SIMULT-PDF 2.0

Geísa Morais Gabriel
PEX1271 - Teste de Software
UFERSA
Pereiro, Brasil
geisa.gabriel@alunos.ufersa.edu.br

Leonardo Inácio Guilherme Dantas
PEX1271 - Teste de Software
UFERSA
Pau dos Ferros, Brasil
leonardo.dantas69361@alunos.ufersa.edu.br

Lívia Beatriz Maia de Lima
PEX1271 - Teste de Software
UFERSA
Pau dos Ferros, Brasil
livia.lima30332@alunos.ufersa.edu.br

***Resumo*—O Sistema de Cadastramento de Multas de Pau dos Ferros (SIMULT-PDFv2) surge como uma solução de inovação para o monitoramento de infrações de trânsito em conformidade com o Código de Trânsito Brasileiro (CTB). Inicialmente desenvolvido em Portugol e subsequentemente aprimorado para uma nova versão desenvolvido em Java, este sistema tem como objetivo facilitar o cadastramento e monitoramento de multas de trânsito. A necessidade do sistema como o SIMULT-PDF se origina por falta de ferramentas eficazes para a realização do monitoramento consistente e preciso das infrações. Nesse cenário, foi conduzido teste de *software* no sistema, aplicando conceitos de qualidade fundamentados em Pressman e Sommerville. Dessa forma, a criação do sistema torna-se relevante uma vez que satisfaz a necessidade organizacional em classificar e atribuir multas a veículos que infrigem as normas estabelecidas pelo CTB. Posto isso, o SIMULT simplifica o gerenciamento de infrações de trânsito, promovendo eficiência e consistência no cadastramento de multas e aplicações de leis. Assim, contribui para a segurança e gestão eficaz do trânsito em Pau dos Ferros. Conclui-se, portanto, que o SIMULT desempenha um papel significativo na gestão de multas de trânsito, aprimorando a eficiência administrativa.**

***Index Terms*—Multas de trânsito; Sistema de Cadastro de Multas; Teste de Software;**

I. Introdução

O Sistema de Multas de Trânsito de Pau dos Ferros (SIMULT-PDF) foi criado no intuito de cadastrar e monitorar a geração de multas provocadas devido ao desrespeito com as normas que regem o Código de Trânsito Brasileiro (CTB). O atual sistema é consequência do aprimoramento do sistema antigo, implementado em linguagem de programação Portugol. A partir disso, o SIMULT-PDF 2.0, traduzido para a linguagem Java, conta com o acréscimo de novas funcionalidades, bem como a geração de casos de teste a fim de garantir um melhor desempenho do programa.

Assim, procura-se expor os passos realizados a partir dos testes de caixa branca e de caixa preta implementados sobre o sistema. Com isso, deseja-se evidenciar a relevância que os testes desempenharam para a avaliação das funcionalidades implementadas, bem como na detecção e prevenção de erros e incosistências, garantindo um desempenho aceitável durante a usabilidade do sistema.

Logo, o artigo é estruturado da seguinte maneira: na Seção II é apresentado todo o embazamento teórico necessário para o melhor entendimento do assunto; na Seção III detalha-se as abordagens de teste de *software* realizadas; e por último, na Seção IV são apresentados as considerações finais e as sugestões para os trabalhos futuros.

II. Fundamentação Teórica

Esta seção possui como finalidade expor os conceitos necessários para o melhor entendimento do assunto especificado. Dessa forma, são abordados os tópicos referentes aos requisitos do sistema; em seguida é apresentado o Código de Trânsito Brasileiro (CTB); logo após é relatado o padrão de arquitetura de *software* denominado *Model-View-Controller (MVC)*; depois disso é definido o conceito de *Unified Modeling Language (UML)* e posteriormente ocorre a explicação sobre teste de *software*. Além disso, são apresentados a ideia de padrões de projeto, e o conceito qualidade de *software*, bem como o paradigma de programação orientada a objetos, a documentação de *software* e, por último, as estratégicas e técnicas utilizadas nos testes.

A. Requisitos do sistema

Os requisitos correspondem ao conjunto de tarefas destinadas ao sistema. Dessa forma, procura-se estabelecer que as ações desempenhadas pelo *software* estejam em conformidade com as demandas dos clientes. Do mesmo modo, a partir dos requisitos também é possível conhecer as limitações do sistema (SOMMERVILLE, 2011). Assim, os requisitos funcionais são as ações desempenhadas pelo sistema de acordo com as entradas especificadas, enquanto os requisitos não funcionais são as restrições impostas (SOMMERVILLE, 2011).

B. Código de Trânsito Brasileiro

O Código de Trânsito Brasileiro (CTB) está associado à Lei Nº 9.503. Sob esse aspecto, a finalidade do documento é explicada pelo artigo 1º, que justifica o uso desse Código como um guia de trânsito nas vias terrestres do território nacional. Nesse sentido, a existência do CTB serve para garantir a melhor organização do trânsito, a partir do qual é possível elencar infrações, bem como a criação de multas. Nesse sentido, qualquer desrespeito para com a Lei implica na geração de multas como penalidade da infração cometida no trânsito.

C. Model-View-Controller

O *Model-View-Controller* (MVC) é um padrão de arquitetura de *software*. (PRESSMAN; MAXIM, 2016) Por meio dele

é possível dividir as responsabilidades no desenvolvimento do *software*. Com base nisso, o MVC é dividido em três camadas ou componentes: o *model* é a camada responsável pela manipulação dos dados, a *view* é o componente que interage com o usuário e a camada *control* realiza o controle entre o *model* e a *view*. A partir desse padrão pode-se realizar uma simplificação da estrutura que contempla apenas duas camadas, *model* e *control*. Nesse sentido, é possível abster a camada de interação com o usuário nas primeiras versões da arquitetura MVC.

D. Unified Modeling Language

O *Unified Modeling Language* (UML) é uma linguagem utilizada para modelar softwares baseados no paradigma de orientação a objetos, aplicada principalmente durante as fases de análise de requisitos e projeto de *software*. (GUEDES, 2018) Fundamentado nisso, a UML foi utilizada no decorrer da modelagem do diagrama de caso e uso e no diagrama de classes. Primeiramente, na determinação da relação dos comportamento requerido do sistema diante da perspectiva do usuário na busca de atingir os requisitos, e em segunda instância na modelagem dos objetos e das relaçẽos que compõem o sistema.

E. Definição de Teste de Software

Sistemas de *software* são programas complexos e, por essa razão, suscetíveis a erros e inconsistências. Desse modo, o teste de *software* visa evidenciar os defeitos de um sistema antes mesmo que este seja lançado ao mercado (SOMMERVILLE, 2011) Sobre tal aspecto, o teste de *software* é um conjunto de atividades estabelecidas após um planejamento, para isso são definidas etapas que, se efetivadas, garantem a qualidade do produto desenvolvido, fundamental para o sucesso do *software* (PRESSMAN; MAXIM, 2016)

F. Padrões de Projeto

Padrões de projeto são tipos de soluções para problemas frequentes no desenvolvimento de um *software*. Sobre tal aspecto, fez-se uso do padrão de projeto criacional denominado Singleton. Sendo assim, é possível garantir apenas uma instância em uma classe, ao mesmo tempo que se pode concede um ponto de acesso global para essa instância. Com isso, espera-se promover mecanismos que aumentam a flexibilidade e reutilização de código já existente. (GURU, 2023)

G. Qualidade de software

Historicamente, empresas perdiam bilhões de dólares em *software* devido a más implementações de funcionalidades. Logo, precisou ser definido o conceito de qualidade de *software*, que diz respeito, em âmbito geral, a aplicação de uma infraestrutura que crie um produto que atenda aos requisitos, seja confiável, livre de erros e que forneça os benefícios esperados para aqueles que o produzem e para aqueles que o utilizam (PRESSMAN; MAXIM, 2016).

H. Paradigmas de Orientação a Objetos

A Programação Orientada a Objetos é um modelo de projeto e programação de *software* que procura relacionar o mundo digital com o mundo real por meio da relação entre os componentes de um programa (WIKIPÉDIA, 2023). Para isso, os componentes se combinam com o domínio do problema ao mesmo tempo que fornecem apoio a serviços para o determinado domínio do problema. (PRESSMAN; MAXIM, 2016)

I. Documentação de software

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 (SOUZA *et al.*, 2007)). Para que sejam 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.

J. Estratégias e Técnicas de Testes

Para que seja possível realizar os testes, é necessário criar 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(WIKIPÉDIA, 2020). Para isso, é relevante estabelecer as condições necessárias para 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, denominando-se teste de caixa branca. Ademais, os testes ainda podem ser voltados para descobrir erros no funcionamento, comportamento e desempenho do programa, sendo definido como teste de caixa preta. (PRESSMAN, 2011)

III. Abordagem

Nesta seção são apresentadas as estratégias utilizadas e os caminhos estabelecidos para a execução efetiva do sistema. A partir disso, é tratado o processo de estabelecimento dos requisitos, o entendimento acerca dos artefatos para a melhor construção do diagrama de caso e uso e do diagrama de classes. Além disso, foram determinadas as regras de negócio, bem como as entidades e atributos necessários para a construção do modelo de banco de dados.

Antecedendo ainda o processo até a finalização da nova versão do programa, vale salientar que, no que diz respeito à elaboração e execução dos testes, houve a divisão entre testes de caixa branca e testes da caixa preta. Sobre esse viés, no que tange aos testes do ponto de vista funcional, elaborou casos de testes de sistema. Já no que se refere aos testes estruturais, foram realizados testes de unidade, integração e em critérios baseado no fluxo de controle e no fluxo de dados do sistema.

Em primeira análise foi revisitado a versão anterior do SIMULT-PDF. A partir disso houve a descrição dos requisitos já presentes na última versão e o acréscimo dos novos requisitos necessários para a atualização do sistema. Sobre o modelo do sistema, foi designado o uso do padrão de projeto criacional *Singleton* e a linguagem de programação Java juntamente com o paradigma de programação orientada a objetos. Definiu-se também a aplicação *web* ElephantSQL para realizar o armazenamento no banco de dados PostgreSQL. Diante disso, ocorreu a estruturação do código com o início do desenvolvimento do programa. Assim, foi implementado progressivamente o que havia sido projetado em conjunto com a aplicação dos testes.

Desse modo, primeiro foram realizados sobre os métodos do sistema os testes de unidade. É importante destacar que esses testes não foram aplicados sobre as entidades, pois, em sistemas orientados a objetos não é necessário realizar testes sobre as particularidades do programa (PRESSMAN; MAXIM, 2016). Em relação aos testes de unidade, destaca-se ainda o uso do *framework open-source JUnit* para a criação de testes automatizados na linguagem de programação Java (WIKIPÉDIA, 2022), ressalva-se, portanto, que os testes feitos com essa ferramenta aconteceram apenas em classes que não necessitavam de qualquer entrada do usuário. Destaca-se como causa da particularidade citada a limitação do ambiente de desenvolvimento *IntelliJ IDEA*. Por essa razão, o restante dos testes foram feitos de forma manual.

A abordagem nos testes de integração ocorreu sobre os métodos de cada classe do programa. Dessa forma, pretende-se percorrer os caminhos do sistema a partir da execução dos casos de teste provenientes das especificações. Assim, a integração procura executar as relações existentes entre os métodos com o intuito de verificar possíveis falhas do sistema.

No que se refere aos testes de sistema, o processo foi conduzido em várias etapas e de modo gradual, desde o planejamento detalhado até avaliação de resultados, utilizando testes de unidade e integração durante o caminho até a realização dos testes sobre as interações entre os componentes. No teste de sistema, foi adotada a abordagem *"bottom-up"*, que envolve a verificação e validação das partes individuais. Consequentemente, essa estrátegia permite identificar falhas em componentes específicos antes de avaliar o funcionamento geral do sistema. Além disso, toda a documentação de teste e resultados foi devidamente registrada, incluindo casos de testes e registro de correções.

No que tange a parte de critérios de técnicas estruturais, foram utilizadas conceitualizações de teste de caminhos para a complexidade ciclomática diante de fluxo de controle e do fluxe de dados.

IV. Considerações finais e trabalhos futuros

Nessa seção são apresentados as considerações finais do presente artigo, juntamente com sugestões de melhorias para trabalhos futuros.

Durante o processo de teste do SIMULT, foi fundamental assegurar que o sistema cumprisse não só os requisitos e regras de negócios estabelecidos, mas também que fosse um programa confiavél e eficaz no cadastramento de multas.

Assim, ao longo do desenvolvimento, foram realizados testes abrangentes para a verificação das funcionalidades do programa, incluindo testes de unidade, integração e de sistema, com o intuito de testar as unidades individualmente, de forma integrada e, por fim, sobre o sistema como um todo. Assim, foi possível validar os requisitos e verificar o comportamento do *software* em diversos cenários.

Dessa forma, os resultados dos testes desempenharam um papel crucial na garantia do objetivo geral do sistema, cadastrando as multas de trânsito de acordo com as normas. Através desses testes, foi possível identificar e corrigir as inconsistências e potenciais erros do sistema.

Diante do que foi exposto, fica claro a importância da realização dos testes no SIMULT-PDFv2 para identificar as áreas de aprimoramento e desenvolvimento no futuro. No que diz respeito à usabilidade, recomenda-se a realização de estudos envolvendo coleta de feedback dos usuários, visando a possibilidade de tornar o acesso ao sistema mais amigável através, por exemplo, da criação da interface do SIMULT-PDFv2.

Além disso, uma funcionalidade a ser considerada para o futuro é a de denúncias, permitindo que cidadãos relatem infrações de trânsito, fornecendo fotos e detalhes para facilitar a aplicação da multa.

Referências

GUEDES, G. T. **UML 2-Uma abordagem prática**. [S.l.]: Novatec Editora, 2018.

GURU, R. **O que é um padrão de projeto?** 2023. Disponível em: ⟨https://refactoring.guru/pt-br/design-patterns/what-is-pattern⟩.

PRESSMAN, R. S. **Engenharia de software**. 7. ed. Porto Alegre: AMGH, 2011.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**. [S.l.]: McGraw Hill Brasil, 2016.

PúBLICA, M. da Justiça e S. **CÓDIGO DE TRÂNSITO BRASILEIRO**. Disponível em: ⟨https://www.gov.br/prf/pt-br/concurso-2021/codigo-de-transito-brasileiro\#c1⟩.

SOMMERVILLE, I. **Engenharia de software**. [S.l.]: Pearson Universidades, 2011.

SOUZA, S. C. B. de; NEVES, W. C. G. das; ANQUETIL, N.; OLIVEIRA, K. M. de. Documentação essencial para manutenção de software ii. In: **IV Workshop de Manutenção de Software Moderna (WMSWM), Porto de Galinhas, PE**. [S.l.: s.n.], 2007.

WIKIPÉDIA. **Caso de Teste**. 2020. Disponível em: ⟨https://pt.wikipedia.org/wiki/Caso_de_teste⟩.

WIKIPÉDIA. **JUnit**. 2022. Disponível em: ⟨https://pt.wikipedia.org/wiki/JUnit⟩.

WIKIPÉDIA. **Orientado Objetos**. 2023. Disponível em: ⟨https://pt.wikipedia.org/wiki/Orienta\%C3\%A7\%C3\%A3o_a_objetos⟩.