessário ativar a opção "Loc. Relativa"associada à parte anatômica em questão, conforme ilustrado na Figura 11. Essa ação abre a tela exibida na Figura 12, onde é possível inserir informações sobre a localização relativa da parte anatômica referenciada em relação à parte que a referencia, e vice-versa. No exemplo apresentado, a parte Vômer é referenciada pela parte Maxila, demonstrando como as relações espaciais entre as estruturas anatômicas podem ser definidas de maneira clara e precisa no sistema (FERREIRA, 2019).

Figura 12 – Cadastro de Roteiro Setado - Informação de Localização Relativa.



Fonte: Ferreira (2019).

Ao finalizar o mapeamento do nome das partes das peças de um roteiro digital aos números indicando a localização destas partes na peça para referência de localização conclui-se a elaboração de um Roteiro de aprendizagem Anatome. E, então, o Roteiro de aprendizagem Anatome poderá ser acessado pelos estudantes utilizando o sistema Anatome.

2.2 Trabalhos Correlatos

Esta seção apresenta trabalhos acadêmicos relacionados ao objetivo deste estudo, que é realizar testes para orientar futuras refatorações.

2.2.1 Trabalhos Acadêmicos Relacionados

Os trabalhos correlatos identificados relatam diferentes abordagens do uso de testes com foco em aprimorar qualidade de sistemas. Em Meger (2024), é discutida a aplicação de testes automatizados utilizando o método BDD (Behavior Driven Development) em conjunto com as ferramentas Cypress e Cucumber. O trabalho destaca as vantagens de integrar essas tecnologias no desenvolvimento de sistemas front-end, como a redução do tempo de execução dos testes em comparação com testes manuais e a melhoria na comunicação entre equipes

técnicas e não técnicas. O trabalho apresenta exemplos práticos da automação para identificação de falhas e a entrega contínua de novas funcionalidades.

Já Gabriel (2024) apresenta um estudo de caso sobre a aplicação de testes estruturais no Sistema de Reserva de Salas (SRS) da UFERSA. O trabalho foca na identificação de defeitos e na melhoria da qualidade do sistema, utilizando análise estatística com ferramentas como PMD e SonarLint, além de testes automatizados com JUnit. Os resultados evidenciam as inconsistências no código, especialmente na validação de entradas, e apontam vulnerabilidades que impactam a integridade do sistema. O trabalho conclui que a realização dos testes estruturados é fundamental para garantir conformidade com padrões de desenvolvimento e atender às expectativas dos usuários.

Em Ramos (2024) apresenta a proposta de implementação de testes unitários automatizados no jogo GiroJampa, voltado para condicionamento físico e reabilitação de pessoas usuárias de cadeiras de rodas. O trabalho utiliza o framework NUnit, integrado à Unity, e a ferramenta Coverlet para análise de cobertura de código, permitido identificar lacunas no teste das funcionalidades do jogo. Os testes foram aplicados após refatoração das classes, abordando desde operações básicas até fluxos mais complexos, com o objetivo de verificar a eficácia do código e garantir sua qualidade.

Como resultados, Ramos (2024) destaca uma cobertura significativa nas classes refatoradas, evidenciando como os testes contribuíram para a identificação de problemas e para a melhoria do processo de desenvolvimento. Além disso, os testes funcionaram como uma documentação dinâmica, facilitando a manutenção e o entendimento do sistema. A trabalho conclui que a integração de testes unitários é essencial para assegurar a robustez e a evolução contínua do GiroJampa, permitindo a adição de novas funcionalidades sem comprometer as já existentes.

Na revisão dos trabalhos correlatos, observou-se que embora o trabalho de Meger (2024) compartilhe o uso do Cypress, o objetivo principal do trabalho é a melhoria do fluxo de desenvolvimento por meio da integração entre especificações e testes, enquanto este trabalho utiliza o Cypress como ferramenta para validar erros no Anatome-AT, orientando refatorações futuras.

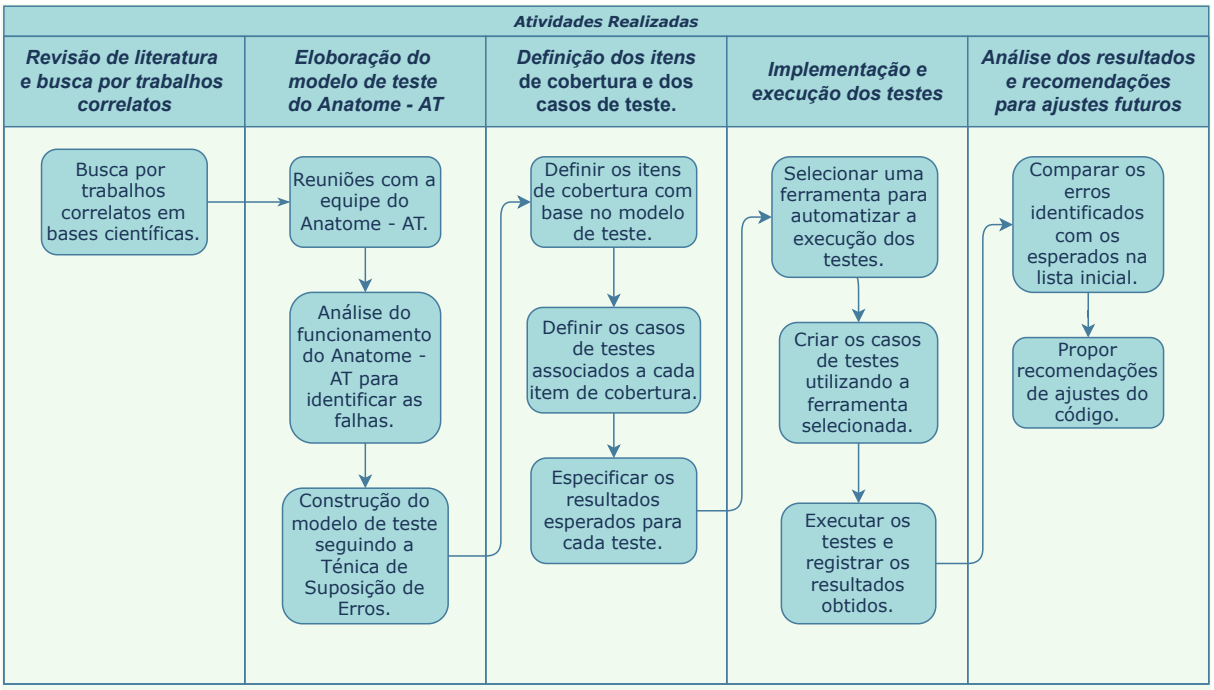
Já no trabalho de Gabriel (2024), o autor emprega ferramentas como PMD e SonarLint para detectar padrões de código problemáticos e JUnit para testes automatizados. Diferentemente deste estudo, que utiliza testes automatizados e a técnica de Suposição de Erros, Gabriel prioriza a análise estática para garantir conformidade com boas práticas de desenvolvimento.

Por fim, no trabalho de Ramos (2024), o autor faz uso do NUnit para validar a funcionalidade do código dentro do ambiente da Unity, garantindo maior cobertura de testes. A principal diferença está na abordagem: enquanto Ramos (2024) foca na validação da funcionalidade de um jogo, este estudo adota uma estratégia baseada na Suposição de Erros para identificar falhas em um sistema educacional e fornecer diretrizes para melhorias no sistema.

3 Procedimentos metodológicos

Este capítulo apresenta os procedimentos metodológicos adotados neste trabalho. Para facilitar a compreensão do método e a visualização da dinâmica das atividades, a sequência das etapas realizadas é ilustrada no fluxograma da Figura 13

Figura 13 – Fluxograma dos Procedimentos Metodológicos realizados.



Fonte: O autor.

3.1 Revisão de literatura e busca por trabalhos correlatos

Primeiro foi realizada uma revisão de literatura sobre conceitos (Seção 2.1) de Qualidade de Software, os conceitos de Erro (Engano), Defeito e Falha, sobre Testes de Software, a norma ISO/IEC/IEEE 29119 (ISO/IEC/IEEE, 2022), a Técnica de Suposição de Erros (ISO/IEC/IEEE, 2021) e também sobre Testes Automatizados e o Cypress (CYPRESS, 2025), pois foi a ferramenta escolhida para automação dos testes. Também foi feita uma busca por trabalhos correlatos.

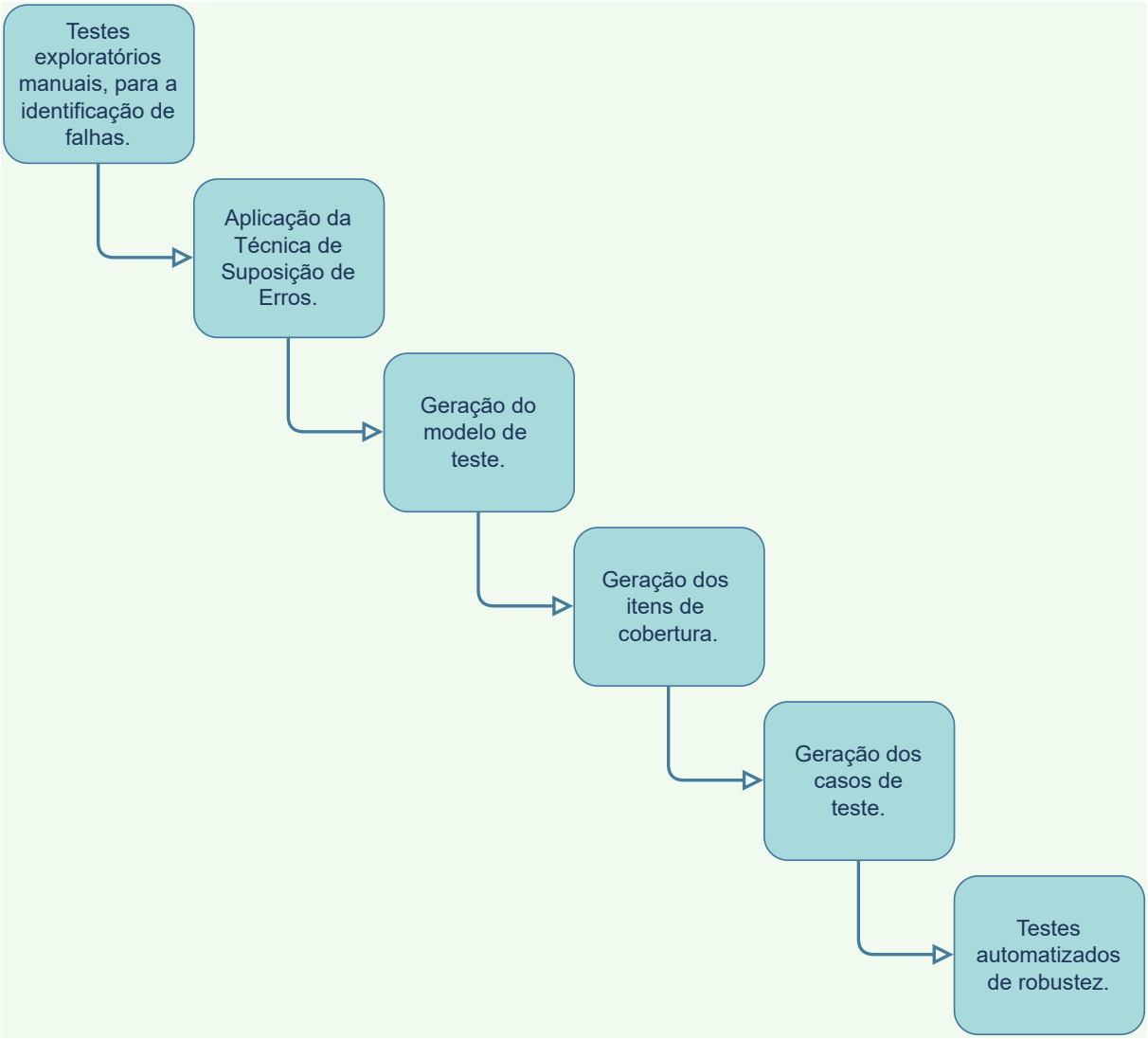
Para encontrar trabalhos acadêmicos com propostas semelhantes a este, foram utilizadas como fontes de pesquisas as bases ACM Digital Library, Google Scholar e Periódicos CAPES. Foi feita uma filtragem por trabalhos em português e inglês publicados nos últimos cinco anos, que contêm os termos “teste” ou “identificação de erros” no título. A String de busca utilizada está disponível no Apêndice A.

3.2 Etapas realizadas para criação dos testes

Para proporcionar uma visão clara do processo seguido na definição e execução dos testes no sistema, foi elaborado o fluxograma a seguir (Figura 14). Nele, são ilustradas todas as etapas realizadas, desde os testes exploratórios até a execução dos testes automatizados.

Com o objetivo de facilitar o entendimento do caminho adotado para definição e execução dos testes no sistema, o fluxograma abaixo (Figura 14) apresenta as etapas que foram realizadas para atingir esse objetivo.

Figura 14 – Fluxograma das etapas realizadas para criação dos testes.



Fonte: O autor.

3.3 Elaboração do modelo de teste

Com o objetivo de identificar falhas no Anatome-AT para a criação do modelo de teste, em primeiro momento foram realizados testes exploratórios manuais (Seção 2.1.3.2), a nível de sistema (Seção 2.1.3.1), com o objetivo de compreender o software e identificar falhas. No início estes testes foram realizados de maneira independente, mas posteriormente eles

aconteceram junto à equipe[1] do Anatome-AT, para aproveitar o conhecimento e a experiência prévia com sistema.

Os testes abrangeram os módulos de Peça, Roteiro e Roteiro Setado do Anatome-AT, com foco específico nos formulários de cada módulo. Na seção 2.1.7, são apresentadas imagens retiradas do sistema referentes a esses módulos.

Após o fim das verificações manuais no sistema e discussões com a equipe do Anatome-AT, foi criado um modelo de teste que contém uma lista dos defeitos encontrados no sistema., seguindo Técnica de Suposição de Erros (Seção 2.1.5), conforme descrita na ISO 29119-4 (Seção 2.1.4). Esta técnica foi escolhida, pois o Anatome-AT é um sistema que carece de documentações formais de requisitos do sistema, descrições de casos de uso e das funcionalidades. E a Técnica de Suposição de Erros (ISO/IEC/IEEE, 2021) tem como base a experiência do testador na identificação dos erros, e com isso não exige uma documentação formal.

A Tabela 1 apresenta uma amostra dos defeitos identificados no modelo de teste, relacionando cada defeito ao seu respectivo local de ocorrência no sistema. A versão completa do modelo de teste pode ser consultada na Tabela 3 da Seção 4.1.1.

Tabela 1 – Defeitos Identificados e Seus Locais de Ocorrência

Defeito	**Local de Ocorrência**
O sistema não valida se os campos obrigatórios no cadastro de peças foram preenchidos com apenas espaços em branco.	Cadastro de conteúdo da peça (Figura 6).
O sistema não valida se os campos destinados ao nome da peça e aos conhecimentos práticos foram preenchidos com apenas um caractere.	Cadastro de conteúdo da peça (Figura 6).
O sistema não garante que ao mapear uma peça digital e clicar no checkbox de "Localização Relativa", se o modal for aberto e, em seguida, cancelado, o mapeamento da parte anatômica selecionada é perdido.	Cadastro de roteiro setado -> Associação entre o nome e a localização da parte na peça (Figuras 11 e 12).

Fonte: O autor.

3.4 Definição dos itens de cobertura e dos casos de teste

Com base no modelo de testes elaborado na Seção 3.3, nesta etapa, foram definidos os itens de cobertura (Seção 2.1.5). Estes itens foram desenvolvidos com o objetivo de avaliar como o sistema se comporta mediante entradas ou ações inválidas.

Cada item buscou cobrir um defeito apontado no Modelo de Testes (Tabela 3 - Seção 4.1.1) e foram identificados com um código único no formato ICXXX, onde:

- **IC** representa "Item de Cobertura de Teste";
- **XXX** é um número sequencial que identifica os cenários de entradas e ações no sistema.

[1] A equipe do Anatome-AT é composta pelo responsável do desenvolvimento dos modelos do sistema e pelo desenvolvedor que implementou os protótipos da aplicação.

Na lista a seguir, são apresentados alguns dos itens de cobertura definidos para o Anatome-AT. A versão completa desses itens pode ser consultada na Seção 4.1.2.

- **IC001** - Digitar espaços em branco para o nome da peça e conhecimento prático.
- **IC002** - Digitar apenas um caractere para o nome e conhecimento prático.
- **IC003** - Excluir um conhecimento prático que já foi mapeado e verificar se ele foi removido do mapeamento.

Seguindo a Técnica de Suposição de Erros (Seção 2.1.5), após definir os itens de cobertura, os casos de teste foram desenvolvidos com o objetivo de cobrir os defeitos apontados.

Para a identificação dos casos de teste, uma nomenclatura semelhante a utilizada nos itens de cobertura foi definida. Para os casos de teste, a identificação segue o padrão CTXXX, onde *CT* significa “Caso de Teste” e *XXX* é o identificador numérico único.

As nomenclaturas adotadas, tanto para os itens de cobertura, quanto para os casos de teste, foram definidas com o objetivo de facilitar a organização e rastreabilidade dos testes.

Para cada caso de teste, foram definidas as entradas, as ações realizadas, os resultados esperados e o IC (Item de Cobertura) associado a cada CT (Caso de Teste).

A Tabela 2, apresenta os casos de teste CT001 a CT003, que foram desenvolvidos para cobrir o item de cobertura IC001, que busca validar entradas preenchidas apenas com espaços em branco para os campos obrigatórios *Nome da Peça* e *Conhecimento Prático* no formulário de peças (Figuras 6 e 7). As entradas que possuem o símbolo — atribuído à elas, significa que essas entradas foram preenchidas com espaços em branco.

As tabelas contendo todos os casos de teste desenvolvidos, podem ser visualizadas na Seção 4.1.3.

Tabela 2 – Casos de Teste (CT001 a CT003)

CT001	
Entradas	Nome da Peça: — Conhecimento Prático: ‘Ventrículo’, ‘Átrio’
Ações Realizadas	Cadastrar uma peça com o campo Nome da Peça e Conhecimento Prático preenchidos com espaços em branco.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC001
CT002	
Entradas	Nome da Peça: "Coração" Conhecimento Prático: —

Ações Realizadas	Cadastrar uma peça com o campo Conhecimento Prático preenchido com espaços em branco.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC001
CT003	
Entradas	Nome da Peça: — Conhecimento Prático: —
Ações Realizadas	Cadastrar uma peça com os campos Nome da Peça e Conhecimento Prático preenchidos com espaços em branco.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC001

Fonte: O autor.

3.5 Implementação e execução dos testes

Nesta etapa foram implementados e executados os testes para validar os erros identificados no Anatome-AT. A ferramenta escolhida para a automação dos testes foi o Cypress (CYPRESS, 2025), na versão 13.15.1, utilizada em conjunto com o Node.js 20.14.0 (ONPENJS, 2025). Os testes foram executados no navegador Google Chrome (GOOGLE, 2025) na versão 132.

O Cypress (Seção 2.1.6) foi a ferramenta escolhida para automatizar a execução dos testes, pois possibilita que os testes sejam executados no mesmo ciclo de eventos do sistema. Isso facilita o acesso à elementos do sistema, permitindo maior controle sobre os testes e aos comportamentos do sistema. E por também utilizar o JavaScript como linguagem para criação dos testes, tornaram, para o autor, que possui conhecimento da linguagem, mais fáceis a escrita dos casos de teste.

O Cypress oferece diversos comandos que facilitam o uso do framework para automação de testes em aplicações web. Entre os principais destacam-se os seguintes (SANTOS, 2024):

Para localizar elementos na interface, os comandos mais utilizados incluem:

- `cy.get()`: permite selecionar elementos com base em identificadores.
  Exemplo: `cy.get('[data-cy=name]')`.
- `cy.contains()`: localiza elementos com base em seu conteúdo textual.
  Exemplo: `cy.contains("Cadastrar")`.

Para interagir com elementos da aplicação, podem ser utilizados comandos como:

- `type`: utilizado para inserir texto em campos.
  Exemplo: `cy.get('[data-cy=name]').type("Joaozinho");`.
- `click`: realiza cliques em elementos.
  Exemplo: `cy.get(locators.CADASTRO.btnCadastrar).click();`.
- `select`: usado para selecionar valores em elementos de lista suspensa.
  Exemplo: `cy.get(Loc.CADASTRO.campoUF).select("RS");`.
- `check` e `uncheck`: aplicam ou removem seleção de checkboxes.
  Exemplo: `cy.get('[type="checkbox"]').check();`.

Além disso, o Cypress oferece ferramentas para realizar asserções, permitindo validar os resultados esperados dos testes:

- `should`: realiza verificações sobre os elementos selecionados.
  Exemplo:
  `cy.get(locators.MENU.nomeUsuarioLogado).should("contain.text", "Joaozinho");`.
- `expect`: valida propriedades em objetos de resposta.
  Exemplo: `expect(response.body).has.property("id");`.

Cada caso de teste definido foi implementado seguindo a estrutura do Cypress para automatizar a execução dos testes. os casos de teste implementados buscaram validar se o sistema se comporta como o esperado em casos de entradas inválidas.

Para facilitar a identificação e seleção de de alguns elementos na interface e reduzir a dependência por seletores baseados em classes ou ids dinâmicos, o atributo `data-cy` foi adicionado.

O código 3.1 apresenta a implementação do caos de teste CT010. O objetivo desse teste é verificar se, ao editar um roteiro já setado, o sistema permite adicionar uma nova peça e associar um novo conteúdo a ela (Figura 10). O teste começa abrindo o último roteiro setado por meio da função `cy.abrirUltimoRoteiroSetado()`. Em seguida, ele simula o clique no botão de adicionar peça (`btn-adicionar-peca` e verifica se o campo de seleção de conteúdo da peça (identificado por `[data-cy*="select-conteudo-peca"]`) está habilitado. A função `.should('be.enabled')` verifica se o último elemento encontrado com o seletor está habilitado para associar um novo conteúdo.

Código Fonte 3.1 – Caso de Teste CT010.

```
it ('CT010 - Ao editar um roteiro setado deve permitir a adição
   de uma nova peça e associar um novo conteúdo a ela.', () => {
    cy.abrirUltimoRoteiroSetado();

    // Clica no botão de adicionar peça
    cy.getElement('btn-adicionar-peca').click();
```

```
    // Recupera todos os elementos que comecem com select-
       conteudo-peca
    // e verifica se ele está habilitado
    cy.get('[data-cy^="select-conteudo-peca"').last().should('be.
       enabled');
});
```

Fonte: O autor.

Os códigos fontes completos dos testes implementados estão disponibilizados no Apêndice B, permitindo uma consulta detalhada de como cada caso foi implementado.

Com as implementações finalizadas, os testes foram executados, utilizando o Cypress, com o objetivo de observar o comportamento do sistema com base nas entradas definidas para os casos de testes (Seção 4.1.3).

3.6 Análise dos resultados e recomendações para ajustes futuros

Com a finalização da execução dos testes, os resultados foram analisados e comparados com as saídas esperadas dos casos de testes definidos na seção 4.1.3. Onde foi possível visualizar através do Cypress e também das definições dos casos, se o sistema teve o comportamento esperado considerando as entradas especificadas.

Com base na análise, foram elaboradas recomendações de ajustes para o código do Anatome-AT. Essas recomendações forma feitas com o objetivo de corrigir os defeitos identificados através de alterações no código do sistema para que o sistema passe a reproduzir o comportamento esperado que foi definido nos casos de testes. E também para contribuir com o aumento da robustez (Seção 2.1.1) do sistema.

4 Resultados

Neste capítulo, é apresentada a documentação da implementação dos testes, que inclui o Modelo de Teste (Seção 4.1.1), os Itens de Cobertura (Seção 4.1.2) e os Casos de Teste (Seção 4.1.3), consolidando todas as definições utilizadas no desenvolvimento dos testes deste trabalho.

Além disso, são apresentados os resultados obtidos com a execução dos testes automatizados de robustez no nível de sistema do Anatome-AT. Esses resultados abrangem a análise do comportamento do sistema (Seção 4.2) diante dos casos de teste definidos na Seção 4.1.3 e as recomendações de ajustes (Seção 4.3) com base nas observações realizadas.

4.1 Documentação dos testes implementados

Nesta seção, apresenta-se a documentação dos testes implementados seguindo a Norma ISO 29119-4 (ISO/IEC/IEEE, 2021). Essa documentação inclui o Modelo de Teste (Seção 3.3), os Itens de Cobertura e os Casos de Teste (Seção 3.4). Essa documentação consolida todas as definições utilizadas no desenvolvimento dos testes executados neste trabalho.

4.1.1 Modelo de Teste

Nesta seção é apresentada a Tabela 3, que contém o modelo de teste resultante da Seção 3.3. Esse modelo organiza os defeitos identificados no Anatome-AT, associando cada defeito ao seu local de ocorrência no sistema.

Tabela 3 – Modelo de teste criado para o Anatome-AT.

Defeito	**Local de Ocorrência**
O sistema não valida se os campos obrigatórios no cadastro de peças foram preenchidos com apenas espaços em branco.	Cadastro de conteúdo da peça (Figura 6).
O sistema não valida se os campos destinados ao nome da peça e aos conhecimentos práticos foram preenchidos com apenas um caractere.	Cadastro de conteúdo da peça (Figura 6).
O sistema não garante que ao mapear uma peça digital e clicar no checkbox de "Localização Relativa", se o modal for aberto e, em seguida, cancelado, o mapeamento da parte anatômica selecionada é perdido.	Cadastro de roteiro setado -> Associação entre o nome e a localização da parte na peça (Figuras 11 e 12).
O sistema não garante que excluir um conhecimento prático que já foi mapeado em um roteiro também o remove do mapeamento.	Alteração do conteúdo da peça (Figura 6).
O sistema permite a exclusão de um conhecimento prático que já foi mapeado.	Alteração do conteúdo da peça (Figura 6).
O sistema permite adicionar uma nova peça na edição de um roteiro previamente setado, mas não permite associar um conteúdo à peça adicionada.	Alteração de roteiro setado -> Inclusão das peças anatômicas (Figura 10).
O sistema não permite a edição de peças durante o processo de criação de um roteiro.	Cadastro de conteúdo do roteiro (Figura 3).
O sistema não informa ao usuário que, ao sair durante o processo de criação ou edição de um roteiro, as informações já preenchidas serão perdidas.	Cadastro de conteúdo do roteiro (Figura 3).
O sistema não seleciona automaticamente, para o roteiro, uma peça que foi cadastrada durante o processo de criação desse mesmo roteiro.	Cadastro de conteúdo do roteiro (Figura 3).

Fonte: O autor.

4.1.2 Itens de Cobertura

Nesta seção, são apresentados dos itens de cobertura elaborados com base na Seção 3.4. Abaixo é apresentada a lista dos itens de cobertura que forma definidos com o objetivo de avaliar como o sistema se comporta diante de entradas ou ações inválidas.

- **IC001** - Digitar espaços em branco para o nome da peça e conhecimento prático.
- **IC002** - Digitar apenas um caractere para o nome e conhecimento prático.
- **IC003** - Excluir um conhecimento prático que já foi mapeado e verificar se ele foi removido do mapeamento.
- **IC004** - Clicar em excluir um conhecimento prático e verificar se é exibida uma mensagem impedindo a exclusão pelo fato de o conhecimento já ter sido mapeado.

- **IC005** - No mapeamento de uma peça digital, ao clicar em uma parte mapeada para referenciá-la a partir de uma localização relativa, e posteriormente cancelar a ação, o sistema deve garantir que o mapeamento da parte permaneça intacto.
- **IC006** - Selecionar um roteiro setado para edição, incluir uma nova peça e verificar se é possível associar um conteúdo a nova peça.
- **IC007** - Começar o cadastro de um novo roteiro, clicar no botão de voltar e confirmar se o sistema exibe um alerta informando que as informações não salvas serão perdidas e se o usuário deseja mesmo voltar.
- **IC008** - No cadastro de um roteiro validar se o sistema permite editar o conteúdo de uma peça dentro do cadastro.
- **IC009** - Dentro do cadastro de um roteiro, iniciar o cadastro de uma nova peça, ao salvar o sistema deve selecionar a peça automaticamente para aquele roteiro.

4.1.3 Casos de Teste

Esta seção apresenta os casos de teste desenvolvidos com base nos itens de cobertura descritos na Seção 4.1.2. As entradas que possuem o símbolo — atribuído a elas indicam que essas entradas foram preenchidas com espaços em branco.

Tabela 4 – Casos de Teste (CT001 a CT003)

CT001	
Entradas	Nome da Peça: — Conhecimento Prático: ‘Ventrículo’, ‘Átrio’
Ações Realizadas	Cadastrar uma peça com o campo Nome da Peça e Conhecimento Prático preenchidos com espaços em branco.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC001
CT002	
Entradas	Nome da Peça: "Coração" Conhecimento Prático: —
Ações Realizadas	Cadastrar uma peça com o campo Conhecimento Prático preenchido com espaços em branco.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC001
CT003	
Entradas	Nome da Peça: — Conhecimento Prático: —

Ações Realizadas	Cadastrar uma peça com os campos Nome da Peça e Conhecimento Prático preenchidos com espaços em branco.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC001

Fonte: O autor.

Tabela 5 – Casos de Teste (CT004 a CT006)

CT004	
Entradas	Nome da Peça: 'C' Conhecimento Prático: 'Frontal', 'Nasal'
Ações Realizadas	Cadastrar uma peça com o nome contendo apenas um caractere.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC002
CT005	
Entradas	Nome da Peça: 'Crânio - Face' Conhecimento Prático: 'F', 'N'
Ações Realizadas	Cadastrar uma peça com os conhecimentos práticos contendo apenas um caractere.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC002
CT006	
Entradas	Nome da Peça: 'C' Conhecimento Prático: 'F', 'N'
Ações Realizadas	Cadastrar uma peça com o nome e os conhecimentos práticos contendo apenas um caractere.
Saída Esperada	Exibir uma mensagem de erro pedindo para o usuário verificar os erros de validação.
Item de Cobertura de Teste	IC002

Fonte: O autor.

Tabela 6 – Casos de Teste (CT007 a CT010)

CT007	
Entradas	Nome da Peça: Crânio Face Nome da Peça no Roteiro: Crânio - Face Conhecimento Prático: Maxila

Ações Realizadas	Clicar no botão de excluir um conhecimento prático que já foi mapeado e validar se foi excluído dos mapeamentos relacionados.
Saída Esperada	Espera 'false', indicando que o conhecimento não foi encontrado em um mapeamento.
Item de Cobertura de Teste	IC003
CT008	
Entradas	Nome da Peça: Crânio Face Nome da Peça no Roteiro: Crânio - Face Conhecimento Prático: Zigomático
Ações Realizadas	Clicar no botão de excluir um conhecimento prático e verificar se ele está em algum mapeamento.
Saída Esperada	Mensagem de erro informando que o conhecimento está mapeado e não pode ser excluído antes de remover as relações.
Item de Cobertura de Teste	IC004
CT009	
Entradas	Parte anatômica referenciada: Frontal
Ações Realizadas	Abrir um roteiro setado, abrir o modal de mapeamento, clicar em uma parte mapeada para referenciar e cancelar a ação.
Saída Esperada	O mapeamento ser mantido.
Item de Cobertura de Teste	IC005
CT010	
Entradas	Nome do botão: "+ Peça Física"
Ações Realizadas	Abrir o último roteiro setado cadastrado, adicionar uma nova peça e associar um novo conteúdo à peça.
Saída Esperada	O campo de conteúdo estar habilitado para associar um conteúdo à peça.
Item de Cobertura de Teste	IC006

Fonte: O autor.

Tabela 7 – Casos de Teste (CT011 a CT013)

CT011	
Entradas	Nome do roteiro: Anatomia da Cavidade Oral - Estudo da Mandíbula Curso: Odontologia Disciplina: Anatomia Odontológica I Peça associada: Mandíbula
Ações Realizadas	Preencher as informações do roteiro e clicar em voltar.

Saída Esperada	Um alerta informando que os dados não salvos serão perdidos.
Item de Cobertura de Teste	IC007
CT012	
Entradas	Nome do Roteiro: Estudo da anatomia do Crânio
Ações Realizadas	Abrir a edição ou cadastro de um roteiro e tentar editar o conteúdo de uma parte anatômica.
Saída Esperada	Abrir o modal contendo o formulário de peça com as informações da peça selecionada.
Item de Cobertura de Teste	IC008
CT013	
Entradas	Nome da Peça: Coração Região: Tronco Sistema: Sistema circulatório Generalidades: vazio Somente conteúdo prático: Não Partes Anatômicas: Átrios, Ventrículos Conhecimento Teórico: Átrios - São as duas câmaras superiores do coração , que recebem sangue do sistema circulatório e o enviam para os ventrículos. Ventrículos - São as duas câmaras inferiores, responsáveis por bombear o sangue para os tecidos do corpo.
Ações Realizadas	Abrir o formulário de cadastro de um roteiro e cadastrar uma nova peça.
Saída Esperada	A peça ter sido selecionada automaticamente para o roteiro.
Item de Cobertura de Teste	IC009

Fonte: O autor.

4.2 Resultados da análise da execução dos testes

A execução dos testes no Cypress mostrou que, dos 13 testes realizas, nenhum teve sucesso na execução. Isso mostra que para todas as entradas definidas nos casos de testes (Seção 4.1.3), o sistema não teve os comportamentos esperados que foram definidos como saídas esperadas nos casos de teste.

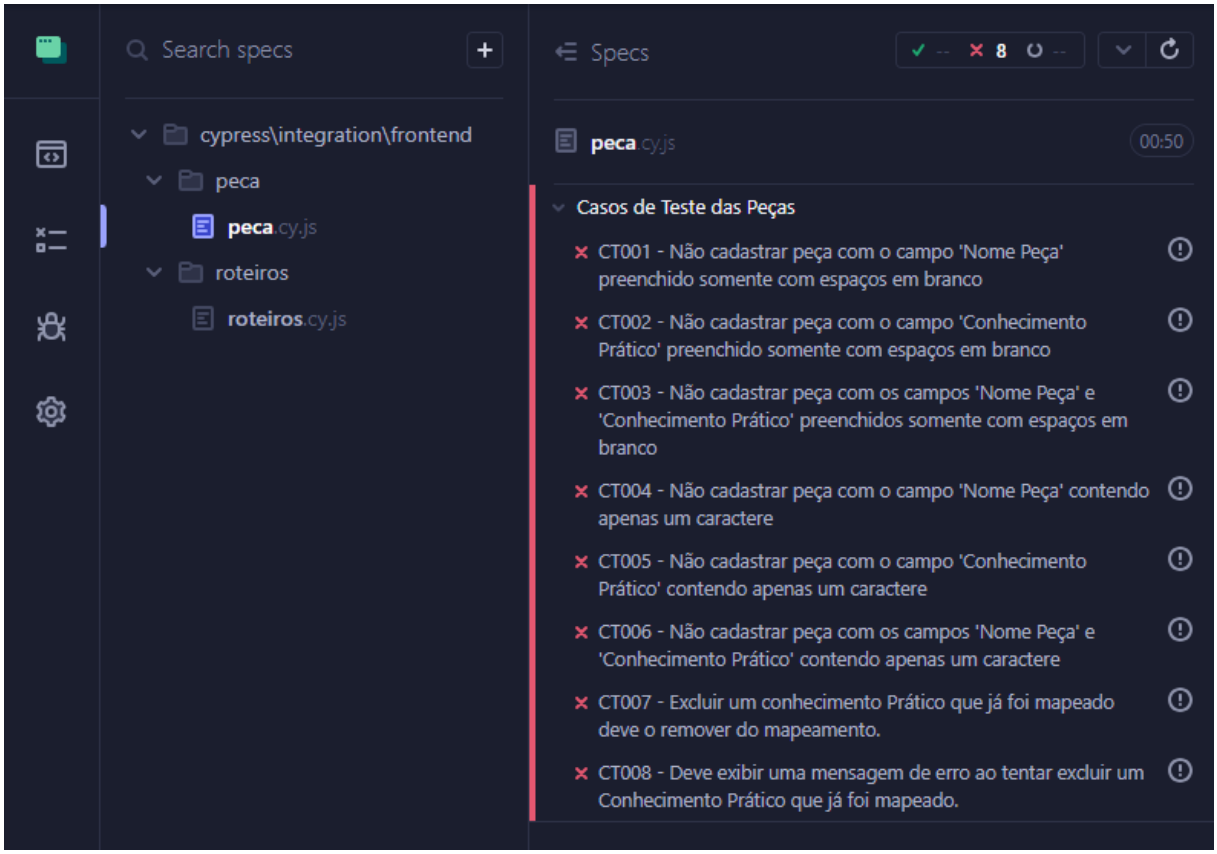
As imagens a seguir (Figuras 15 e 16) mostram os resultados das execuções dos testes para os casos relacionados às peças e aos roteiros. Como citado anteriormente, nenhum dos itens de teste tiveram sucesso e isso pode ser observado nas imagens, onde todos os casos falharam.

Os casos de teste foram implementados com o objetivo de verificar se o sistema se comporta como o esperado. Para o caso de teste CT001, por exemplo, que o intuito era verificar

se o sistema trata entradas inválidas, no caso espaços em branco, para o campo obrigatório Nome da Peça. Como resultado, esperava-se uma mensagem de erro, informando que existem erros de validação relacionados ao campo Nome da Peça, mas como observado no resultado da execução dos testes, o sistema não realiza essa validação e permite que seja cadastrada uma peça com o nome vazio.

Essa mesma forma de análise pode ser estendida aos demais casos de teste. E esses resultados indicam que o sistema precisa de ajustes para a correção dessas falhas, com o objetivo de tornar o sistema mais robusto, em relação aos casos verificados.

Figura 15 – Resultado dos casos de testes da peça.



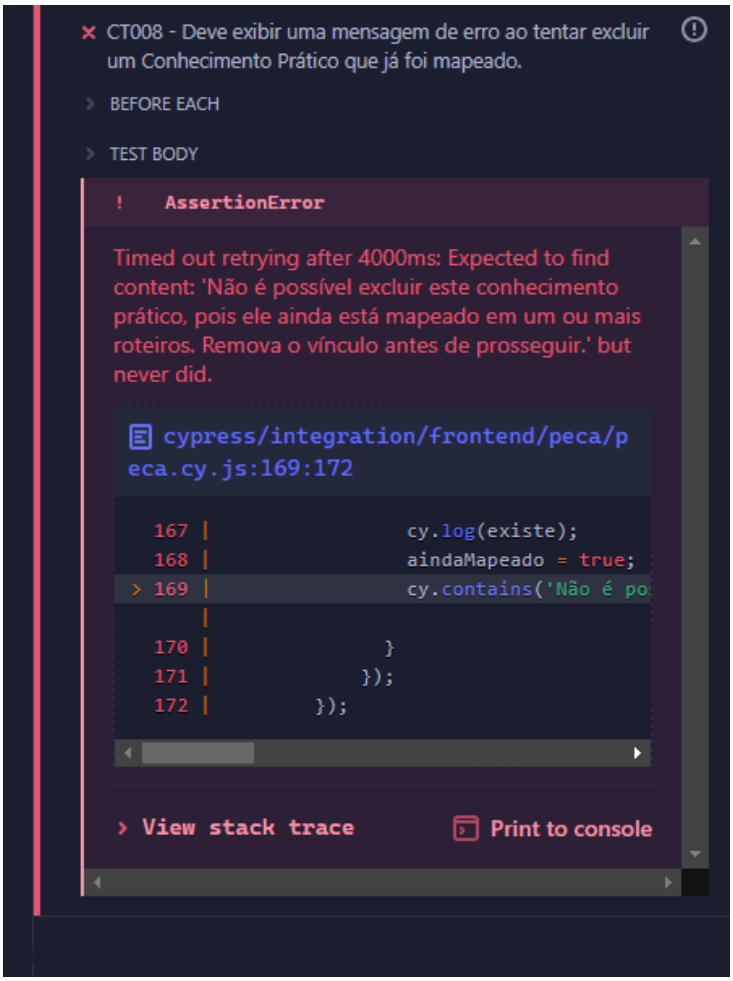
Fonte: O autor.

Figura 16 – Resultado dos casos de testes dos roteiros.

Fonte: O autor.

Na Figura 17, é possível observar a causa do erro no caso de teste CT-008, que tinha como objetivo validar se o sistema impede a exclusão de um conhecimento prático já mapeado no conteúdo de um roteiro.

Figura 17 – Detalhes do erro no caso de testes CT008.



Fonte: O autor.

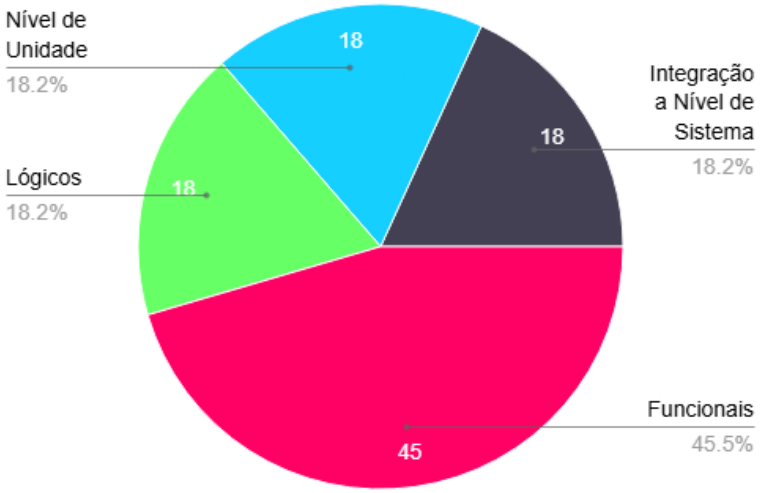
Neste caso, o erro ocorre porque o comportamento esperado era que o sistema alertasse o usuário informando que a exclusão não poderia ser realizada. No entanto, durante a execução do teste, não foi possível identificar a exibição dessa mensagem, indicando uma falha no mecanismo de validação do sistema.

Os resultados obtidos com a execução dos testes evidenciam inconsistências, entre o que era esperado e como o sistema realmente se comportou em relação aos critérios de validação estabelecidos. A ausência de sucesso em todos os itens de teste reforça a necessidade de ajustes no código do Anatome-AT para garantir que o sistema atenda aos comportamentos esperados.

A falta de sucesso nos itens de teste estão associadas a diferentes tipos de defeitos, dentre eles os defeitos funcionais, onde o sistema não está se comporta conforme o esperado. No gráfico da Figura 18 pode-se observar de forma mais detalhada uma distribuição dos ICs (Itens de Cobertura de Teste) pelos tipos de defeitos identificados. Esse gráfico mostra que 45,5% dos defeitos que foram identificados são de natureza funcional, o que significa que a maioria dos itens de teste não tiveram sucesso por não identificarem que o sistema se comportou como era o esperado. E distribuídos de forma proporcional, temos os defeitos lógicos (fluxo

ou tomadas de decisões erradas no código), os defeitos a nível de unidade (erros em partes isoladas do código, como os métodos de validação dos formulários) e, por fim, os defeitos de integração a nível de sistema (que são problemas ao associar diferentes partes do sistema).

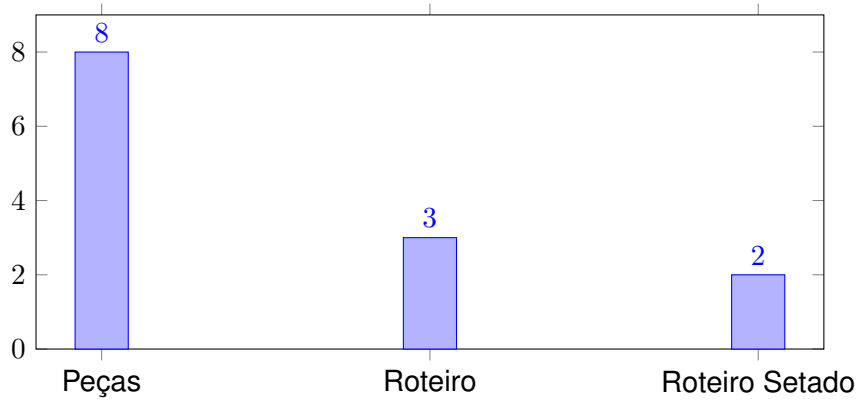
Figura 18 – Distribuição dos ICs por Tipo de Defeito



Fonte: O autor.

Já no gráfico da Figura 19, é possível observar a quantidade de defeitos identificados por módulo. O gráfico mostra que o módulo de Peças concentra a maior parte dos defeitos identificados, seguido pelo módulo de Roteiros e Roteiro Setado. Essa distribuição sugere que o módulo de peças pode necessitar de mais atenção na fase de implementação dos ajustes, pois é onde de concentram as maiores quantidades de defeitos identificados durantes os testes automatizados.

Figura 19 – Quantidade de Defeitos por Módulo



Fonte: O autor.

4.3 Recomendações de ajustes para a melhoria da robustez do Anatome-AT

Com base no resultados dos testes realizados e no comportamento esperado do sistema, foram identificadas ajustes que podem ser realizadas com o objetivo de solucionar os erros identificados e aumentar a robustez do sistema. A seguir, são apresentadas as recomendações para os ajustes a serem implementadas.

As recomendações foram elaboradas com base nos itens de cobertura de teste (IC), desenvolvidos para identificar erros no sistema.

- **IC001 - Digitar espaços em branco para o nome da peça e conhecimento prático.**
  - **Casos de Teste (CT):** CT001, CT002 e CT003
  - **Problema:** O sistema não está validando corretamente entradas com espaços em branco.
  - **Recomendações:**
    - Implementar uma validação no frontend e no backend para rejeitar entradas que consistam apenas em espaços em branco.
    - Adicionar uma mensagem de erro para o usuário informando que o campo não pode conter apenas espaços em branco.
- **IC002 - Digitar apenas um caractere para o nome e conhecimento prático.**
  - **Casos de Teste (CT):** CT004, CT005 e CT006
  - **Problema:** O sistema não está validando o comprimento mínimo dos campos.
  - **Recomendações:**
    - Adicionar validações para garantir que os campos *Nome* e *Conhecimento Prático* tenham um número mínimo de caracteres (ex.: 2 caracteres).

    - Exibir uma mensagem de erro informando o comprimento mínimo necessário.

- **IC003 - Excluir um conhecimento prático que já foi mapeado e verificar se ele foi removido do mapeamento.**
  - **Caso de Teste (CT):** CT007
  - **Problema:** A exclusão de um conhecimento prático mapeado não está sendo refletida corretamente nos mapeamentos associados ao conhecimento.
  - **Recomendações:**
    - Verificar a lógica de exclusão no backend para garantir que o conhecimento prático seja removido corretamente dos mapeamentos associados a ele.
- **IC004 - Clicar em excluir um conhecimento prático e verificar se é exibida uma mensagem impedindo a exclusão pelo fato de o conhecimento já ter sido mapeado.**
  - **Caso de Teste (CT):** CT008
  - **Problema:** O sistema não apresentou a mensagem esperada ao tentar excluir um conhecimento mapeado.
  - **Recomendações:**
    - No clique no botão de excluir um conhecimento prático adicionar um evento para verificar se o conhecimento prático está mapeado antes de permitir a exclusão.
    - Após validação, se o conhecimento prático estiver mapeado, exibir para o usuário uma mensagem de erro informando que o conhecimento prático não pode ser excluído por já estar mapeado.
- **IC005 - No mapeamento de uma peça digital, ao clicar em uma parte mapeada para referenciá-la a partir de uma localização relativa, e posteriormente cancelar a ação, o sistema deve garantir que o mapeamento da parte permaneça intacto.**
  - **Caso de Teste (CT):** CT009
  - **Problema:** O mapeamento da peça não está sendo preservado após o cancelamento da ação.
  - **Recomendações:**
    - Verificar a lógica de seleção da localização referenciada e do cancelamento para garantir que o estado original do mapeamento seja preservado.
- **IC006 - Selecionar um roteiro setado cadastrado para edição, incluir uma nova peça e verificar se é possível associar um conteúdo a nova peça.**
  - **Caso de Teste (CT):** CT010
  - **Problema:** O sistema não está permitindo a associação de conteúdo à uma nova peça na edição de um roteiro setado.
  - **Recomendações:**

    - Verificar a lógica de adição de peça e identificar o porque do campo de associar o conteúdo a peça permanece desabilitado.

- **IC007 - Começar o cadastro de um novo roteiro e clicar no botão de voltar e confirmar se o sistema exibe um alerta informando que as informações não salvas serão perdidas e se o usuário deseja mesmo voltar.**
  - **Caso de Teste (CT):** CT011
  - **Problema:** O sistema não exibe um alerta informando sobre a perda de dados não salvos ao tentar sair do cadastro de um novo roteiro.
  - **Recomendações:**
    - Implementar um modal de confirmação no frontend para alertar o usuário sobre a perda de dados não salvos.
    - Garantir que o modal seja exibido apenas quando houver alterações no roteiro.
- **IC008 - No cadastro de um roteiro validar se o sistema permite editar o conteúdo de uma peça dentro do cadastro.**
  - **Caso de Teste (CT):** CT012
  - **Problema:** O sistema não está permitindo a edição de uma peça durante o cadastro de um roteiro.
  - **Recomendações:**
    - Adicionar a possibilidade de editar uma peça dentro do cadastro do roteiro e abrir o modal do formulário de edição contendo as informações da peça.
- **IC009 - Dentro do cadastro de um roteiro, iniciar o cadastro de uma nova peça, ao salvar o sistema deve selecionar a peça automaticamente para aquele roteiro.**
  - **Caso de Teste (CT):** CT013
  - **Problema:** A nova peça cadastrada dentro de um roteiro, não está sendo selecionada automaticamente para aquele roteiro.
  - **Recomendações:**
    - Implementar uma lógica no frontend para selecionar automaticamente a nova peça após o salvamento.

5 Considerações Finais

Este trabalho teve como objetivo identificar erros no Anatome-AT, utilizando a técnica de Suposição de Erros para fornecer uma base que oriente futuras melhorias no sistema. Para isso foram conduzidas etapas metodológicas, incluindo a revisão de literatura, a elaboração de modelo de teste, a definição e execução dos testes e análise dos resultados obtidos.

A realização dos testes permitiu mapear falhas que poderiam comprometer a robustez e adequação funcional do sistema, evidenciando os pontos que precisam ser ajustados. A escolha do Cypress, viabilizou a automação dos testes, tornando o processo mais eficiente e garantindo maior precisão na detecção de erros. O resultado dos testes possibilitou uma análise dos erros que serviram para definir as recomendações de melhorias futuras.

Ao longo deste trabalho, constatou-se que a aplicação da técnica de Suposição de Erros, combinada com a execução de testes automatizados, mostrou-se eficaz na identificação de falhas e na compreensão do comportamento do sistema. Essa abordagem forneceu uma base real para a definição de recomendações que visam melhorar o sistema.

Por fim, conclui-se que a abordagem adotada neste trabalho apresenta um potencial para contribuir com a evolução do Anatome-AT, tornando-o mais confiável e robusto. A identificação e análise dos erros, por meio dos testes realizados, evidenciou pontos de melhoria, possibilitou a elaboração de recomendações para versões futuras. Essas recomendações, quando implementadas, têm o potencial de elevar a qualidade do sistema, garantindo maior eficiência e satisfação dos usuários.

5.1 Trabalhos futuros

Como trabalhos futuros, sugere-se a implementação das melhorias indicadas com base nos resultados dos testes e a realização de novos ciclos de validação para verificar os impactos das correções. Além disso, a expansão do conjunto de testes, incorporando outras técnicas de teste descritas na norma ISO/IEC/IEEE 29119, pode contribuir para uma análise ainda mais abrangente da estabilidade e usabilidade do sistema.

Referências

BRITTON, J. *What Is ISO 25010?* 2021. Disponível em: <https://www.perforce.com/blog/qac/what-is-iso-25010>. Acesso em: 27 jan. 2025. Citado na página 16.

CYPRESS. *Cypress Documentation*. 2024. Disponível em: <https://docs.cypress.io/app/get-started/why-cypress>. Acesso em: 25 jan. 2025. Citado na página 21.

CYPRESS. *Testing Frameworks for Javascript | Write, Run, Debug | Cypress*. 2025. Disponível em: <https://www.cypress.io/>. Acesso em: 25 jan. 2025. Citado nas páginas 35 e 39.

FERREIRA, M. V. R. *Anatome: ensino e aprendizagem de anatomia modelados para todos*. 2019. 197 p. Tese (Doutorado) — Universidade Federal do Paraná, 2019. Citado nas páginas 14, 15, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32 e 33.

GABRIEL, G. M. *Realização de testes estruturais para a identificação de defeitos e auxílio na qualidade: um estudo de caso*. 2024. TCC — Universidade Federal Rural do Semi-árido, 2024. Citado na página 34.

GOOGLE. *Navegador web Google Chrome*. 2025. Disponível em: <https://www.google.com/intl/pt-BR/chrome/>. Acesso em: 25 jan. 2025. Citado na página 39.

HIRAMA, K. *Engenharia de software : qualidade e produtividade com tecnologia*. Rio de Janeiro: Elsevier, 2011. Citado nas páginas 14 e 18.

ISO/IEC/IEEE. *ISO/IEC/IEEE 29119-4:2021: Software and systems engineering — Software testing — Part 4: Test techniques*. Switzerland: ISO/IEC/IEEE, 2021. Citado nas páginas 14, 15, 19, 20, 35, 37 e 42.

ISO/IEC/IEEE. *ISO/IEC/IEEE 29119-1:2022: Software and systems engineering - Software testing - Part 1: General concepts*. Switzerland: ISO/IEC/IEEE, 2022. Citado nas páginas 15, 16, 19, 20 e 35.

KHAN, F. *Error Guessing*. 2022. Disponível em: <https://www.toolsqa.com/software-testing/error-guessing-technique-software-testing/>. Acesso em: 25 jan. 2025. Citado na página 14.

MEGER, B. de S. *Testes automatizados de front-end com utilização de BDD e Cypress*. 2024. TCC — Universidade Presbiteriana Mackenzie, 2024. Citado nas páginas 33 e 34.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The Art of Software Testing*. 3rd. ed. [S.l.]: John Wiley & Sons, Inc., 2012. Citado nas páginas 14, 17, 18 e 20.

NAIK, K.; TRIPATHY, P. *Software Testing and Quality Assurance: Theory and Practice*. 1. ed. [S.l.]: Wiley Publishing, 2008. Citado nas páginas 18 e 21.

ONPENJS. *Node.js — Executar a JavaScript em Toda Parte*. 2025. Disponível em: <https://nodejs.org/pt>. Acesso em: 25 jan. 2025. Citado na página 39.

PARMAR, D. *Exploratory testing*. 2024. Disponível em: <https://www.atlassian.com/continuous-delivery/software-testing/exploratory-testing>. Acesso em: 27 jan. 2025. Citado na página 18.

PRESSMAN, R. S. *Engenharia de software : uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016. Citado nas páginas 16 e 17.

RAMOS, R. R. B. *Proposta de testes para serious games voltados ao condicionamento físico e reabilitação com aplicação no jogo GiroJampa*. 2024. TCC — Universidade Federal da Paraíba, 2024. Citado na página 34.

SANTANA, A. C. R. *Apoio ao ensino e aprendizagem remotos de anatomia: extensão do projeto Anatome*. 2022. TCC — Centro Federal de Educação Tecnológica de Minas Gerais - Campus Timóteo, 2022. Citado nas páginas 15, 23 e 78.

SANTOS, E. M. dos. *Cypress: Passo a passo para começar a usar*. 2024. Disponível em: <https://softdesign.com.br/blog/cypress_passo_a_passo_para_comecar_a_usar/#h-por-que-usar-o-cypress>. Acesso em: 26 jan. 2025. Citado nas páginas 21 e 39.

SOEIRO, K. *Conheça as diferenças entre Testes Exploratórios e Ad-hoc*. 2023. Disponível em: <https://testingcompany.com.br/blog/conheca-as-diferencas-entre-testes-exploratorios-e-ad-hoc>. Acesso em: 27 jan. 2025. Citado na página 18.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. Citado nas páginas 14, 17 e 18.

Apêndices

APÊNDICE A – String de busca para trabalhos correlatos

A *string* de busca elaborada com termos relevantes para o tema deste trabalho:

- (teste) AND
- (software OR sistema) AND
- (ferramenta de autoria) AND
- (refatoração)

Para bases internacionais esses termos foram adaptados para o inglês:

- (*test*) AND
- (*software* OR *system*) AND
- (*authoring tool*) AND
- (*refactoring*)

APÊNDICE B – Códigos dos casos de testes

Este apêndice apresenta os códigos utilizados para a execução dos testes. Esses códigos incluem a especificação dos cenários de teste, os comandos executados e as validações aplicadas, garantindo a reprodução dos testes.

B.1 Casos de Testes Peças

O código mostrado em B.1, mostra a estrutura base do script de testes para os casos relacionados às peças. Todos os códigos dos casos de testes que serão apresentados em seguida, estão agrupados dentro do `describe`.

Código Fonte B.1 – Estrutura base dos testes relacionados a Peça.

```
describe('Casos de Teste das Peças', () => {
  let testes;

  before(() => {
    cy.fixture('dadosTestesCadastroPeca').then((data) => {
      testes = data;
    });
  });

  beforeEach(() => {
    cy.visit('/peca/cadastrar');
  });

});
```

Fonte: O autor.

Código Fonte B.2 – Caso de Teste CT001.

```
it ("CT001 - Não cadastrar peça com o campo 'Nome Peça'
   preenchido somente com espaços em branco", () => {
    cy.validarCamposPeca(testes.CT001);
});
```

Fonte: O autor.

Código Fonte B.3 – Caso de Teste CT002.

```
it ("CT002 - Não cadastrar peça com o campo 'Conhecimento Prático
    ' preenchido somente com espaços em branco", () => {
     cy.validarCamposPeca(testes.CT002);
});
```

Fonte: O autor.

Código Fonte B.4 – Caso de Teste CT003.

```
it ("CT003 - Não cadastrar peça com os campos 'Nome Peça' e '
    Conhecimento Prático' preenchidos somente com espaços em branco
    ", () => {
     cy.validarCamposPeca(testes.CT003);
});
```

Fonte: O autor.

Código Fonte B.5 – Caso de Teste CT004.

```
it ("CT004 - Não cadastrar peça com o campo 'Nome Peça' contendo
    apenas um caractere", () => {
     cy.validarCamposPeca(testes.CT004);
});
```

Fonte: O autor.

Código Fonte B.6 – Caso de Teste CT005.

```
it ("CT005 - Não cadastrar peça com o campo 'Conhecimento Prático
    ' contendo apenas um caractere", () => {
     cy.validarCamposPeca(testes.CT005);
});
```

Fonte: O autor.

Código Fonte B.7 – Caso de Teste CT006.

```
it ("CT006 - Não cadastrar peça com os campos 'Nome Peça' e '
    Conhecimento Prático' contendo apenas um caractere", () => {
     cy.validarCamposPeca(testes.CT006);
});
```

Fonte: O autor.

Código Fonte B.8 – Caso de Teste CT007.

```
it ("CT007 - Excluir um conhecimento Prático que já foi mapeado
   deve o remover do mapeamento.", () => {
    const nomePecaRoteiro = "Crânio Face";
    const pecaAnatomica = "Crânio - Face"
    const nomeConhecimentoPratico = "Maxilia";

    cy.visit('/pecas');

    cy.get('[data-cy="tabela-pecas"] tbody tr')  // Seleciona
       todas as linhas da tabela
    .each(($row) => {
      if ($row.text().includes(pecaAnatomica)) {  // Verifica se
         o nome da peça está na linha
        cy.wrap($row)
          .find('button')  // Encontra o botão de editar dentro
             da linha
          .first()  // Seleciona o primeiro botão
          .click();  // Clica no botão de editar

        return false;  // Interrompe o loop `each`
      }
    });

    cy.get('[data-cy="linha-conhecimento-teorico"]')
      .contains(nomeConhecimentoPratico)  // Verificar o nome do
         conhecimento
      .parents('[data-cy="linha-conhecimento-teorico"]') //
         Localizar o item inteiro
      .within(() => {
          // Clicar no botão de excluir (ícone de lixeira)
          cy.get('[data-cy="excluir-ct"]').click();
      });

    // CONFIRMAR A EXCLUSÃO NO MODAL
    cy.getElement('modal-excluir-conhecimento-teorico').should('
       be.visible');
    cy.getElement('confirmar-exclusao-ct').click() // Clica no
       botão de excluir o conhecimento teórico
    cy.getElement('modal-excluir-conhecimento-teorico').should('
       not.be.visible'); // Verifica se o modal foi fechado
```

```
    // EXCLUIR O CONHECIMENTO PRÁTICO
    cy.getElement(`icone-excluir-cp-${nomeConhecimentoPratico}`).
       click(); // clica para excluir o conhecimento pratico
    cy.getElement(`cp-parte-${nomeConhecimentoPratico}`).should('
       not.exist'); // verifica se foi excluído

    // SALVA A PEÇA
    cy.getElement('salvar-peca').click();
    cy.contains(`O conteúdo da peça ${pecaAnatomica} foi salvo
       com sucesso!`).should('be.visible');

    // VERIFICAR SE O CONHECIMENTO PRÁTICO TAMBÉM FOI EXCLUÍDO
       DOS MAPEAMENTOS
    cy.request({
      method: 'GET',
      url: 'https://anatome-api.onrender.com/anatomp',
      failOnStatusCode: false
    })
      .then((response) => {
        const listaRoteirosSetadosJson = response.body.data;

        // Valida se o Conhecimento Prático excluído ainda está
           mapeado em algum
        // roteiro setado.
        listaRoteirosSetadosJson.forEach((roteiro) => {
          cy.verificaConhecimentoPraticoExiste(roteiro,
             nomeConhecimentoPratico, nomePecaRoteiro)
            .then((aindaMapeada) => {
              expect(aindaMapeada).to.be.false;
            });
        });
      });
});
```

Fonte: O autor.

Código Fonte B.9 – Caso de Teste CT008.

```
it ("CT008 - Deve exibir uma mensagem de erro ao tentar excluir
   um Conhecimento Prático que já foi mapeado.", () => {
    const nomePecaRoteiro = "Crânio Face";
    const pecaAnatomica = "Crânio - Face"
```

```
const nomeConhecimentoPratico = "Zigomático";

cy.visit('/pecas');

cy.get('[data-cy="tabela-pecas"] tbody tr')  // Seleciona
   todas as linhas da tabela
.each(($row) => {
  if ($row.text().includes(pecaAnatomica)) {  // Verifica se
     o nome da peça está na linha
    cy.wrap($row)
      .find('button')  // Encontra o botão de editar dentro
         da linha
      .first()  // Seleciona o primeiro botão
      .click();  // Clica no botão de editar

    return false;  // Interrompe o loop `each`
  }
});

cy.get('[data-cy="linha-conhecimento-teorico"]')
  .contains(nomeConhecimentoPratico)  // Verificar o nome do
     conhecimento
  .parents('[data-cy="linha-conhecimento-teorico"]') //
     Localizar o item inteiro
  .within(() => {
      // Clicar no botão de excluir (ícone de lixeira)
      cy.get('[data-cy="excluir-ct"]').click();
  });

// CONFIRMAR A EXCLUSÃO NO MODAL
cy.getElement('modal-excluir-conhecimento-teorico').should('
   be.visible');
cy.getElement('confirmar-exclusao-ct').click() // Clica no
   botão de excluir o conhecimento teórico
cy.getElement('modal-excluir-conhecimento-teorico').should('
   not.be.visible'); // Verifica se o modal foi fechado

// EXCLUIR O CONHECIMENTO PRÁTICO
// CLICA PARA EXCLUIR O CONHECIMENTO PRÁTICO
cy.getElement(`icone-excluir-cp-${nomeConhecimentoPratico}`).
   click().then(() => {
  // Faz a requisição para verificar se o conhecimento prá
     tico está mapeado
```

```
      cy.request({
        method: 'GET',
        url: 'https://anatome-api.onrender.com/anatomp',
        failOnStatusCode: false
      }).then((response) => {
        const listaRoteirosSetadosJson = response.body.data;
        let aindaMapeado = false;

        // Verifica se o conhecimento prático ainda está em algum
            roteiro
        listaRoteirosSetadosJson.forEach((roteiro) => {
          cy.verificaConhecimentoPraticoExiste(roteiro,
             nomeConhecimentoPratico, nomePecaRoteiro)
            .then((existe) => {
              if (existe) {
                cy.log(existe);
                aindaMapeado = true;
                cy.contains('Não é possível excluir este
                   conhecimento prático, pois ele ainda está
                   mapeado em um ou mais roteiros. Remova o ví
                   nculo antes de prosseguir.').should('be.visible
                   ');
              }
            });
        });
      });
    });
});
```

Fonte: O autor.

B.1.1 Arquivo JSON com dados para validação da peça

O JSON mostrado no código B.10, é as informações de entradas utilizadas para validar os campos do formulário de cadastro de uma peça.

A estrutura completa do JSON que armazena as informações da peça pode ser visualizado no código B.11.

Código Fonte B.10 – Arquivo `dadosTestesCadastroPeca.json`.

```
{
  "CT001": {
    "descricao": "Não cadastrar peça com o campo 'Nome Peça'
       preenchido somente com espaços em branco",
```

```
    "entradas": {
      "nomePeca": "  ",
      "partesAnatomicas": ["Ventrículo", "Átrio"]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
    "messageContent": "Verifique os erros de validação!"
  },

  "CT002": {
    "descricao": "Não cadastrar peça com o campo 'Conhecimento Pr
       ático' preenchido somente com espaços em branco",
    "entradas": {
      "nomePeca": "Coração",
      "partesAnatomicas": ["  ", "  "]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
    "messageContent": "Verifique os erros de validação!"
  },

  "CT003": {
    "descricao": "Não cadastrar peça com os campos 'Nome Peça' e
       'Conhecimento Prático' preenchidos somente com espaços em
       branco",
    "entradas": {
      "nomePeca": "  ",
      "partesAnatomicas": ["  ", "  "]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
    "messageContent": "Verifique os erros de validação!"
  },

  "CT004": {
    "descricao": "Não cadastrar peça com o campo 'Nome Peça'
       contendo apenas um caractere",
    "entradas": {
      "nomePeca": "C",
      "partesAnatomicas": ["Frontal", "Nasal"]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
```

```
    "messageContent": "Verifique os erros de validação!"
  },

  "CT005": {
    "descricao": "Não cadastrar peça com o campo 'Conhecimento Pr
       ático' contendo apenas um caractere",
    "entradas": {
      "nomePeca": "Crânio - Face",
      "partesAnatomicas": ["F", "N"]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
    "messageContent": "Verifique os erros de validação!"
  },

  "CT006": {
    "descricao": "Não cadastrar peça com os campos 'Nome Peça' e
       'Conhecimento Prático' contendo apenas um caractere",
    "entradas": {
      "nomePeca": "C",
      "partesAnatomicas": ["F", "N"]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
    "messageContent": "Verifique os erros de validação!"
  }
}
```

Fonte: O autor.

Código Fonte B.11 – Estrutura completa do JSON que armazena informações de uma peça.

```
"casoDeTeste": {
    "descricao": "Descrição do Caso de Teste",
    "entradas": {
      "nomePeca": "",
      "regiao": null,
      "sistema": null,
      "generalidades": "",
      "somenteConteudoPratico": false,
      "partesAnatomicas": ["Cabeça"],
      "conhecimentoTeorico": [
```

```
        { "parte": "Cabeça", "conhecimento": "Parte superior do
           corpo."},
        { "parte": "Cabeça", "conhecimento": "Parte superior do
           corpo."}
      ]
    },
    "resultadoEsperado": "ERROR_MESSAGE",
    "expectedURL": null,
    "messageContent": "Verifique os erros de validação!"
  }
```

Fonte: O autor.

B.2 Casos de Testes Roteiro e Roteiro Setado

O código mostrado em B.12, mostra a estrutura base do script de testes para os casos relacionados aos roteiros. Todos os códigos dos casos de testes que serão apresentados em seguida, estão agrupados dentro do `describe`.

Código Fonte B.12 – Estrutura base dos testes relacionados aos Roteiros.

```
describe('Casos de Teste do Roteiro e Roteiro Setado', () => {
  beforeEach(() => {
    cy.visit('/');
  });
});
```

Fonte: O autor.

Código Fonte B.13 – Caso de Teste CT009.

```
it ('CT009 - O ícone de mapeamento deve permanecer visível após
   clicar em cancelar no modal de Localização Relativa', () => {
    cy.abrirUltimoRoteiroSetado();

    cy.get('[data-cy^="cd-peca-digital"]') // Seleciona todos os
       cards
    .first() // Pega o primeiro card
    .within(() => { // Garante que a busca ocorre dentro do card
        cy.get('[data-cy^="btn-setar-localizacao"]').click(); //
           Clica no botão de setar localização
    });
```

```
    const nomePeca = 'Frontal';

    // Clica no checkbox da parte anatomica selecionada para
       abrir o modal de Localização Relativa
    cy.getElement(`mp-checkbox-referenciado-${nomePeca}`).click()
       ;

    // Clica no botão de cancelar no modal de Localização
       Relativa
    cy.getElement('cancelar-referenciamento').click();

    // Verifica se o mapeamento da parte anatomica foi mantido
    cy.getElement(`mp-tag-red-referencia-${nomePeca}`).should('be
       .visible');
});
```

Fonte: O autor.

Código Fonte B.14 – Caso de Teste CT010.

```
it ('CT010 - Ao editar um roteiro setado deve permitir a adição
   de uma nova peça e associar um novo conteúdo a ela.', () => {
    cy.abrirUltimoRoteiroSetado();

    // Clica no botão de adicionar peça
    cy.getElement('btn-adicionar-peca').click();

    // Recupera todos os elementos que comecem com select-
       conteudo-peca
    // e verifica se ele está habilitado
    cy.get('[data-cy^="select-conteudo-peca"]').last().should('be.
       enabled');
});
```

Fonte: O autor.

Código Fonte B.15 – Caso de Teste CT011.

```
it ('CT011 - Deve exibir uma mensagem informando que o usuário
   perderá os dados não salvos quando ele clicar no botão de
   voltar no cadastro de um roteiro.', () => {
    cy.visit('/roteiro/cadastrar');
```

```
    const nomeRoteiro = 'Anatomia da Cavidade Oral - Estudo da
       Mandíbula';
    const curso = 'Odontologia';
    const disciplina = 'Anatomia Odontológica I';
    const pecaConhecimentoPratico = 'Mandíbula';

    // Preenche as informações gerais do roteiro
    cy.getElement('nome-roteiro').type(nomeRoteiro);
    cy.getElement('curso').type(curso);
    cy.getElement('disciplina').type(disciplina);

    // Localiza uma peça na seção de conhecimento prático.
    cy.get('[data-cy="tree-list-pecas"]') // Acessa a lista
      .contains('span.ant-tree-title', pecaConhecimentoPratico)
        // Localiza a peça pelo nome
      .closest('li') // Sobe para o item da árvore correspondente
      .find('.ant-tree-checkbox') // Encontra o checkbox
      .first()
      .click(); // Marca o checkbox

    // Seleciona todos os conhecimentos teóricos associados a peç
       a.
    cy.get('[data-cy="tabela-conhecimentos-teoricos"] thead .ant-
       checkbox-input')
      .not('[disabled]')
      .check();

    // Verifica se foi algum conteúdo foi adicionado ao roteiro,
       depois clica
    // no botão de voltar e verifica se a mensagem informando que
        os dados não
    // salvos serão perdidos é exibida.
    cy.get('input').then(($inputs) => {
      let algumInputPreenchido = false;
      $inputs.each((input) => {
        if (input.value !== '') {
          algumInputPreenchido = true;
          return false;
        }
      });

      cy.getElement('btn-voltar').click().then(() => {
        if (algumInputPreenchido)
```

```
            cy.contains('Você tem alterações não salvas no seu
               roteiro. Se você sair desta página, elas serão
               perdidas. Deseja continuar?');
        });
    });
});
```

Fonte: O autor.

Código Fonte B.16 – Caso de Teste CT012.

```
it ('CT012 - Deve permitir editar uma peça no cadastro de um
   roteiro', () => {
    const nomeRoteiro = 'Estudo da anatomia do Crânio';

    cy.aguardarCarregamentoDosRoteiros();

    cy.get('[data-cy="tabela-roteiros"] tbody tr')  // Seleciona
       todas as linhas da tabela
      .each(($row) => {
        if ($row.text().includes(nomeRoteiro)) {  // Verifica se
           o nome do roteiro está na linha
          cy.wrap($row)
            .find('button')  // Encontra o botão de editar dentro
               da linha
            .first()  // Seleciona o primeiro botão (editar)
            .click();  // Clica no botão de editar

          return false;  // Interrompe o loop `each`
        }
      });

    cy.get('[data-cy="tree-list-pecas"] li').each(($el) => {
      cy.wrap($el).within(() => {
        cy.get('[aria-label="icon: edit"]').should('exist');
      });
    });
});
```

Fonte: O autor.

Código Fonte B.17 – Caso de Teste CT013.

```
it ('CT013 - Ao cadastrar uma peça no processo de cadastro de um
   roteiro, a peça deve ser selecionada automaticamente para
   aquele roteiro.', () => {
    cy.visit('/roteiro/cadastrar');

    const conteudoPeca = {
      entradas: {
        nomePeca: "Coração",
        regiao: "Tronco",
        sistema: "Sistema circulatório",
        generalidades: "",
        somenteConteudoPratico: false,
        partesAnatomicas: ["Átrios", "Ventrículos"],
        conhecimentoTeorico: [
          { parte: "Átrios", conhecimento: "São as duas câmaras
             superiores do coração, que recebem sangue do sistema
             circulatório e o enviam para os ventrículos." },
          { parte: "Ventrículos", conhecimento: "São as duas câ
             maras inferiores, responsáveis por bombear o sangue
             para os tecidos do corpo." }
        ]
      }
    }

    cy.getElement('adicionar-peca-roteiro').click(); // Abre o
       modal de adicionar peça dentro do roteiro

    cy.preencherFormularioPeca(conteudoPeca);

    cy.getElement('botao-modal-salvar-conteudo-peca').click();

    cy.contains(`O conteúdo da peça ${conteudoPeca.entradas.
       nomePeca} foi salvo com sucesso!`);

    cy.get('[data-cy="tree-list-pecas"]') // Acessa a lista de pe
       ças
    .contains('span.ant-tree-title', conteudoPeca.entradas.
       nomePeca) // Localiza a peça pelo nome
    .closest('li') // Sobe para o item da árvore correspondente
    .find('.ant-tree-checkbox') // Encontra o checkbox
    .first()
```

```
    .should('have.class', 'ant-tree-checkbox-checked'); //
       Verifica se está selecionado
});
```

Fonte: O autor.

B.3 Comandos Customizados

O código a seguir apresenta os comando personalizados que foram adicionados no arquivo `commands.js`, com o objetivo de facilitar o reuso de trechos de códigos nos casos de teste.

Código Fonte B.18 – Comandos customizados adiconados ao Cypress.

```
const apiURL = "https://anatome-api.onrender.com";

// ==================== COMMON COMMANDS ====================

Cypress.Commands.add('getElement', (selector, ...args) => {
  return cy.get(`[data-cy="${selector}"]`, ...args);
})

// ==================== PEÇA COMMANDS ====================

Cypress.Commands.add('validarCamposPeca', (caso) => {
  if (caso.entradas.nomePeca)
    cy.getElement('nome-peca').clear().type(caso.entradas.
       nomePeca);

  if (caso.entradas.regiao)
    cy.getElement('regiao').type(caso.entradas.regiao);

  if (caso.entradas.sistema)
    cy.getElement('sistema').type(caso.entradas.sistema);

  if (caso.entradas.generalidades)
    cy.getElement('generalidades').type(caso.entradas.
       generalidades);

  if (caso.entradas.somenteConteudoPratico)
    cy.getElement('somenteConteudoPratico').click();

  if (caso.entradas.partesAnatomicas) {
    caso.entradas.partesAnatomicas.forEach(parte => {
```

```
      cy.getElement('partes').type(`${parte}{enter}`);
    });
  }

  if (!caso.entradas.somenteConteudoPratico && caso.entradas.
     conhecimentoTeorico) {
    caso.entradas.conhecimentoTeorico.forEach((ct, index) => {
      cy.getElement(`conhecimentoTeorico_${index}`).type(ct.parte
         );
      cy.getElement(`conhecimentoTeoricoSingular_${index}`).type(
         ct.conhecimento);

      if (index === caso.entradas.conhecimentoTeorico.length - 1)
          return;

      cy.getElement('btn-adicionarCT').click();
    });
  }


  cy.getElement('salvar-peca').click();

  //cy.url().should('eq', `${Cypress.config("baseUrl")}/peca/
     cadastrar`)

  if (caso.resultadoEsperado === 'ERROR_MESSAGE')
    cy.contains(caso.messageContent).should('be.visible');

  if (caso.resultadoEsperado === 'SUCCESS_MESSAGE')
    cy.contains(caso.messageContent.replace("*", caso.entradas.
       nomePeca)).should('be.visible');

  if (caso.resultadoEsperado === 'URL')
    cy.url().should('eq', `${Cypress.config("baseUrl")}${caso.
       expectedURL}`)

  cy.reload();
});

Cypress.Commands.add('aguardarCarregamentoPecas', () => {
  cy.intercept('GET', `${apiURL}/peca`).as('getPecas');
  cy.wait('@getPecas');
});
```

```
Cypress.Commands.add('preencherFormularioPeca', (peca) => {
  if (peca.entradas.nomePeca)
    cy.getElement('nome-peca').clear().type(peca.entradas.
       nomePeca);

  if (peca.entradas.regiao)
    cy.getElement('regiao').type(peca.entradas.regiao);

  if (peca.entradas.sistema)
    cy.getElement('sistema').type(peca.entradas.sistema);

  if (peca.entradas.generalidades)
    cy.getElement('generalidades').type(peca.entradas.
       generalidades);

  if (peca.entradas.somenteConteudoPratico)
    cy.getElement('somenteConteudoPratico').click();

  if (peca.entradas.partesAnatomicas) {
    peca.entradas.partesAnatomicas.forEach(parte => {
      cy.getElement('partes').type(`${parte}{enter}`);
    });
  }

  if (!peca.entradas.somenteConteudoPratico && peca.entradas.
     conhecimentoTeorico) {
    peca.entradas.conhecimentoTeorico.forEach((ct, index) => {
      cy.getElement(`conhecimentoTeorico_${index}`).type(ct.parte
         );
      cy.getElement(`conhecimentoTeoricoSingular_${index}`).type(
         ct.conhecimento);

      if (index === peca.entradas.conhecimentoTeorico.length - 1)
          return;

      cy.getElement('btn-adicionarCT').click();
    });
  }
});

// ================== END PEÇA COMMANDS ==================
```

```
// ================= ROTEIROs COMMANDS ====================

Cypress.Commands.add('abrirUltimoRoteiroSetado', () => {
  cy.intercept('GET', `${apiURL}/roteiro`).as('getRoteiros');
  cy.wait('@getRoteiros');

  cy.getElement('tabela-roteiros-setados') // Seleciona a tabela
  .find('tbody tr') // Encontra todas as linhas da tabela
  .last() // Pega a última linha
  .find('td:last-child button') // Seleciona os botões na última
     célula da linha
  .first() // Pega o primeiro botão (editar)
  .click();

  cy.wait(3000);
});

Cypress.Commands.add('verificaConhecimentoPraticoExiste', (json,
   nomeParte, nomePeca) => {
  // Verifica se alguma peça física contém o nome da peça
  const pecaEncontrada = json.pecasFisicas.find(peca => peca.nome
      === nomePeca);

  if (!pecaEncontrada) {
      return false;
  }

  // Verifica se a peça encontrada tem mídias
  if (!pecaEncontrada.midias || pecaEncontrada.midias.length ===
     0) {
      return false;
  }

  // Itera sobre as mídias da peça
  for (const midia of pecaEncontrada.midias) {
      // Verifica se a mídia tem pontos
      if (midia.pontos && midia.pontos.length > 0) {
          // Itera sobre os pontos da mídia
          for (const ponto of midia.pontos) {
              // Verifica se o ponto tem uma parte com o nome
                 especificado
              if (ponto.parte && ponto.parte.nome === nomeParte)
                 {
```

```
                    return true;
                }
            }
        }
    }
    return false;
});

Cypress.Commands.add('aguardarCarregamentoDosRoteiros', () => {
    cy.intercept('GET', `${apiURL}/roteiro`).as('getRoteiros');
    cy.wait('@getRoteiros');
});

// ================ END ROTEIROS COMMANDS ===================
```

Fonte: O autor.

APÊNDICE C – Cypress

Este apêndice apresenta informações sobre a instalação e configuração do Cypress para execução dos testes desenolvidos neste trabalho.

C.0.1 Instalação e configuração do Cypress

Antes de iniciar a implementação dos testes, foi necessário clonar as soluções do site e api do Anatome-AT. As soluções foram clonadas a partir do repositório do projeto Anatome (GitHub Anatome), utilizando a branch *dev-digital-peaces* disponibilizada por Santana (2022).

Com a solução clonada, foi criada um pasta com o nome de *testes* que contém os conteúdos de testes e configurações do Cypress.

Após a clonagem do repositório e dentro do diretório testes, o Cypress foi instalado utilizando o gerenciador de pacotes npm através do seguinte comando:

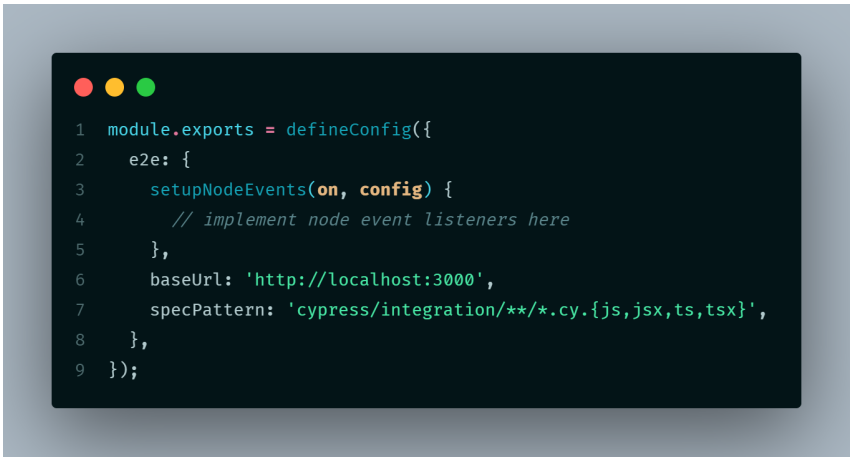
```
npm install cypress
```

Após a instalação, o Cypress foi inicializado com o comando:

```
npx cypress open
```

Esse comando criou a estrutura básica de diretórios e arquivos de configuração necessários para a execução dos testes. O arquivo cypress.config.js foi ajustado (Figura 20) para definir os parâmetros específicos do projeto, como a URL base (*baseUrl*) da aplicação e o diretório contendo os arquivos para teste (*specPattern*).

Figura 20 – Código de configuração do arquivo `cypress.config.js`.

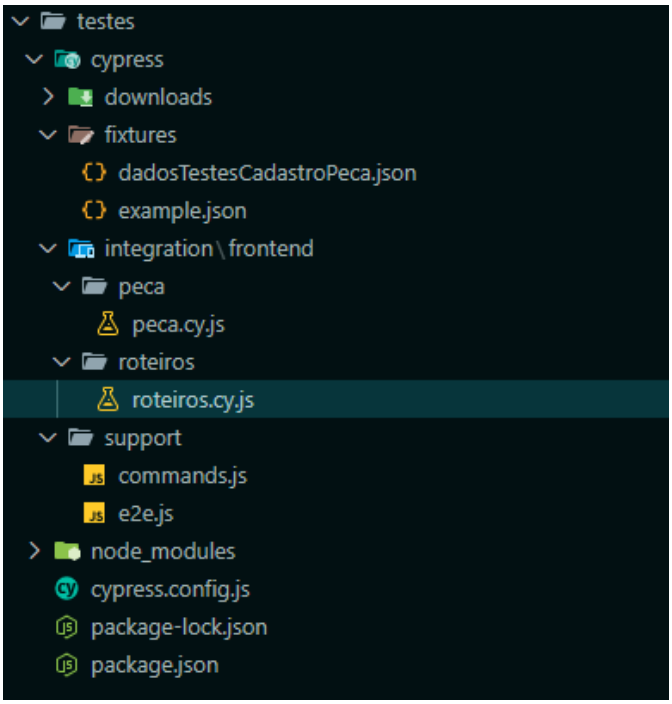


Fonte: O autor.

A Figura 21 apesenta a estrutura de pastas que foi criada para os testes, com o intuito de organizar os testes implementados. O diretório principal `testes/cypress` agrupa os subdiretórios responsáveis por estrutura os testes.

- `fixtures`: Contém arquivos JSON com dados reutilizáveis para os testes, como `dadosTestesCadasti` que armazena informações utilizadas na validação do cadastro de peças.
- `integration/frontend`: Organiza os testes por domínio.
  - `peca.cy.js`: Contém os testes relacionados às peças anatômicas.
  - `roteiros.cy.js`: Contém os testes referentes aos roteiros e roteiros setados.
- `support`: Inclui os arquivos auxiliares como `commands.js`, onde são definidos comandos personalizados do Cypress, e `e2e.js`, que contém configurações globais dos testes end-to-end.

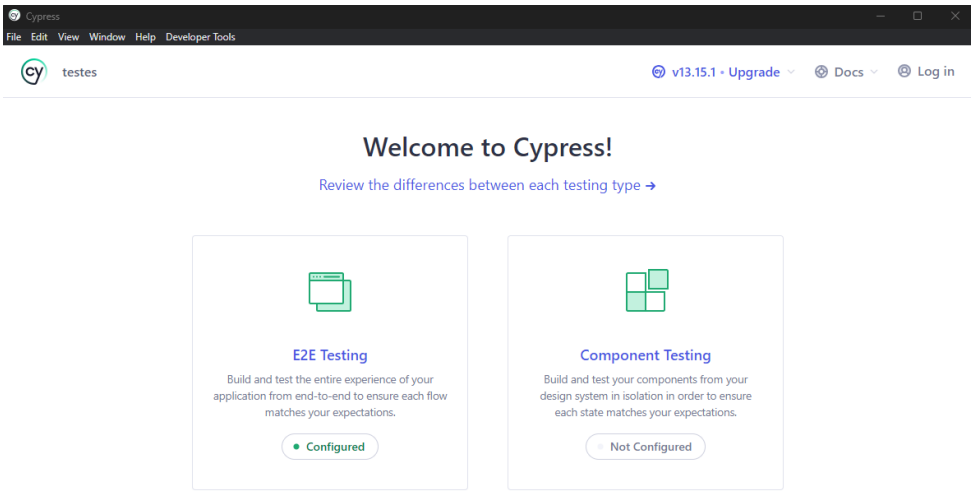
Figura 21 – Estrutura de Pastas do Cypress



Fonte: O autor.

Após instalação e configuração do Cypress, foi possível acessar a sua interface gráfica (Figura 22), que permite a execução e o gerenciamentos dos testes automatizados. A tela inicial apresenta as opções de testes: E2E (End-to-End) Testing ou Component Testing. Para os testes realizados o tipo de teste E2E foi o configurado para conseguir validar fluxos completos da aplicação.

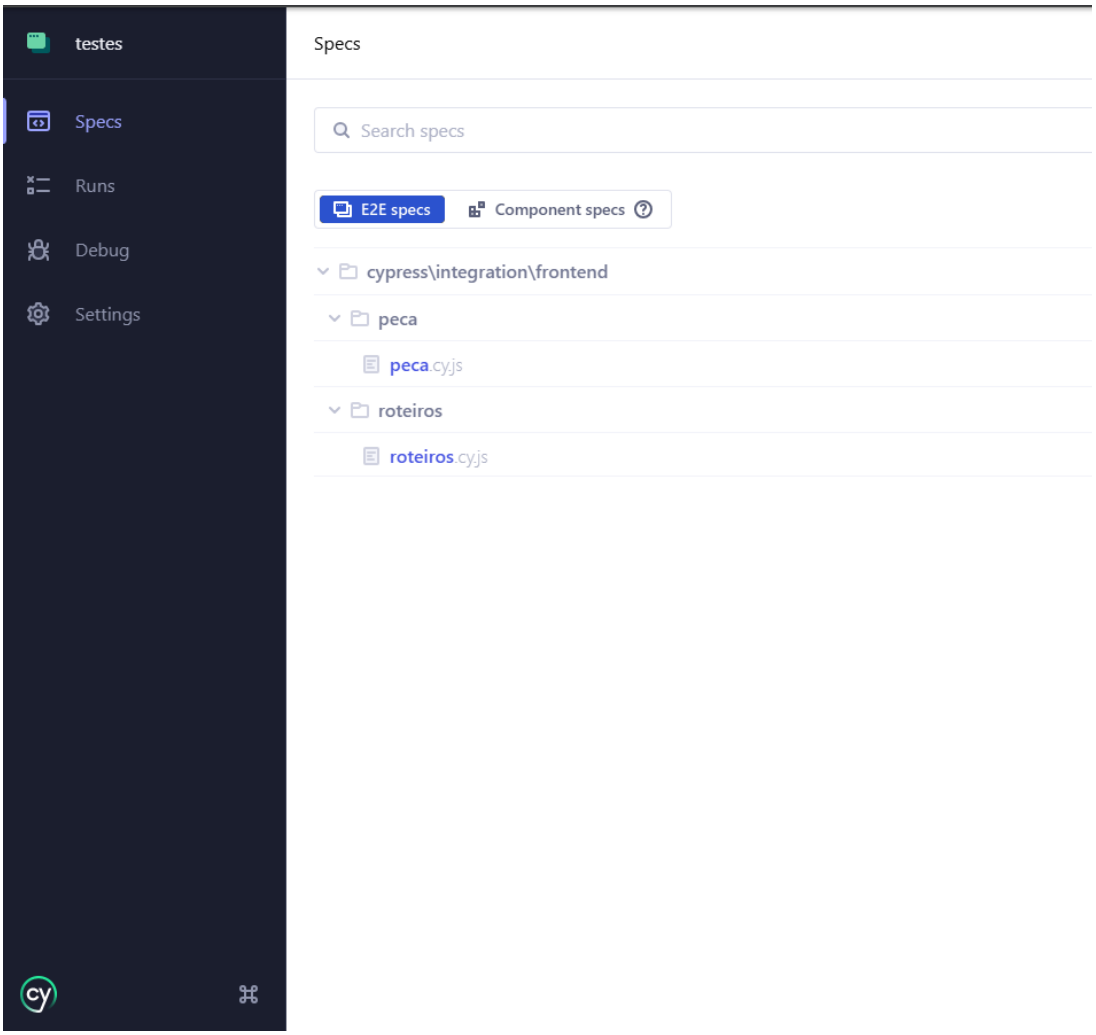
Figura 22 – Tela inicial do Cypress.



Fonte: O autor.

Na tela de especificações (Specs) (Figura 23) do Cypress, é possível visualizar todos os arquivos de testes implementados no projeto. Além da visualização, a tela permite a execução dos testes diretamente pela interface gráfica. Dessa forma, é possível rodar cada arquivo de teste individualmente, o que facilita a depuração e a análise dos resultados.

Figura 23 – Tela de especificações.



Fonte: O autor.