RBTI – Revista Brasileira em Tecnologia da Informação
Volume 5, número 2 – 2023
ISSN: 2675-1828
Editor Científico: Jaime Cazuhiro Ossada

Aplicação em Java desenvolvida como uso da POO e Padrões de Projetos

Christian Morgado Silva
Universidade CEUMA, christian-silva@live.com
Edílson Carlos Silva Lima
Universidade CEUMA, edilsonlima@ceuma.com.br

RESUMO

O projeto de urna eletrônica tem como principal objetivo disponibilizar de um software que possa simular uma pequena eleição, onde está pode ser utilizada em pleito para grêmio estudantil, líder de empresa, dentre outros, substituindo os votos em cédulas por algo mais moderno e eficaz. Os objetivos específicos são desenvolver um software rápido e eficaz, a linguagem utilizada na sua construção é o Java. Para a construção do software foi seguido as etapas de análise de requisitos, o uso da UML, o processo unificado, padrões de projeto e por fim realizar os testes de software.

Palavras-Chave: Processo unificado; POO; UML; Padrões de projeto.

Data do recebimento do artigo: 29/11/2022
Data do aceite de publicação: 30/07/2023
Data da publicação: 31/12/2023

Java application developed using OOP and Design Patterns

ABSTRACT

The electronic voting machine project has as main objective to provide a software that can simulate a small election, where it can be used in elections for student union, company leader, among others, replacing ballot votes for something more modern and effective. The specific objectives are to develop fast and efficient software, the language used in its construction is Java. For the construction of the software, the steps of requirements analysis, the use of the UML, the unified process, design patterns and finally the software tests were followed.

Key Words: Unified process; OPP; UML; Desing patterns.

1 Introdução

Quando é preciso escolher um representante para ocupar algum cargo dentro de um grupo, existem diversos processos formais que podem ser adotados para auxiliar o processo, desde processos simples onde os grupos são pequenos e a democracia pode ser falada e expressada verbalmente, ou processos um pouco mais complexo é conferido e divulgado por parte do grupo e ou podendo ser também registrado em ata.

Com o crescente avanço da tecnologia, onde cada vez mais processos e sistemas vem sendo informatizados para atender as necessidades e no melhoramento da qualidade de vida dos usuários para esse feito, a mesma tornou-se muito importante no cotidiano de muitas pessoas buscando melhorar as suas atividades diárias.

Partido desse pré-suposto e acompanhando a velocidade de como os processos se tornam cada vez mais eletrônicos nota-se ainda que algumas atividades ainda não estão totalmente relacionadas a tais atualizações, como: Escolhas de lideranças de salas de aula, eleições para escolha de agremiações estudantis, entre outros, ainda funcionam de maneira rudimentar.

No Brasil, o processo eleitoral, é um sentido amplo e diz respeito às fases organizacionais das eleições, que compreende também um período posterior. O Processo Eleitoral é organizado pela Justiça Eleitoral (JE), em nível municipal, estadual e federal. Na esfera federal, a JE possui como órgão máximo o Tribunal Superior Eleitoral (TSE), bem como juízes e juntas eleitorais (TSE, 2016).

A JE organiza, fiscaliza e realiza as eleições regulamentado o processo eleitoral, examinando as contas de partidos e candidatos em campanhas, controlando o cumprimento da legislação pertinente em período eleitoral e julgando os processos relacionados com as eleições. Neste contexto, é primordial que ao eleitor conheça e entenda do processo eleitoral.

Vivemos num regime democrático e a participação da população na democracia se dá pelo processo eleitoral. É necessário que as pessoas entendam como funcionam as eleições, democracia só funciona porque temos eleições e as eleições só acontecem com o voto livre do público. A votação, totalização dos votos e os resultados são as fases mais conhecidas do processo eleitoral, mas é importante considerar todo o processo para que se possa exercer de forma consciente o direito do voto.

Buscando melhorar essa e outras situações, este artigo tem por objetivo apresentar um software personalizado de votação eletrônica para pequenos negócios e eventos, onde

poderão ser feitas diversas eleições, como por exemplo escolha de síndico de condomínio, escolha de presidente de comunidades, entre outras, substituindo a famosa e conhecida votação em cédulas, otimizando a velocidade do resultados das eleições.

Com um login registrado pelo administrador do software, o usuário escolhe entre os candidatos cadastrados previamente para quem irá seu voto, sendo este feito dividido por cargos, tudo será salvo no banco de dados, isso incluirá logins dos usuários, informações dos candidatos e partidos além dos resultados da votação, que poderá ser salvo em formato PDF e consequentemente podendo ser impresso.

O software da urna eletrônica foi desenvolvido em linguagem JAVA através da IDE NetBeans seguindo os padrões de projeto processo unificado, por sua vez o banco de dados foi criado utilizando a linguagem MySQL através da IDE MySQL Workbench.

Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Diferente das linguagens de programação convencionais, que são compiladas para código nativo.

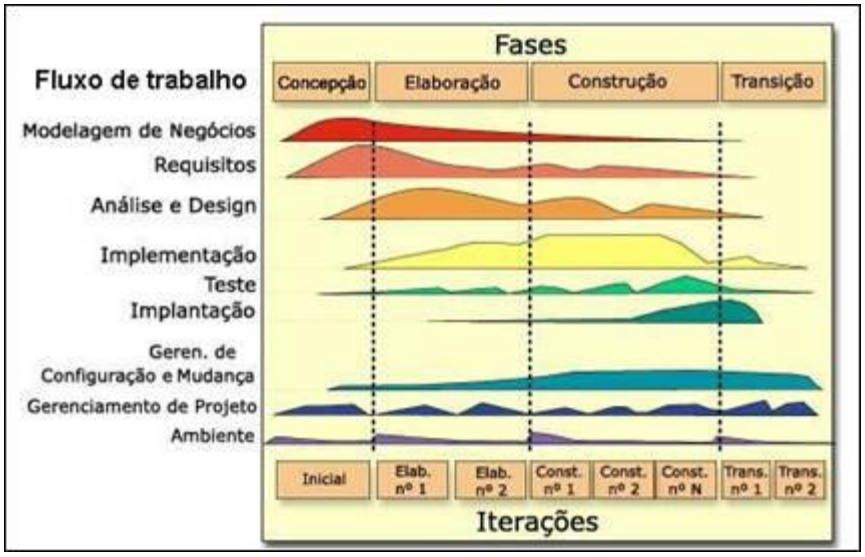
2 Referencial teórico

Neste capítulo, será abordada uma revisão literária dos assuntos essenciais para a elaboração de um aplicativo desenvolvido em linguagem Java e está dividido nos seguintes itens: 2.1 Processo unificado, no item 2.2 Programação orientada a objeto, no item 2.3 UML, no item 2.4 Padrões de projeto, e por fim no item 2.5 Teste de software.

2.1 Processo unificado

O Processo Unificado (RUP) é um processo de desenvolvimento que combina ciclos iterativos e incrementais para construir softwares, seu processo é dividido em 4 fases: Concepção, Elaboração, Construção e Transição, como mostra a figura 1 (AQUINO, 2008).

Figura 1 – Overview do processo unificado



Fonte: AQUINO (2008)

A primeira fase é a concepção, que é onde é definido o escopo do projeto, ou seja, um rascunho de como será a arquitetura do projeto, nessa fase são identificados os requisistos de negócio, esses requisitos são descritos através de casos de uso. Os objetivos desta fase são: criar o escopo do projeto, identificar os requisitos do sistema, custo geral do sistema e verificar os riscos em potencial.

A segunda fase é a elaboração, é nessa fase que é projetada uma base para a arquitetura do projeto, nela é discutido o que pode ser feito com as questões levantadas como restrições financeiras por exemplo. Os requisitos funcionais que foram identificados na fase anterior são aplicados na elaboração. Os objetivos desta fase são: obter uma "baseline" da arquitetura, verificar os riscos em potencial, componentes do sistema e reusabilidade.

A terceira fase é a construção, durante esta fase que o projeto é posto em prática, a arquitetura que foi elaborada na fase anterior agora é executável, ou seja, o sistema agora torna-se um produto com operações iniciais que poderá ser testado pelo cliente. Nesta fase os erros poderam ser encontrados e corrigidos. Os objetivos desta fase são: saber a qualidade do sistema, elaborar versões alfa e beta e release do sistema.

A quarta e última fase é a transição, é aqui que o projeto ganha sua primeira versão e é entregue aos usuários, os erros das versões alfa e beta são corrigidos e são adicionadas melhorias. Os objetivos desta fase são: teste beta e a distribuição do produto final.

2.2 Programação Orientada a Objeto

Segundo Wohlinc (2015) como na maioria das atividades que fazemos no cotidiano, programar também possui modos diferentes de se fazer. Esses modos são chamados de paradigmas de programação e, entre eles, estão a programação orientada a objetos (POO). Quando são utilizadas as linguagens como Java, C#, Python e outras que possibilitam o paradigma orientado a objetos, é comum cometer erros e aplicar programação estruturada deduzindo que estão sendo usados recursos da orientação a objetos.

A Orientação a Objetos na programação é um paradigma que corresponde a interação entre várias unidades que são denominadas objetos. Na programação orientada a objetos são definidos novos tipos através da criação de classes, e esses tipos podem ser instanciados criando objetos (GUERRA 2014). A ideia é que um objeto represente uma entidade concreta enquanto sua classe representa uma abstração dos seus conceitos. A Orientação a Objetos é utilizada para resolução de problemas de software baseados na vida real, ela surgiu com a intenção de aproximar o manuseio de software com coisas do mundo real. Esse paradigma tem como base dois principais conceitos, que são classes e objetos. Pode-se entender classe como uma descrição genérica dos objetos atribuídos a um conjunto, uma classe define o comportamento e as características de um conjunto de objetos. Por outro lado, o objeto é uma instância de uma classe, ou seja, ele é uma representação concreta da classe. Ele é capaz de armazenar estados por intermédio de seus atributos e também pode se relacionar com outros objetos.

Programas orientados a objetos são muitas vezes mais fáceis de entender, corrigir e modificar do que técnicas anteriormente conhecidas como programação estruturada (DEITEL 2016). Pode-se dizer que a principal diferença entre sistemas estruturados e orientados a objetos é o grau para o qual a inteligência (e, portanto, o fluxo de controle) é vertical ou horizontal (GILBERT 1998).

A Orientação a Objetos tem como principais pilares a herança, o encapsulamento, a abstração e o polimorfismo.

A herança na Orientação a Objetos permite a reutilização de código, pois de acordo com essa definição, é possível criar uma nova classe a partir de uma outra classe já existente, ou seja, essa nova classe será uma espécie de cópia da classe que a gerou, porém com algumas características adicionais.

O encapsulamento é utilizado na Orientação a Objetos para restringir o acesso a atributos, métodos ou classes, isto é, ficando ocultos ao usuário. O encapsulamento dos

atributos e métodos evita o vazamento de escopo, impedindo que esse atributo ou método fique visível para alguém que não deveria vê-lo, como outro objeto ou classe.

Uma classe é uma combinação de dados e as operações legais que podem ser realizadas nesses dados. Os dados em um sistema orientado a objetos, em vez de ser passado entre as operações (subprogramas), é ocultado e protegido do acesso por outras partes do programa, por meio de um princípio chamado encapsulamento (GILBERT 1998).

A abstração é uma forma de diminuir a complexidade e aumentar a efetividade do sistema. Na Orientação a Objetos esse conceito que dizer esconder os detalhes de uma implementação, ou seja, é uma maneira de facilitar escondendo aquilo que não for necessário.

O poliformismo na Orientação a Objetos é quando duas classes têm o mesmo método más a implantação é diferente para cada classe apesar de ambas terem o mesmo efeito.

2.3 UML

UML – *Unified Modeling Language* ou Linguagem de Moldelagem Unificada – é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos (GUEDES, 2011). É uma linguagem de modelagem de propósito geral que pode ser aplicada a todos os domínios de aplicação. Essa linguagem tornou-se, nos últimos anos, a linguagem-padrão de modelagem adotada internacionalmente e pela indústria de engenharia de software.

Cada diagrama da UML analisa o sistema, ou parte dele, sob uma determinada óptica. É como se o sistema fosse modelado em camadas, sendo que alguns diagramas enfocam o sistema de forma mais geral, apresentando uma visão externa do sistema, como é objetivo do Diagrama de Casos de Uso, enquanto outros oferecem uma visão de uma camada mais profunda do software, apresentando um enfoque mais técnico ou ainda visualizando apenas uma característica específica do sistema ou um determinado processo. A utilização de diversos diagramas permite que falhas sejam descobertas, diminuindo a possibilidade da ocorrência de erros futuros.

O diagrama de casos de uso é o diagrama mais geral e informal da UML, utilizado normalmente nas fases de levantamento e análise de requisitos do sistema, embora venha a ser consultado durante todo o processo de modelagem e possa servir de base para outros

diagramas. Apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma ideia geral de como sistema ira se comportar. Procura identificar os atores (usuários, outros sistemas ou até mesmo algum hardware especial) que utilizarão de alguma forma o software, bem como os serviços, ou seja, as funcionalidades que o sistema disponibilizará aos atores, conhecidas nesse diagrama como casos de uso (GUEDES, 2011).

Para BOOCH (2006), o diagrama de classes é provavelmente o mais utilizado e é um dos mais importantes da UML. Serve de apoio para a maioria dos demais diagramas. Como o próprio nome diz, define a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos que cada classe tem, além de estabelecer como as classes se relacionam e trocam informações entre si.

Por sua vez, o diagrama de objetos representa uma instância do diagrama de classes, ou seja, assim será possível visualizar como será implementado o sistema. Eles são utilizados para representar as instâncias de um sistema bem como o relacionamento entre elas.

O diagrama de objetos é uma variação do diagrama de classes que utiliza quase a mesma notação, a diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes, esse diagrama é útil para exemplificar diagramas complexos de classes ajudando muito em sua compreensão (RURA, 2015).

2.4 Padrões de projeto

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável.

Padrões de projeto estão num nível acima das bibliotecas. Os padrões de projeto nos dizem como resolver alguns problemas, e é tarefa nossa adaptar esses projetos para adequá-los a nosso aplicativo (FREEMAN, 2007).

Cada padrão de projeto permite a algum aspecto da estrutura do sistema variar independentemente de outros aspectos, desta forma tornando um sistema mais robusto em relação a um tipo particular de mudança (GAMMA, 2000). Eles podem ser divididos em três propósitos: criacionais, estruturais e comportamentais.

Os padrões criacionais abstraem o processo de instanciação. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados.

Exemplos de padrões criacionais: *Abstract factory, Builder, Factory method, Prototype* e *Singleton*.

Os padrões estruturais têm como foco a maneira de como os objetos são compostos para formar estruturas maiores. Exemplos de padrões estruturais: *Adapter, Bridge, Composite, Decorator, Façade, Flyweight e Proxy*.

Os padrões comportamentais têm como foco determinar responsabilidades entre objetos e também na comunicação desses objetos com as classes e entre eles mesmos. Exemplos de padrões comportamentais: *Chain of responsability, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template method e Visitor*.

2.5 Teste de software

O Teste de Software é um controle de qualidade que tem por objetivo analisar o que o programa faz, como ele funciona, se não tem nada de incomum com o que o foi planejado, ou seja, ele serve para verificar os possíveis defeitos antes do uso, para assim determinar se o software corresponde às especificações esperadas.

Segundo RIOS (2013) a qualidade do software depende do investimento feito no processo de testes. Um software mal testado poderá custar (e muito) caro para a organização.

A qualidade do software nada mais é que a garantia de que as características do software atendam as necessidades dos usuários. Então, a função principal da garantia de qualidade de software é assegurar que os testes sejam planificados corretamente e postos em prática de maneira eficiente para assim eles terem maior probabilidade de alcançar seu objetivo.

A validação é o processo em que se discute se o software é adequado, ou seja, se o produto construído é o correto. O principal objetivo da validação é saber se o software atende às expectativas dos clientes (SOMMERVILLE 2019).

A verificação é o processo em que se discute se o software foi construído corretamente, ou seja, é verificado se o software apresenta falhas e problemas, seja no código, na interface ou nas funcionalidades. O objetivo da verificação é checar se o software atende aos requisitos funcionais e não funcionais (SOMMERVILLE 2019).

O teste é onde o software é executado, para poder testar a interação dele com o cliente, ou seja, ele vai avaliar o comportamento do software. Existem vários tipos de

testes, dentre eles: teste de unidade, teste de integração, teste de violação, teste de sistema, entre outros.

Existem também algumas técnicas de testes, sendo os testes de caixa preta e caixa branca os mais utilizados.

O teste de caixa preta é um teste que não se tem interação direta com o código fonte, ele é realizado através do contato com a interface, e seu principal objetivo é saber se o software atende aos seus requisitos, ou seja, verificar se ele está se comportando da maneira adequada, testando sua funcionalidades para assim saber se o software atende as funções que deve executar.

Diferente do teste de caixa preta, o teste de caixa branca tem interação direta com o código fonte, ele tem por principal objetivo verificar a lógica do software, ou sejam ele trabalha na estrutura interna, verificando e testando o código para saber se o software atente as funcionalidades pré-estabelecidas

3 Estudo de Caso

Neste capítulo, será abordada uma revisão literária dos assuntos essenciais para a elaboração de um aplicativo desenvolvido em linguagem Java e está dividido nos seguintes itens: 3.1 O Sistema, no item 3.2 As fases do sistema, no item 3.3 Padrões de projeto, no item 3.4 Teste de unidade e por fim no item 3.5 Teste de unidade.

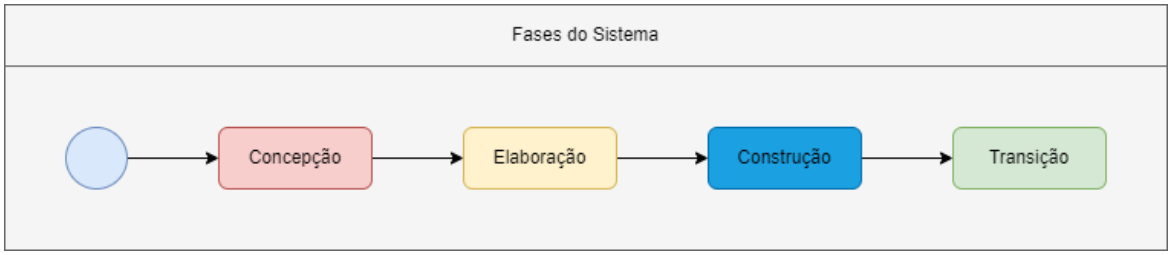
3.1 O Sistema

O software desenvolvido para sistema de eleição tem por objetivo principal facilitar as votações em pequenos negócios e eventos, por meio de uma aplicação onde serão cadastrados os eleitores e os candidatos e os votos computados num banco de dados, substituindo assim os votos em cédulas, permitindo que seja realizada de maneira simples e eficaz, ajudando também o meio ambiente com a não utilização de papel.

O programa funciona da seguinte maneira, um usuário com o perfil administrador cadastra os usuários, que poderão votar logando no sistema, os candidatos e os partidos. Ele também ficará responsável pelo relatório com os votos totais e o resultado da eleição. O usuário poderá votar escolhendo o número do candidato pretendido, o sistema exibirá as informações do candidato tais como a foto, o nome e o partido, onde o usuário poderá confirmar ou não o seu voto, os votos de cada usuário são separados por cargos.

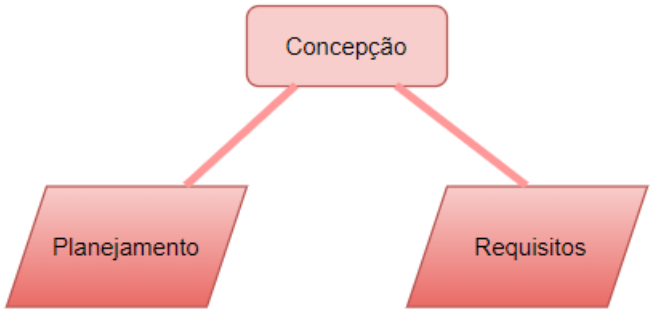
3.2 As fases do sistema

Figura 2 – Fases do sistema



Fonte: Autoral (2022)

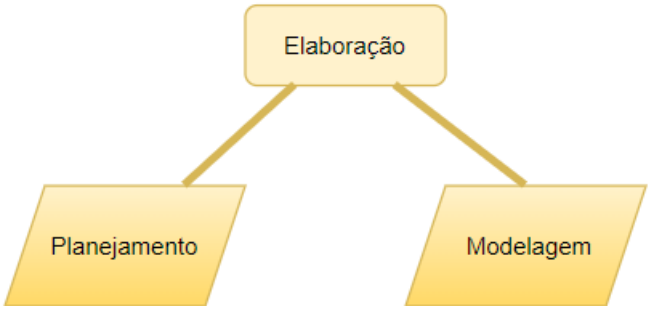
Figura 3 – Fase de concepção



Fonte: Autoral (2022)

A primeira fase é a fase de Concepção, ela tem como objetivo principal a formalização do módulo a ser desenvolvido. Nesta fase foi depatido que o projeto será desenvolvido em linguagem Java através da IDE NetBeans e com o banco de dados My SQL. Identificou-se os atores, requisitos funcionais e não-funcionais além das regras de negócio.

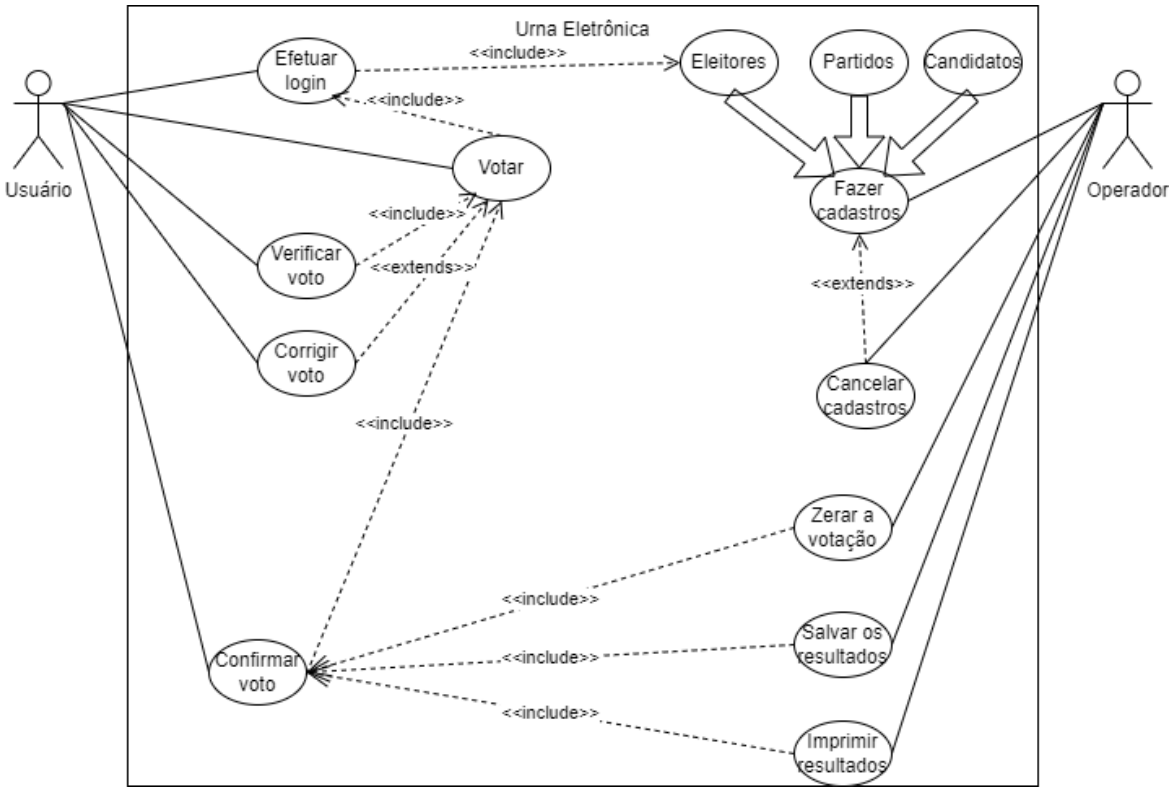
Figura 4 – Fase de elaboração



Fonte: Autoral (2022)

Na fase de Elaboração são discutidos os diagramas UML, eles são divididos em duas categorias, sendo elas: diagramas estruturais e comportamentais. A UML (Unified Modeling Language) – Linguagem de Modelagem Unificada é uma linguagem padrão para modelagem orientada a objetos. No entando, essa linguagem de modelagem não é um método de desenvolvimento.

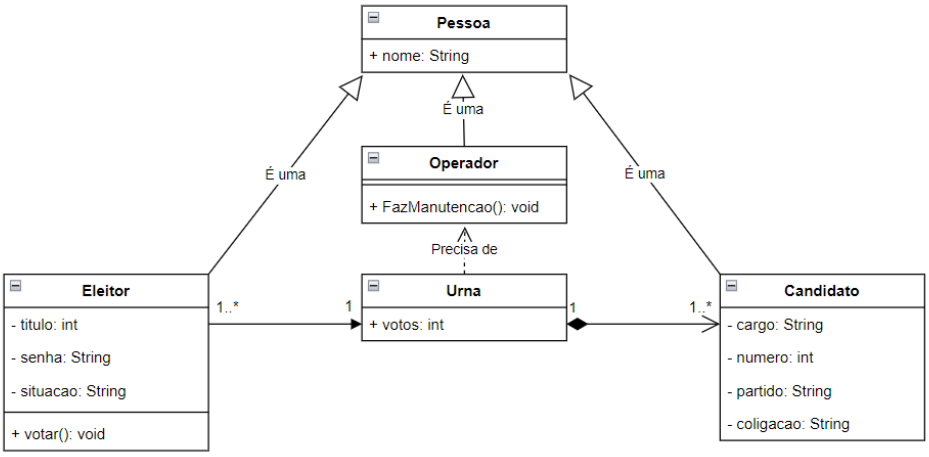
Figura 5 – Diagrama de caso de uso



Fonte: Autoral (2022)

Os diagramas de caso de uso colaboram no levantamento de requisitos funcionais do sistema, descrevendo um conjunto de funcionalidades do sistema e suas interações com elementos externos e entre elas mesmas.

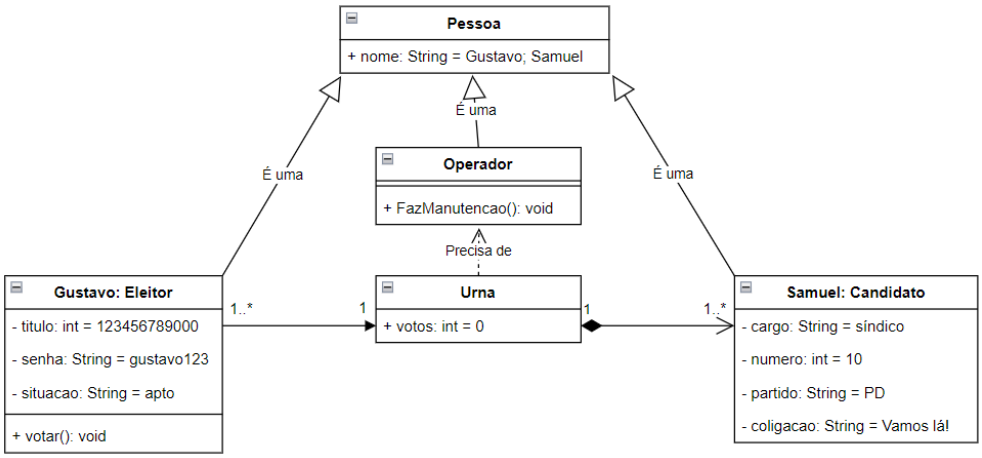
Figura 6 – Diagrama de classes



Fonte: Autoral (2022)

O diagrama de classes em UML tem por objetivo retratar o conjunto de objetos com os mesmos atributos. É utilizado na construção e visualização de Sistemas Orientados a Objetos. Ele descreve como é a estrutura do projeto apresentando classes, atributos, interações e operações.

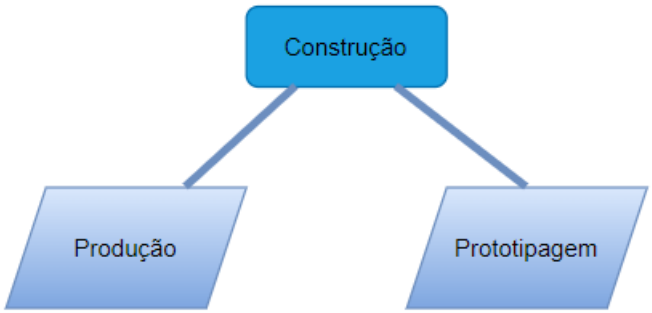
Figura 7 – Diagrama de objetos



Fonte: Autoral (2022)

Os diagramas de objetos em UML tem por objetivo exemplificar como seria o sistema com os dados preenchidos, ou seja, ele auxilia o diagrama de classes.

Figura 8 – Fase de construção

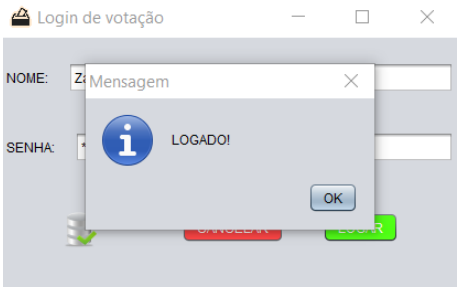


Fonte: Autoral (2022)

A fase de construção é a terceira etapa, o trabalho desta fase foi iniciado baseado na arquitetura produzida pela fase anterior, a fase de Elaboração. As interações e incrementos foram realizados com o objetivo de completar o modelo de requisitos, análise e projeto. Nesta fase foi elaborada uma prototipagem do sistema.

A interface do software começa com uma tela de login, em que o usuário informa seus dados para que possa ser liberado o acesso as principais funções do software.

Figura 9 – Tela de login



Fonte: Autoral (2022)

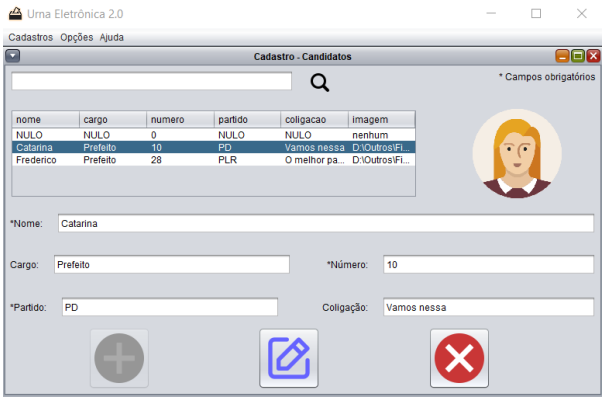
Após login for aceito, o usuário tem acesso ao software em si, onde ele tem um menu com opções para o usuário escolher. A primeira opção de menu é a área de cadastros, onde só é liberado para perfis administradores, essa área cuida do banco de cadastros de eleitores, partidos e candidatos. A segunda opção de menu é onde fica o banco de votos, a urna em si e os relatórios de resultados. A terceira e última opção apresenta informações sobre o software.

Figura 10 – Tela de votação

Fonte: Autoral (2022)

A urna contém o nome do eleitor e a data, os números onde o eleitor poderá clicar e verificar o candidato que por sua vez tem seus dados informados tais como, nome, foto, partido, coligação. Há também três botões que suas funções são respectivamente, confirmar o voto, corrigir o voto e voto em branco.

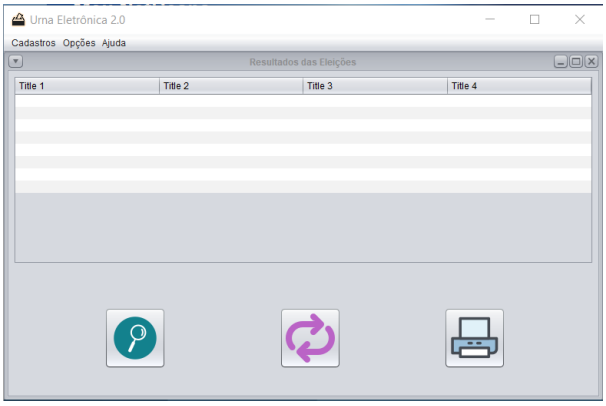
Figura 11 – Tela de cadastros de candidatos



Fonte: Autoral (2022)

A área de cadastros é restrita a perfis administradores. Nessa área poderão ser adicionados novos eleitores, candidatos e partidos, ou também poderão ser removidos e/ou alterados os dados dos mesmos.

Figura 12 – Tela de resultados



Fonte: Autoral (2022)

Os resultados da eleição serão mostrados na tela de resultados, onde esta, assim como a opção de cadastros, é restrita a perfis administradores. Na tela consta uma tabela com resultado dos votos, uma opção de resetar a eleição, deixando todos os candidatos com zero votos e também o software dispõe de uma opção para salvar os resultados em formato PDF.

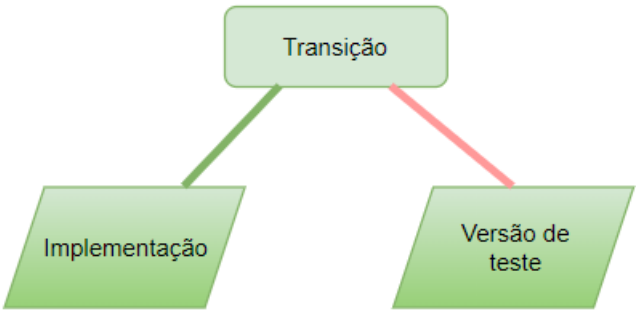
Figura 13 – Trecho do código fonte mostrando a conexão com o banco de dados

```java
package dal; // Pacote dal

import java.sql.*; // Importando a biblioteca sql e o * significa que é tudo relacionado a biblioteca

/**
 *
 * @author CHRISTIAN
 */
public class ModuloConexao {
    // metodo responsavel por estabelecer a conexão com o banco de dados
    public static Connection conector() {
        java.sql.Connection conexao = null;
        // a linha abaixo chama o driver que eu importei
        String driver = "com.mysql.cj.jdbc.Driver";
        // armazenando informações referente ao banco de dados
        String url = "jdbc:mysql://localhost:3306/urna";
        String user = "root";
        String password = "root";
        // Estabelecendo a conexão com o DB
        try {
            Class.forName(driver);
            conexao = DriverManager.getConnection(url, user, password);
            return conexao;
        } catch (Exception e) {
            System.out.println(e);
            return null;
        }
    }
}
```

Fonte: Autoral (2022)

Para conectar o software ao banco de dados primeiro é necessário importar a biblioteca do MySQL connector, e depois criar o código da conexão no NetBeans.

Figura 14 – Fase de transição



Fonte: Autoral (2022)

A útlima etapa é agora a fase de Transição, onde se começa a preocupação com a arquitetura da aplicação e a tecnologia utilizada, diferente da fase de construção onde foi esboçado o problema a ser resolvido.

3.3 Padrões de projeto

O Singleton tem como foco principal que uma classe tenha apenas uma instância no código, isso pode ser utilizado para acessar um recurso compartilhado. Ele tem por vantagem encapsular a instância, evitando assim que outro código possa sobrescrever o valor guardado ali. A classe pode garantir que nenhuma outra instância seja criada (pela interceptação das solicitações para criação de novos objetos), bem como pode fornecer um meio para acessar sua única instância. Ele é utilizado em algumas bibliotecas do Java, como por exemplo a biblioteca java.awt.Desktop, como pode ser visto no trecho do código fonte mostrado na Figura 15.

Figura 15 – Trecho do código fonte mostrando a utilização da classe desktop

```
        }

        // Aqui percorremos a lista e adicionamos na tabela do PDF
        for (Candidatos c : candidatos){
            cell = new PdfPCell(new Paragraph(c.getNome()));
            cell.setHorizontalAlignment(Element.ALIGN_CENTER);
            cel2 = new PdfPCell(new Paragraph(c.getCargo()));
            cel2.setHorizontalAlignment(Element.ALIGN_CENTER);
            cel3 = new PdfPCell(new Paragraph(c.getNumero()+""));
            cel3.setHorizontalAlignment(Element.ALIGN_CENTER);
            cel4 = new PdfPCell(new Paragraph(c.getPartido()));
            cel4.setHorizontalAlignment(Element.ALIGN_CENTER);
            cel5 = new PdfPCell(new Paragraph(c.getColigacao()));
            cel5.setHorizontalAlignment(Element.ALIGN_CENTER);
            cel6 = new PdfPCell(new Paragraph(c.getVotos()+""));
            cel6.setHorizontalAlignment(Element.ALIGN_CENTER);

            table.addCell(cell);
            table.addCell(cel2);
            table.addCell(cel3);
            table.addCell(cel4);
            table.addCell(cel5);
            table.addCell(cel6);
        }

        doc.add(table);
        doc.close();
        Desktop.getDesktop().open(new File(arquivoPDF));

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e);
    }
}
```

Fonte: Autoral (2022)

Figura 16 – Documentação da biblioteca desktop

getDesktop

```
public static Desktop getDesktop()
```

Returns the `Desktop` instance of the current browser context. On some platforms the Desktop API may not be supported; use the `isDesktopSupported()` method to determine if the current desktop is supported.

Returns:
the Desktop instance of the current browser context

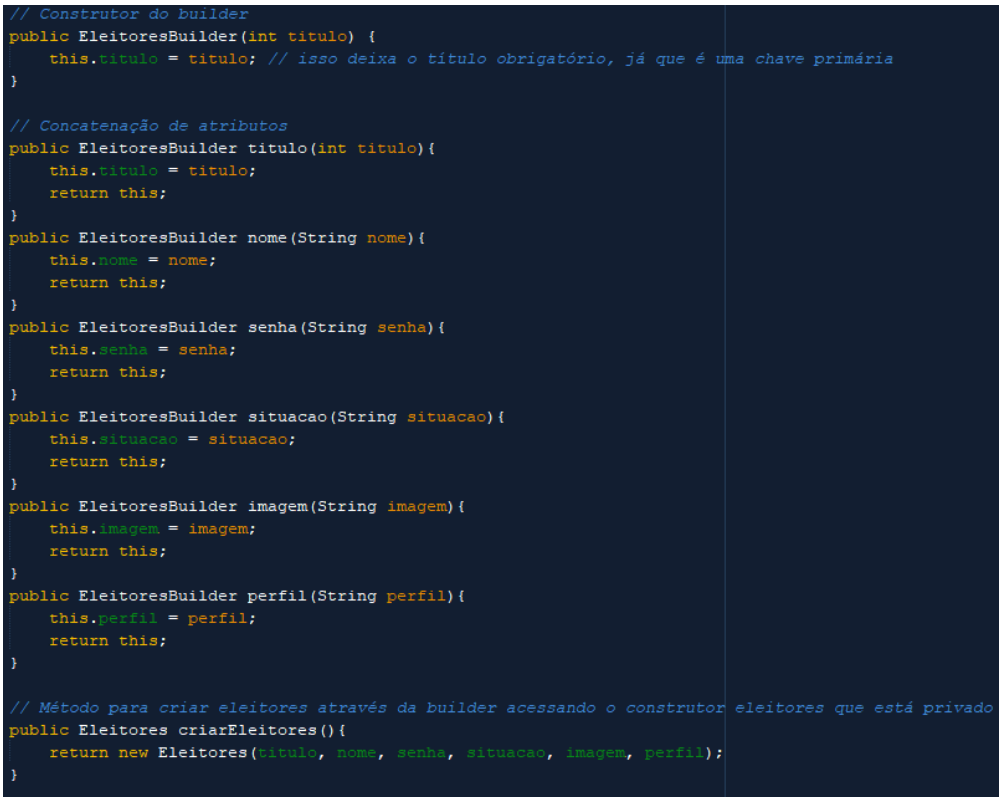
Throws:
`HeadlessException` - if `GraphicsEnvironment.isHeadless()` returns `true`
`UnsupportedOperationException` - if this class is not supported on the current platform

See Also:
`isDesktopSupported()`, `GraphicsEnvironment.isHeadless()`

Fonte: docs.oracle.com (2022)

O Builder é um padrão criacional em que a sua vantagem é que poder criar um objeto passo a passo, isso é bastante útil quando se tem uma classe com inúmeros métodos que podem confundir na hora de instanciar. Os métodos builder normalmente suportam concatenação de atributos, como pode ser observado no trecho do código fonte mostrado na figura 17.

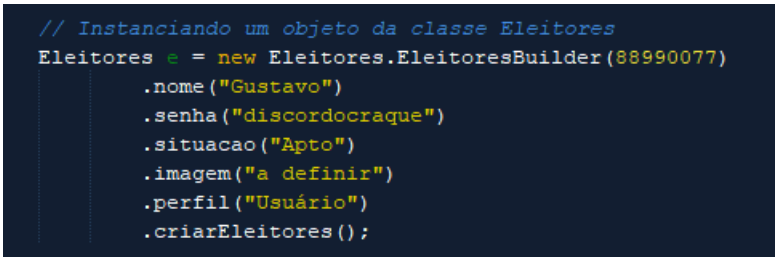
Figura 17 – Trecho do código fonte mostrando a classe builder dentro da classe eleitores



```
// Construtor do builder
public EleitoresBuilder(int titulo) {
    this.titulo = titulo; // isso deixa o título obrigatório, já que é uma chave primária
}

// Concatenação de atributos
public EleitoresBuilder titulo(int titulo){
    this.titulo = titulo;
    return this;
}
public EleitoresBuilder nome(String nome){
    this.nome = nome;
    return this;
}
public EleitoresBuilder senha(String senha){
    this.senha = senha;
    return this;
}
public EleitoresBuilder situacao(String situacao){
    this.situacao = situacao;
    return this;
}
public EleitoresBuilder imagem(String imagem){
    this.imagem = imagem;
    return this;
}
public EleitoresBuilder perfil(String perfil){
    this.perfil = perfil;
    return this;
}

// Método para criar eleitores através da builder acessando o construtor eleitores que está privado
public Eleitores criarEleitores(){
    return new Eleitores(titulo, nome, senha, situacao, imagem, perfil);
}
```

Fonte: Autoral (2022)

Figura 18 – Trecho do código fonte mostrando a instanciação de um objeto criado no builder



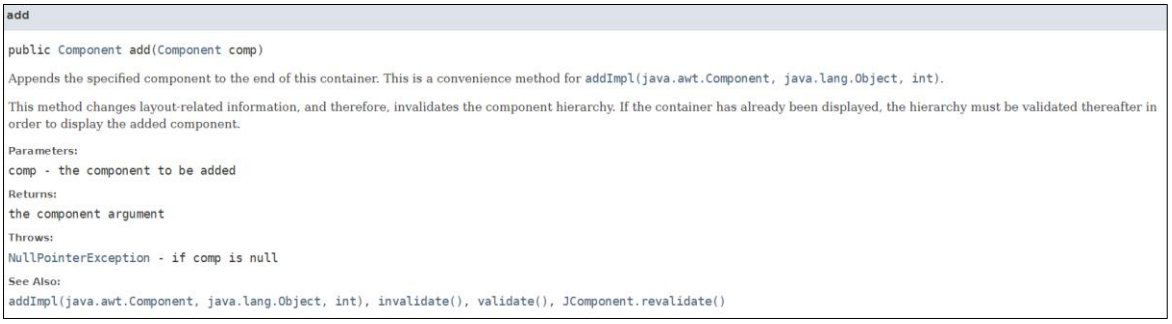
```
// Instanciando um objeto da classe Eleitores
Eleitores e = new Eleitores.EleitoresBuilder(88990077)
        .nome("Gustavo")
        .senha("discordocraque")
        .situacao("Apto")
        .imagem("a definir")
        .perfil("Usuário")
        .criarEleitores();
```

Fonte: Autoral (2022)

O Composite tem por objetivo organizar os objetos em estruturas de árvore, servindo como solução para estruturas complexas, priorizando a composição ao invés da herança. Através desse padrão fica mais fácil usar polimorfismo e recursão. O Composite permite tratar de maneira uniforme objetos individuais e composição de objetos. A chave para o padrão Composite é uma classe abstrata que representa tanto as primitivas como

os seus recipientes. Ele é bastante utilizado em Java, como na biblioteca java.awt.Container.

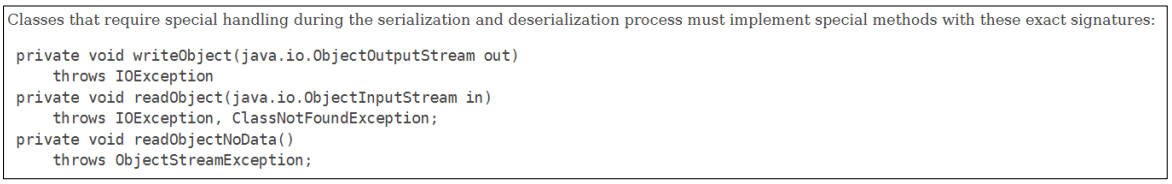
Figura 19 – Documentação da biblioteca container



Fonte: docs.oracle.com (2022)

O memento é um padrão que permite armazenar o estado interno de um objeto em um determinado momento para que seja possível retorná-lo a este estado. Memento evita a exposição de informação que somente um originador deveria administrar, mas que, contudo, deve ser armazenada fora do originador. Ele também garante o encapsulamento e consistência nos backups. Esse padrão é utilizado na biblioteca do Java java.io.Serializable.

Figura 20 – Documentação da biblioteca serializable

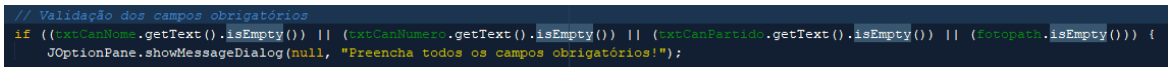


Fonte: docs.oracle.com (2022)

3.4 Teste de unidade

O Teste de unidade é realizado para testar unidades individualmente, tais como funções, métodos, procedimentos. Seu principal objetivo é verificar uma unidade isoladamente afim de determinar se ela está realizando aquilo que foi especificado, assim como mostra na Figura 21.

Figura 21 – Trecho do código fonte que verifica se os campos obrigatórios estão preenchidos



```
// Validação dos campos obrigatórios
if ((txtCanNome.getText().isEmpty()) || (txtCanNumero.getText().isEmpty()) || (txtCanPartido.getText().isEmpty()) || (fotopath.isEmpty())) {
    JOptionPane.showMessageDialog(null, "Preencha todos os campos obrigatórios!");
```

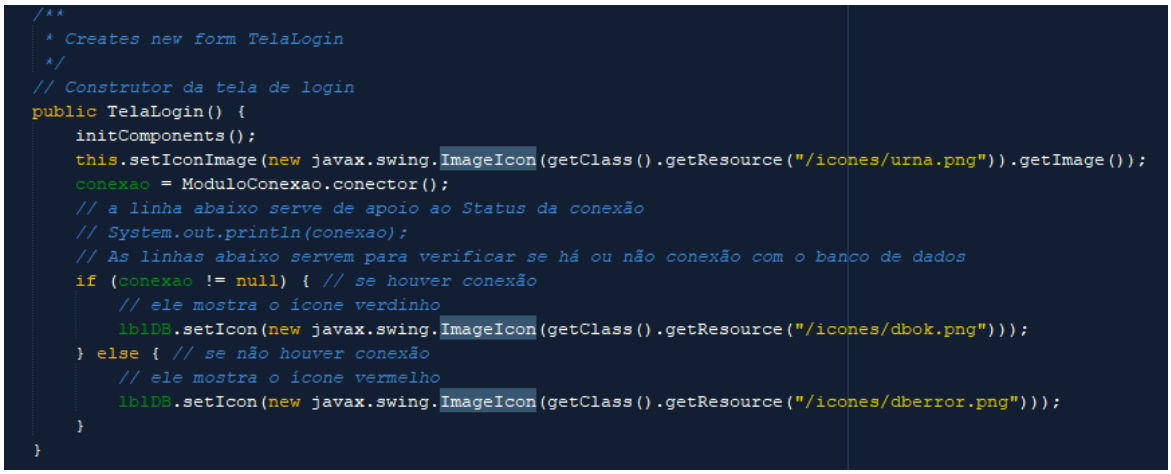
Fonte: Autoral (2022)

Uma boa vantagem na utilização do teste de unidade é detectar precocemente possíveis erros nas funcionalidades do software.

3.5 Teste de integração

O Teste de integração tem foco nas interfaces de comunicação de unidades, afim de garantir que elas funcionem juntas. Seu principal objetivo é verificar a integração entre unidades afim de detectar possíveis falhas.

Figura 22 – Trecho do código fonte que verifica a conexão com o banco de dados



```
/**
 * Creates new form TelaLogin
 */
// Construtor da tela de login
public TelaLogin() {
    initComponents();
    this.setIconImage(new javax.swing.ImageIcon(getClass().getResource("/icones/urna.png")).getImage());
    conexao = ModuloConexao.conector();
    // a linha abaixo serve de apoio ao Status da conexão
    // System.out.println(conexao);
    // As linhas abaixo servem para verificar se há ou não conexão com o banco de dados
    if (conexao != null) { // se houver conexão
        // ele mostra o ícone verdinho
        lblDB.setIcon(new javax.swing.ImageIcon(getClass().getResource("/icones/dbok.png")));
    } else { // se não houver conexão
        // ele mostra o ícone vermelho
        lblDB.setIcon(new javax.swing.ImageIcon(getClass().getResource("/icones/dberror.png")));
    }
}
```

Fonte: Autoral (2022)

Dentre suas vantagens, podemos destacar que o teste de integração é útil para verificar se todas as unidades do aplicativo estão corretamente integradas e se elas funcionam bem juntas. Ele também deceta possíveis conflitos entre essas integrações de unidades.

4 Resultados e discussões

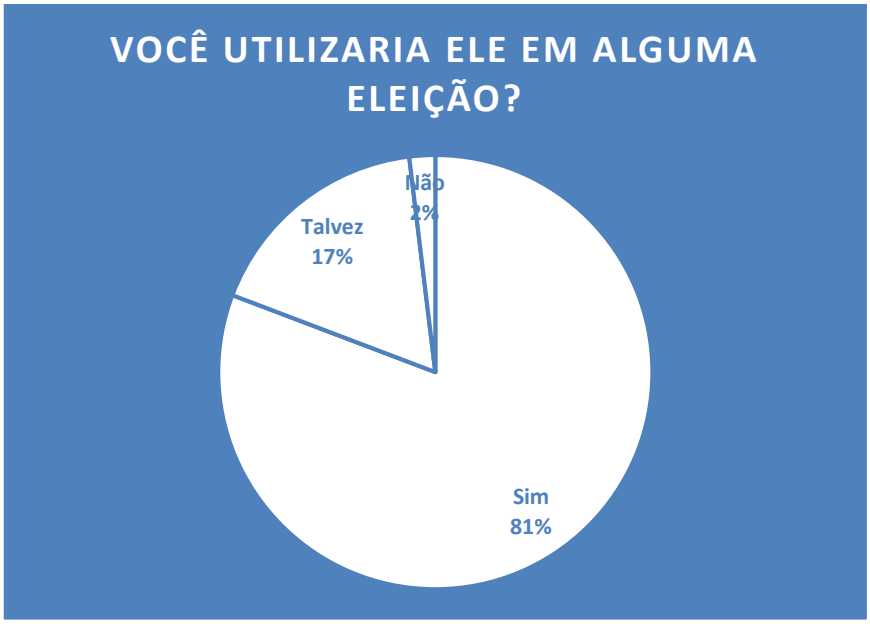
Ao longo do desenvolvimento deste trabalho, foi utilizada a pesquisa de caráter qualitativa, sendo aplicado um questionário de diagnóstico para identificar as maiores dificuldades em manusear a ferramenta computacional. O público alvo do software de votação eletrônica são pequenos negócio e organizações e/ou associações de pessoas, e para frisar o bom funcionamento da aplicação e também para saber se o problema foi resolvido da maneira mais satisfatória possível, foi realizada uma pesquisa com 100 clientes do público alvo afim de obter-se um feedback de como foi o desempenho do software.

Gráfico 1 – Usabilidade do sistema

Fonte: Autoral (2022)

A primeira pergunta foi “qual o nível de dificuldade para usar o software?”, analisando o gráfico nota-se que o software em geral é de fácil manueio pelo usuário por ser bem intuitivo e ter uma boa operabilidade.

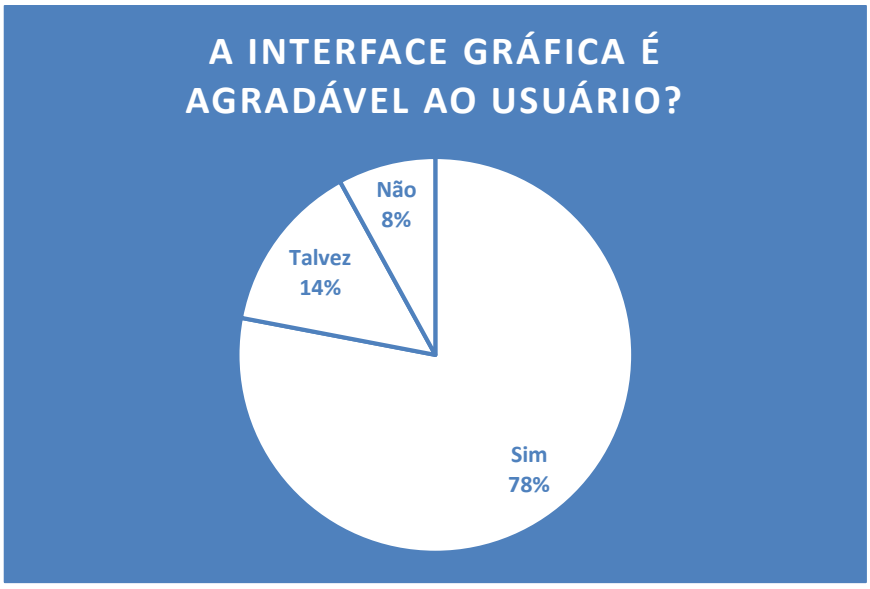
Gráfico 2 – Disponibilidade do sistema



Fonte: Autoral (2022)

A segunda pergunta foi “você utiliaria ele em alguma eleição?”, e pelo resultado mais da metade dos usuários disseram que utilizariam o software para a realização de alguma eleição.

Gráfico 3 – Interface do sistema



Fonte: Autoral (2022)

A terceira pergunta foi “a interface gráfica é agradável ao usuário?”, os resultados da pesquisa mostraram que uma grande maioria teve boas impressões com a interface do software.

Gráfico 4 – Desempenho do sistema

Fonte: Autoral (2022)

A quarta e última pergunta foi “o desempenho do software foi o ideal?”, como mostra o gráfico, os usuários, quase de maneira unânime, tiveram uma boa impressão a respeito do desempenho do software.

5 Considerações finais

Como foi abordado nos resultados, o projeto proposto foi bem aceito pelos usuários, obtendo um bom desempenho, com uma boa interface e principalmente de fácil usabilidade, com isso podemos destacar a qualidade do software, uma vez que ele atendeu as necessidades dos clientes.

A solução do problema se dá devido a automação e a facilidade de manuseio do software, o que o torna mais prático, rápido e eficaz numa eleição de carácter associativo, substituindo os votos em cédulas, economizando assim tempo e papel pois a votação e a contagem, além também da verificação e confirmação do usuário, é tudo feito automaticamente.

Entendemos assim como funciona um projeto construído em uma linguagem orientada a objetos, e também os passos que se deve seguir para fazer um software de qualidade como diagramas UML, padrões de projeto e os testes de software.

Conclui-se então que a tecnologia está cada vez mais se tornando parte do cotidiano das pessoas com o intuito de facilitar as tarefas que antes eram feitas a mão e, além do tempo decorrido na realização da tarefa ser maior, o desempenho do software é realizado de maneira mais eficaz.

5.1 Trabalhos futuros

A ideia que foi pensada ao longo do trabalho, que podem ser implementados futuramente são: Adaptação do sistema para dispositivos móveis, podendo o eleitor votar através do celular; A construção do pensamento lógico que é uma das principais habilidades adquiridas pelo pensamento computacional. Afinal, ela permite que os profissionais identifiquem padrões e definem determinadas ações a partir deles. Abstração e algoritmos são duas bases do pensamento computacional, e elas abrem espaço para uma habilidade importante que é a autonomia.

Atualmente, a exigência por profissionais completos é alta, além de concluírem uma graduação, os jovens precisam saber uma segunda língua e compreender teorias e ferramentas que excedem o escopo de suas profissões. Nesse cenário, ter um pensamento computacional e saber aplicar a teoria na prática será um grande diferencial no mercado de trabalho. Elas permitem que os profissionais deixem de ser apenas consumidoras das tecnologias criadas e produzidas e passem também a ser produtores de recursos digitais consequentemente, isso os prepara para o mundo em que diferentes tecnologias são inseridas diariamente.

Referências Bibliográficas

AQUINO, Rodrigo S. Prudente de. O processo unificado integrado ao desenvolvimento Web. Disponível em http://www.devmedia.com.br/artigo-engenharia-de-software-o-processo-unificado-integrado-ao-desenvolvimento-web/8032. Publicado em Março de 2008. Acessado em Julho de 2016.

Tribunal Superior Eleitoral. Eleições Seguras: saiba como surgiu a urna eletrônica e por que ela está em constante processo de evolução. 2016. Disponível em: url. Acesso em: 20/11/2022.

BARTIE, Alexandre. Garantia de qualidade de software. 1ª edição. Editora GEN LTC. 14 outubro. 2002.

BOOCH, Grady; RUMBAUCH, James; JACOBSON, Ivair. UML – Guia do usuário. Rio de Janeiro: Campus, 2006. ISBN: 978-85-352-1784-1

GAMMA, Erich; HELM Richard; JOHNSON, Ralph e VLISSIDES, John. Padrões de Projeto – Soluções Reutilizáveis de Software Orientados a Objetos. 1ª edição. Editora Bookman. 1 janeiro. 2000.

GUEDES, Gilleanes T. A. UML 2 – Uma abordagem prática. Editora Novatec. 2011.

MARTIN, Robert C. Código limpo – Habilidade práticas do Agile Software. 1ª edição. Editora Alta Books. 8 setembro. 2009.

SCOTT, Kendall. O Processo Unificado Explicado. 1ª edição. Editora Bookman. 1 janeiro. 2003.

SOMMERVILLE, Ian. Engenharia de software. 10ª edição. Editora Pearson Universidades. 22 abril. 2019.

GILBERT, Stephen; McCARTY, Bill. Object-oriented deseing in Java. 1ª edição. Editora Waite Group Pr. 1 janeiro. 1998.

DEITEL, Paul; DEITEL, Harvey. Java: Como programar. 10ª edição. Editora Pearson Universidades. 24 junho. 2016.

FREEMAN, Eric. Use a cabeça! Padrões de projeto. 2ª edição. Editora Alta Books. 22 novembro. 2007.

RIOS, Emerson; MOREIRA, Trayahú. Teste de software. 3ª edição. Editora Alta Books. 4 julho. 2013.

DELAMARCO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. Introdução ao Teste de Software. 1ª edição. Editora Elsevier. 26 junho. 2007.

GERRA, Eduardo. Design Patterns com Java: Projeto orientado a objetos guiado por padrões. 1ª edição. Editora Casa do Código. 16 abril. 2014.

RURA, L.; ISSAC, B.; HALDAR, M. Vulnerability studies of e2e voting systems. In: Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering. [S.I.]: Springer, 2015.

WOHLIN, C.; AURUM, A. Towards a decision-making structure for selecting a research desing in empirical software engineering. Empirical Software Engineering, Springer, v.20, n. 6, p. 1427-1455, 2015.