Sistema de Agendamento de Serviços Acadêmicos

Antonio Cauê Oliveira Morais
UFERSA
Pau dos Ferros, Brasil
antonio.morais36234@alunos.ufersa.edu.br

Cristiana de Paulo
UFERSA
Pau dos Ferros, Brasil
cristiana.paulo@alunos.ufersa.edu.br

Eriky Abreu Veloso
UFERSA
Pau dos Ferros, Brasil
eriky.veloso@alunos.ufersa.edu.br

Francisco Renan Leite da Costa
UFERSA
Pau dos Ferros, Brasil
francisco.costa57942@alunos.ufersa.edu.br

Geísa Morais Gabriel
UFERSA
Pau dos Ferros, Brasil
geisa.gabriel@ufersa.edu.br

Lavínia Dantas de Mesquita
UFERSA
Pau dos Ferros, Brasil
lavinia.mesquita@alunos.ufersa.edu.br

Lívia Beatriz Maia de Lima
UFERSA
Pau dos Ferros, Brasil
livia.lima30332@alunos.ufersa.edu.br

Maria Lanuza dos Santos Silva
UFERSA
Pau dos Ferros, Brasil
maria.silva44240@alunos.ufersa.edu.br

Tiago Amaro Nunes
UFERSA
Pau dos Ferros, Brasil
tiago.nunes@alunos.ufersa.edu.br

***Resumo*—Este artigo focaliza o processo de desenvolvimento de um sistema por meio de técnicas de Verificação e Validação de Software, bem como de Métodos Formais. Logo, aborda a necessidade da criação do Sistema de Agendamento de Serviços Acadêmicos como um meio viável para resolver a problemática na realização de agendamentos dos serviços prestados pela Universidade Federal Rural do Semi-Árido (Ufersa) para discentes. Dessa forma, pretende-se evidenciar o dever da aplicação de processos formais e técnicas de qualidade de software em um sistema real. Com isso, procura-se entender mais detalhadamente a abordagem utilizada para cada um dos mecanismos supracitados, assim como retratar a relevância de construir sistemas conforme os aspectos de produção determinados pela engenharia de software. O presente trabalho justifica-se por relacionar aspectos de apoio ao desenvolvimento de sistemas visando reduzir erros, melhorar a comunicação e facilitar a manutenção por meio da integração e automatização dos processos. Assim, este artigo representa uma pesquisa de tipologia teórica a partir da criação de documentos, manuais e relatórios sobre o Sistema ASA, assim como da perspectiva prática do desenvolvimento de software em linguagem de programação C#. Revela-se, por meio deste, a pertinência em aplicar estratégias de aprimoramento de software com o uso de conceitos formais e na identificação de defeitos. Sobre tal conjuntura, pode-se notar a necessidade de aperfeiçoamento do sistema após a realização dos testes executados, bem como da conformidade dos contratos aplicados com as regras específicas de cada requisito, com base no tratamento dos atributos.**

***Index Terms*—Sistema de Agendamento de Serviços Acadêmicos, Verificação e Validação, Métodos Formais.**

I. Introdução

Do processo empregado para melhorar a qualidade dos produtos e a produtividade no desenvolvimento do software, define-se Verificação e Validação (V&V) [5]. Por meio da verificação, é possível avaliar um sistema a fim de analisar se os artefatos produzidos em determinada fase do desenvolvimento correspondem às condições impostas anteriormente. Já no que diz respeito ao processo de avaliar se um determinado sistema, no final do seu desenvolvimento, cumpre com os requisitos estabelecidos, conceitua-se validação [6].

No processo de qualidade, o teste é a principal técnica de Verificação e Validação utilizada, contribuindo na avaliação, controle e confiança do software [13]. Entretanto, por mais que o emprego de testes comprovem uma determinada hipótese sobre o sistema com base em experimentos, a ausência de um caso que destoe do comportamento esperado enfraquece o método científico abordado. Nesse sentido, surge a possibilidade de utilizar um método matemático para garantir que programas estão corretos ou apresentam certas propriedades [3]. Assim, os métodos formais para computação estabelecem maneiras de construir e analisar sistemas com base em métodos matemáticos [3]. Logo, os métodos formais podem ser utilizados como suporte à geração de bons conjuntos de casos de teste e até mesmo para orientar no encontro de soluções mais elegantes, simples e eficientes [3].

Dado o desenvolvimento de sistemas, integrar atividades e automatizar processos está cada vez mais presente no mundo moderno [14]. Nesse ínterim, a partir do estudo de viabilidade, resolveu-se construir o sistema para agendar serviços acadêmicos, a fim de facilitar o suporte e disponibilidade dos serviços ofertados, bem como centralizar os serviços em um só local. Procura-se, com isso, facilitar o acesso aos profissionais e tornando o processo de agendamento mais transparente e atrativo para os universitários da Universidade Federal Rural do Semi-Árido (Ufersa) campus Pau dos Ferros.

Este artigo aborda as etapas necessárias durante o desenvolvimento de sistemas, assim como a definição e processo das atividades de V&V e de métodos formais ao longo da construção de softwares. Nesse sentido, a partir do estudo de viabilidade definido para a implantação web do Sistema de Agendamento de Serviços Acadêmicos, faz-se necessário aplicar estratégias de V&V que contribuam para o aperfeiçoamento e qualidade da tecnologia criada. Além disso,

urge compreender e aplicar técnicas de especificação formal para sistemas de software. Assim, durante este estudo, serão definidas técnicas de controle e garantia de qualidade, bem como ferramentas de suporte ao processo de desenvolvimento de software, considerando não apenas aspectos funcionais do sistema, mas também quesitos de avaliação para um adequado gerenciamento do software.

II. Fundamentação Teórica

Esta seção, possui como finalidade expor os conceitos necessários para o melhor entendimento do assunto abordado. Dessa forma, é apresentada a definição de sistemas web; logo após, são relatados aspectos associados à qualidade de software; depois disso é definido o conceito e ramificações a respeito do teste de software. Além disso, são apresentadas a documentação realizada na elaboração de sistemas, bem como as tecnologias e ferramentas utilizadas.

A. Sistemas Web

Um sistema Web é, normalmente, constituído por um conjunto de páginas Web e por uma base de dados [4]. Esses sistemas utilizam a arquitetura cliente-servidor. Neles, a comunicação do cliente no navegador ocorre através do endereço IP (*Internet Protocol*) do servidor, responsável por hospedar a estrutura de dados [8]. Nesse sentido, espera-se desenvolver um sistema Web de agendamento de serviços acadêmicos para a Ufersa, Campus Pau dos Ferros, que facilite a disponibilidade dos serviços ofertados por meio da integração e automatização do processo de agendamento. Além disso, deve-se aplicar técnicas de Verificação e Validação de Software a fim de incluir atividades de garantia da qualidade do software.

B. Qualidade de software

Historicamente, empresas perdiam bilhões de dólares em software devido a más implementações de funcionalidades. Sob esse aspecto, a qualidade de software refere-se à avaliação de um produto que atenda aos requisitos solicitados pelo cliente. O conceito de qualidade estende-se, portanto, em dois segmentos: qualidade do processo e qualidade do produto [15]. Nesse sentido, a garantia da qualidade de software engloba atividades de apoio aos processos, a fim de construir produtos adequados para o uso pretendido. Logo, a área de garantia da qualidade empenha-se em verificar os processos e suas definições, bem como validar o produto final a partir de um ciclo de revisões, melhoria e adaptação contínua [15].

C. Teste de software

Em sistemas de software cada vez mais complexos, a prática de testes para a descoberta de inconsistências torna-se cada vez mais necessária. Assim, a aplicação de testes, além de identificar a presença de erros, é fundamental para a garantia de qualidade e validação dos requisitos de um sistema de software [13]. Na realização de testes, cria-se um conjunto de casos de teste, ou seja, uma gama de condições que descrevem os comportamentos a serem testados no software a partir da análise das especificações do sistema [10]. Para tal, são estabelecidas condições necessárias a fim de determinar se o teste passou ou falhou.

Com base nisso, os testes podem ser definidos em dois aspectos gerais - a estrutura interna do programa e o seu aspecto funcional. Assim, os testes podem ser direcionados para o exercício da lógica dos componentes internos (teste de caixa branca) ou ainda podem ser voltados para descobrir erros no funcionamento, comportamento e desempenho do programa (teste de caixa preta) [10]. A organização dos testes é ainda efetuada em etapas, adotadas através das técnicas de teste de unidade, de integração e de sistema. Nesse viés, o teste unitário foca na garantia do funcionamento correto do componente do sistema. Do agrupamento entre os componentes origina-se o teste de integração. Após a integração, é realizado o teste de sistema, responsável por verificar o desempenho adequado da combinação dos elementos [11].

D. Documentação de sistemas

A documentação de software é essencial por dois motivos principais, uma das razões é facilitar a comunicação no processo de desenvolvimento do projeto e outra para esclarecer o conhecimento do programa nas atividades de manutenção (Ambler apud de [2]). Para serem elaborados os mais variados casos de testes, é necessário o entendimento sobre aquilo que está sendo testado. A documentação atua, portanto, como uma facilitadora, esclarecendo precisamente ao testador o comportamento esperado do programa, bem como os possíveis desvios de fluxo.

E. Tecnologias e ferramentas

Devido à crescente complexidade dos softwares, irrompe a necessidade de sistematizar tarefas, a fim de torná-la menos suscetível ao erro humano e menos custosa [1]. Para tal, apresentam-se algumas tecnologias e ferramentas de apoio ao processo de Verificação e Validação e de Métodos Formais.

1) SonarLint: SonarLint [1] é um *plugin* para IDE gratuito e de código aberto responsável por encontrar e corrigir problemas de codificação. A disposição para o uso da ferramenta está em garantir a qualidade do código e aumentar a produtividade na resolução de problemas. A tecnologia possui suporte para mais de 20 linguagens e usa mais de 5000 regras de *Clean Code* específicas de linguagem que buscam identificar erros comuns de codificação, *bugs* e vulnerabilidades.

2) NUnit: O NUnit [2] é um *framework* de teste unitário para todas as linguagens .Net. No processo de elaboração e identificação dos testes, o NUnit usa atributos personalizados e fornece um conjunto de asserções como métodos estáticos da classe Assert. Para realizar testes unitários utilizando o *framework*, o código de teste desenvolvido deve conter asserções capazes de demonstrar o correto funcionamento da funcionalidade testada. Posto isto, os principais tipos de asserções estão entre as de igualdade, comparação, condição, identidade e tipos.

[1]Para mais informações: https://docs.sonarsource.com/sonarlint/eclipse/
[2]Para mais informações: https://nunit.org/

3) Selenium: O Selenium [3] é caracterizado como uma ferramenta para automação de testes de aplicação web. Sendo portátil e possuindo código aberto, o Selenium oferece suporte para diversos navegadores web, aplicações web e tecnologias. O Selenium pode ser definido como um conjunto de diferentes ferramentas de software, cada qual com um objetivo específico a fim de auxiliar o processo de automação de testes baseado nas principais necessidades para testes em aplicação web [12].

4) DevTools e Lighthouse: O Chrome DevTools [4] é um conjunto de ferramentas para desenvolvedores da Web integrado diretamente ao navegador Google Chrome. Com ele, é possível diagnosticar problemas em tempo real, o que colabora no tempo de criação e edição de sites. O Lighthouse [5] é uma ferramenta automatizada de código aberto incorporada ao DevTools e criada para melhorar a qualidade das páginas da Web. Por meio da execução dos testes usando a ferramenta, é possível gerar um relatório sobre o desempenho da página web.

5) Code Contracts: O Code Contracts[6] é uma ferramenta de plataforma .NET, que permite a inserção de contratos no código-fonte e facilita a verificação formal, para garantir que o software se comporte de maneira correta e previsível. Ele introduz três tipos de contratos: pré-condições, que especificam o que deve ser cumprido antes da execução de um método; pós-condições: Que estabelecem o estado ideal de certos artefatos após a execução do método; e invariantes: Que garantem que certas condições permaneçam no mesmo estado durante o ciclo de vida de um objeto e validando a integridade do estado interno da classe. Esses contratos são validados tanto no tempo de execução quanto na compilação, auxiliando na detecção de erros lógicos no sistema, além de servirem de documentação implícita, descrevendo o comportamento esperado sem anotações externas.

III. Trabalhos Relacionados

O trabalho redigido pelos autores [9], é referente a elaboração de um software com intuito da informatização de processos e gestão do setor de Assistência Estudantil. No decorrer do artigo é destacado o processo de criação e desenvolvimento do software com proposito de otimizar os serviços relacionados a moradia estudantil.

Análogo, o trabalho de Helen [7], aborda uma análise de impacto em utilizar softwares para a automatização dos processos de gestão acadêmica. Nessa perspectiva, a autora ainda destaca, que um dos problemas da evasão dos alunos na adesão dos serviços é a demora pela conclusão do processo.

Ambos trabalhos demonstram a importância da automatização dos processos vinculados a gestão acadêmica, assim percebem-se alguns pontos em comum deste trabalho aos demais. Primeiramente por se tratar de temas de gestão acadêmica, sendo neste trabalho um nicho mais aprofundado em serviços relacionado a psicologia, nutrição e afins. Segundo, por se tratar da automatização dos processos e efetividade dos serviços realizados na universidade.

IV. Abordagem

Este estudo adota uma abordagem de cunho quali-quantitativo, de natureza aplicada e com o objetivo exploratório sobre a construção e análise do Sistema de Agendamento de Serviços Acadêmicos ou Sistema ASA. Para tal, os resultados obtidos no alicerce da metodologia empregada são detalhados sobre a visão no que diz respeito à documentação criada para o sistema, bem como na aplicação prática do desenvolvimento do software.

Nesse ínterim, na tentativa de aplicar uma abordagem sistemática e disciplinada para o desenvolvimento do software, o trabalho partiu, inicialmente, da criação da identidade visual e padronização de informações do sistema, com o Manual de Identidade Visual e Prototipagem do Software. Posteriormente, criou-se a documentação do sistema com finalidade de registrar as informações que caracterizam o software em sua totalidade. A partir disso, foi elicitado os requisitos funcionais e não funcionais, realizado a modelagem do sistema por meio de diagramas de caso de uso e de classe e também realizado a modelagem do banco de dados.

Sobre o modelo do sistema, foi designado o uso do padrão de projeto criacional *Singleton* e a linguagem de programação C# utilizando para o back-end o framework ASP.NET Core Web API e para o front-end o framework Blazor WebAssembly, diante de uma arquitetura cliente-servidor. Para a persistência dos dados foi utilizado o banco de dados MySQL e o Entity Framework Core, onde permitiu com essa estrutura criar e manter o sistema.

A estrutura do back-end determina-se com separações do código em camadas distintas para garantir a coesão do sistema, sendo: controllers, services e models. A camada de controllers lida com as requisições HTTP (do inglês *Hypertext Transfer Protocol*) e transfere as operações para a camada de services, onde são feitas as regras de negócio e validações. Já a camada de models define as entidades do sistema, referenciando as tabelas no banco de dados. Essa separação contribuiu, durante o processo de desenvolvimento, na adaptação do sistema quanto a manutenção e correções, o que acabou favorecendo também a conformidade do código com os requisitos do sistema. No que toca à segurança, foi utilizado a autenticação por meio dos *tokens* JWT - JSON Web Token - com intuito de fornecer segurança na manipulação de dados.

Na construção da documentação do código-fonte elaborada em todas as classes do projeto, foi utilizado os comentários com elementos XML para assim definir uma documentação de saída. A documentação começa com /// e contém algumas marcações, sendo ¡summary¿, ¡param¿, ¡return¿ e ¡exception¿ que corresponde a respectivamente, exibir informações sobre o bloco de código; descrever os parâmetros; informa o valor retornado; especifica os tipos de exceções que o método pode retornar. Após as anotações realizadas, é gerado toda a

[3]Para mais informações: https://www.selenium.dev/documentation/

[4]Para mais informações: https://developer.chrome.com/docs/devtools?hl=pt-br

[5]Para mais informações: https://developer.chrome.com/docs/lighthouse/overview?hl=pt-br

[6]Para mais informações: https://learn.microsoft.com/en-us/dotnet/framework/debug-trace-profile/code-contracts

documentação do código-fonte. A vantagem de utilizar essa estrutura de comentários visa a padronização e coesão do entendimento do código.

A estruturação do front-end do sistema foi realizada por meio da distribuição de arquivos principais, onde foram organizadas as funcionalidades essenciais da aplicação. Essa abordagem visa facilitar a manutenção e a escalabilidade do projeto, garantindo uma melhor compreensão do código.

Dentre esses arquivos, temos, primeiramente, a pasta de Layout, que contém os componentes reutilizáveis, como cabeçalhos e barras de navegação. Esses elementos são compartilhados e utilizados nas páginas da aplicação, que estão contidas no diretório Pages, cada uma associada a uma rota específica.

No que diz respeito aos elementos de estilização e aos recursos estáticos utilizados no sistema, como imagens e códigos JavaScript, esses foram organizados na pasta wwwroot. Na pasta Services, foram implementados serviços que realizam a comunicação e o consumo das APIs desenvolvidas no back-end, fazendo requisições HTTP, as quais são, então, integradas às páginas do sistema para serem utilizadas.

Com relação à autenticação dos usuários, foi empregado os serviços do Blazor para realizar a autenticação dos usuários utilizando os tokens JWT fornecidos pelo back-end. Esses tokens são armazenados no localStorage, permitindo que o sistema verifique a autenticidade do usuário.

Para garantir o controle de qualidade, foi elaborado um plano de teste que inclui o escopo, metodologia e objetivos que guiam a avaliação do sistema. Dessa forma, foram escolhidos os testes de caixa branca e de caixa preta para a verificação e validação das funcionalidades do ASA, em diferentes níveis de testes. Sendo assim, serão conduzidos testes de unidade, integração e de sistema com auxílio de ferramentas de apoio como NUnit, Selenium, SonarLint, DevTools e Lighthouse.

Tratando da execução dos testes, inicialmente foram realizados os testes estruturais utilizando a ferramenta NUnit como forma de automatização de execução de teste. Assim, fez-se necessário um estudo do sistema diante da documentação para iniciar a realização dos casos de teste. Diante disso, foi escolhido as classes e métodos que tinham maiores prioridades para serem testadas, sendo o models e o services por conter as informações vinculadas ao banco de dados. Posterior também foram testadas as classes controllers e pages com intuito de aumentar a cobertura de linhas no sistema.

No que diz respeito aos testes funcionais, realizou-se os testes de responsividade em todas as páginas do sistema diante das métricas de *mobile* (375x780), desktop (1920x1080) e tablet (768x1280), tendo como intuito a verificação do sistema em se adaptar a diferentes plataformas. Seguinte, foram feitos testes de portabilidade com a ferramenta Selenium para a automatização de simulações de ações de usuário, onde foi testada a página de Entrar do sistema nos navegadores Chrome, Firefox e Edge.

Ademais, os testes relacionados a qualidade, inicialmente foram realizados testes de qualidade de página para colher as métricas de desempenho, boas práticas e acessibilidade. Foi utilizado a ferramenta Lighthouse para automatizar esse processo realizado em todas as páginas do sistema. A execução é bem simples e ocorre ao gerar um arquivo de auditoria na página que deseja com a extensão no navegador ou na opção Lighthouse do DevTools.

Para os testes de análise estática do código-fonte foi utilizada a ferramenta SonarLint. Sobre essa perspectiva, foi identificado três tipos de regras definidas pela ferramenta, variando entre regras de *bug* e de *code smell*. Além disso, os problemas identificados revelaram impacto sobre a manutenibilidade e confiabilidade do código-fonte, variando entre atributos de código limpo de adaptabilidade e intencionalidade.

Quanto aos testes funcionais, a abordagem ocorreu de forma exploratória. Nesse sentido, os testes realizados para o referido sistema procuraram identificar erros e comportamentos inesperados a partir da aprendizagem, experiência, conhecimento e observação do testador. Para a execução, devido à natureza desse tipo de teste, não foi preciso criar casos de teste predefinidos, tornando a abordagem criativa e flexível. Por fim, após a análise diante dos testes foi elaborado um relatório contendo os tipos de testes efetuados, bem como os erros e/ou falhas identificados em cada granularidade do teste.

Após o desenvolvimento das principais funções responsáveis por garantir os requisitos do sistema, localizadas especialmente em DiscenteService.cs, AgendamentoService.cs e ServicoController.cs, foram aplicados contratos de código visando testar a eficácia dessas implementações. Utilizando Code Contracts, foram definidas pré-condições e pós-condições para garantir que as entradas estivessem corretas antes da execução dos métodos e a validação dos resultados após sua conclusão. As invariantes não foram necessárias devido à lógica utilizada na criação das funções, enquanto alguns requisitos foram automaticamente tratados pela estrutura do banco de dados, eliminando a necessidade de abordá-los por meio de contratos.

Dessa maneira, o estudo tem como principal característica a implementação e manutenção do sistema proposto dentro dos aspectos de métodos formais e de verificação e validação abordados.

V. Considerações Finais e Trabalhos Futuros

Nessa seção, são apresentadas as considerações finais do presente artigo, juntamente com pontos para trabalhos futuros.

Assim, o objetivo principal do trabalho foi desenvolver um sistema de agendamento de serviços acadêmicos da UFERSA, campus Pau dos Ferros, que tornasse unificado e transparente tal processo, bem como agilizar o agendamento desses serviços aos discentes.

O sistema conta com a visão implementada do discente e, como próximo passo, almeja a implementação dos protótipos das telas que contêm a visão do profissional, garantindo a completude das funcionalidades do sistema. Pretende-se também realizar a implantação completa do sistema, a fim de que toda a comunidade da Ufersa possa utilizá-lo.

As análises com os testes executados demonstram que o sistema ainda precisa de manutenção, como em destaque de

responsividade. Dessa forma, tem-se como objetivo também sanar os erros e *bugs* identificados durante a realização dos testes estruturais e funcionais. Ademais, com os contratos e documentação realizada, foi possível reconhecer a importância dessa abordagem para o entendimento do domínio do sistema.

No entanto, é importante reconhecer o impacto e os aprendizados proporcionados pelo desenvolvimento do sistema ASA. Vale destacar que o desenvolvimento apresentou diversos desafios, como a integração de tecnologias distintas e a adaptação do sistema às necessidades dos usuários, que demandou um tempo além do que era fornecido, fazendo com que a equipe montasse estratégias para a sua conclusão.

Em suma, este trabalho contribuiu com a importância da aplicação de abordagens de verificação e validação, bem como a construção da especificação formal para o aprimoramento contínuo do sistema, tendo como destaque uma melhor gestão dos serviços prestados pela universidade.

REFERÊNCIAS

[1] BOAS, A. *Gestão de configuração para teste de software*. PhD thesis, Dissertação de Mestrado, FEEC/UNICAMP, Campinas, SP, 2003.

[2] DE SOUZA, S. C. B., DAS NEVES, W. C. G., ANQUETIL, N., AND DE OLIVEIRA, K. M. Documentação essencial para manutenção de software ii. In *IV Workshop de Manutenção de Software Moderna (WMSWM), Porto de Galinhas, PE* (2007).

[3] DÉHARBE, D., MOREIRA, A. M., RIBEIRO, L., AND RODRIGUES, V. M. Introduçao a métodos formais: Especificaçao, semântica e verificaçao de sistemas concorrentes. *Revista de Informatica Teorica e Aplicada. Porto Alegre. Vol. 7, n. 1 (set. 2000), p. 7-48* (2000).

[4] DELAMARO, M. E., MALDONADO, J. C., AND JINO, M. *Introdução ao Teste de Software*. Elsevier, 2016.

[5] HIRAMA, K. *Engenharia de Software*. Grupo GEN, 2011.

[6] IEEE. Ieee standard glossary of software engineering terminology. Standard ANSI/IEEE Std 610.12-1990, IEEE, New York, 1990.

[7] MARQUES, H. M. F. *Gestão em processos acadêmicos: estudo de caso de uma Instituição de Ensino Superior privada na cidade de São Luis–Maranhão-Brasil*. PhD thesis, Escola Superior de Educação João de Deus, 2019.

[8] MARTHA, L. F. *Desenvolvimento de uma aplicação web para modelagem colaborativa*. PhD thesis, PUC-Rio, 2022.

[9] MELO, E. C., DA SILVA, G., AND CRUZ, M. A. Gestão informatizada de um setor de assistência estudantil:: Um estudo de caso no ifmg–campus. *International Journal of Management-PDVG 1*, 1 (2021).

[10] PRESSMAN, R. S. *Engenharia de software*, 7 ed. AMGH, Porto Alegre, 2011.

[11] PRESSMAN, R. S., AND MAXIM, B. R. *Engenharia de software*. McGraw Hill Brasil, 2016.

[12] SANTORI, R. P., ET AL. Avaliação da ferramenta de testes selenium no desenvolvimento guiado por teste de uma aplicação web. *Universidade de Brasília* (2019).

[13] SOMMERVILLE, I. *Engenharia de software*. Pearson Universidades, 2011.

[14] VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Editora: Independente, 2020.

[15] ZANIN, A., JÚNIOR, P. A. P., AND ROCHA, B. C. *Qualidade de software*. Grupo A, 2018.