



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Jefferson Lima Dantas

**Projeto e Desenvolvimento de Software para Cálculo da Parte Ativa de
Transformadores de Corrente**

Blumenau
2023

Jefferson Lima Dantas

**Projeto e Desenvolvimento de Software para Cálculo da Parte Ativa de
Transformadores de Corrente**

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.

Orientador: Prof. Mauri Ferrandin, Dr.

Blumenau
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Dantas, Jefferson Lima

Projeto e Desenvolvimento de Software para Cálculo da
Parte Ativa de Transformadores de Corrente / Jefferson
Lima Dantas ; orientador, Mauri Ferrandin, 2023.

59 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Campus Blumenau,
Graduação em Engenharia de Controle e Automação, Blumenau,
2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2.
Desenvolvimento de software de engenharia. 3. Engenharia
de requisitos. 4. Transformadores de corrente. I.
Ferrandin, Mauri. II. Universidade Federal de Santa
Catarina. Graduação em Engenharia de Controle e Automação.
III. Título.

Jefferson Lima Dantas

Projeto e Desenvolvimento de Software para Cálculo da Parte Ativa de Transformadores de Corrente

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Engenheiro de Controle e Automação” e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 14 de Julho de 2023.

Banca Examinadora:

Prof. Mauri Ferrandin, Dr.
Universidade Federal de Santa Catarina

Prof. Carlos Roberto Moratelli, Dr.
Universidade Federal de Santa Catarina

Prof. Guilherme Brasil Pintarelli, Dr.
Universidade Federal de Santa Catarina

Eu dedico este trabalho a todos os educadores com quem
tive a oportunidade de aprender e crescer. A cada
professor e professora que cruzou meu caminho, desde os
primeiros anos escolares até a minha trajetória acadêmica,
agradeço por seu compromisso e paixão pela educação.

AGRADECIMENTOS

Agradeço principalmente a minha família pelo apoio durante essa jornada acadêmica. Agradeço, também, a todos os professores e alunos que abdicaram de seu tempo e energia para fornecer suporte adicional, tirar minhas dúvidas, orientar projetos e oferecer uma perspectiva valiosa.

"A dúvida incomoda muito e, não obstante, dela nasce quase toda transformação científica ou de valores." (KARNAL, 2019)

RESUMO

Esta pesquisa visa identificar e analisar os requisitos de negócio necessários para o desenvolvimento de um software que automatiza o processo de cálculo da parte ativa de transformadores de corrente. O estudo envolveu revisões detalhadas da literatura técnica, realização de entrevistas, consultas com especialistas da área e visitas técnicas às fábricas, além de se basear inteiramente no programa já existente que foi desenvolvido no decorrer dos últimos 35 anos de uma empresa já consolidada no mercado de transformadores para instrumentos. Os requisitos foram identificados e categorizados em termos de arquitetura, funcionalidades e segurança necessárias para o software, como modularidade, escalabilidade e flexibilidade, permitindo, assim, a fácil ampliação a diferentes requisitos e futuros projetos. Os resultados englobam a arquitetura do modelo de cálculo, descritos com diagramas de classe, exemplos do código fonte implementado e uma breve discussão sobre a interface de usuário desenvolvida, que foi projetada de forma intuitiva e amigável para permitir aos usuários uma experiência fluida e eficiente na entrada de dados, no acompanhamento das etapas do cálculo e na análise dos resultados obtidos. Por fim, a implementação desse software trará benefícios significativos para a empresa, facilitando a implementação de diversos desenvolvimentos, sendo uma das formas de absorver a cultura referente às melhorias contínuas, impulsionando a inovação e fortalecendo a competitividade da organização no mercado.

Palavras-chave: Arquitetura de Software; Requisitos de Negócio; Engenharia de Software.

ABSTRACT

This research aims to identify and analyze the business requirements necessary for the development of software that automates the calculation process of the active part of current transformers. The study involved detailed reviews of technical literature, conducting interviews, consulting with experts in the field, and technical visits to factories. It is entirely based on the existing program that has been developed over the past 35 years by a well-established company in the instrument transformers market. The requirements were identified and categorized in terms of architecture, functionalities, and security necessary for the software, such as modularity, scalability, and flexibility, allowing for easy expansion to different requirements and future projects. The results involve the architecture of the calculation model, described with class diagrams, examples of implemented source code, and a brief discussion about the developed user interface, which was designed intuitively and user-friendly to provide users with a smooth and efficient experience in data entry, monitoring calculation steps, and analyzing the results obtained. Finally, the implementation of this software will bring significant benefits to the company, facilitating the implementation of various developments and being one of the ways to assimilate the culture of continuous improvement, driving innovation, and strengthening the organization's competitiveness in the market.

Keywords: Software Architecture; Business Requirements; Software Engineering.

LISTA DE FIGURAS

| | |
|---|----|
| Figura 1 – Exemplo de circuito de um TC. | 16 |
| Figura 2 – Exemplo de arquitetura que utiliza DAO e DTO. | 30 |
| Figura 3 – Modelo MVVM. | 32 |
| Figura 4 – Cálculo da Indução Nominal do Núcleo. | 37 |
| Figura 5 – Cálculo da Tensão de Circuito Aberto. | 38 |
| Figura 6 – Transformador de Alta e Média Tensão. | 40 |
| Figura 7 – Arquitetura do Modelo - Alta Tensão. | 44 |
| Figura 8 – Arquitetura do Modelo - Média Tensão. | 45 |
| Figura 9 – Arquitetura do Modelo - Parte Ativa. | 46 |
| Figura 10 – Arquitetura do Modelo - Primário. | 47 |
| Figura 11 – Arquitetura do Modelo - Secundário de Medição. | 48 |
| Figura 12 – Arquitetura do Modelo - Derivações. | 49 |
| Figura 13 – Arquitetura do Modelo - Secundário de Proteção. | 50 |
| Figura 14 – Arquitetura do Modelo - Secundários. | 51 |
| Figura 15 – Arquitetura do Modelo - Completa. | 52 |

SUMÁRIO

| | | |
|----------|---|-----------|
| 1 | INTRODUÇÃO | 12 |
| 1.1 | OBJETIVOS | 13 |
| 1.2 | DESCRIÇÃO DOS CAPÍTULOS | 13 |
| 2 | FUNDAMENTAÇÃO TEÓRICA | 15 |
| 2.1 | TRANSFORMADOR DE CORRENTE | 15 |
| 2.1.1 | Requisitos de projeto de Transformadores de Corrente | 16 |
| 2.1.2 | Normas | 17 |
| 2.1.3 | Parâmetros comuns para Transformadores de Corrente de Me- dição e Proteção | 18 |
| 2.1.4 | Parâmetros específicos para Transformadores de Corrente de Medição | 19 |
| 2.1.5 | Parâmetros específicos para Transformadores de Corrente de Proteção | 20 |
| 2.2 | REQUISITOS DE NEGÓCIO | 21 |
| 2.2.1 | Etapas Para Levantamento de Requisitos | 22 |
| 2.2.2 | Tipos de Requisitos | 23 |
| 2.2.3 | Técnicas de Elicitação de Requisitos | 24 |
| 2.3 | ENGENHARIA NO DESENVOLVIMENTO DE SOFTWARE | 26 |
| 2.4 | ARQUITETURA DE SOFTWARE | 26 |
| 2.4.1 | Princípios | 26 |
| 2.4.2 | UML | 27 |
| 2.4.3 | DAO e DTO | 29 |
| 2.4.4 | Padrões Criacionais | 29 |
| 2.4.5 | Modelo MVVM | 30 |
| 2.4.6 | SOLID | 32 |
| 3 | DESENVOLVIMENTO | 34 |
| 3.1 | REQUISITOS DE NEGÓCIO | 34 |
| 3.1.1 | Análise Fit/Gap | 34 |
| 3.1.2 | Detalhamento dos Requisitos | 35 |
| 3.2 | DESENVOLVIMENTO DOS REQUISITOS | 37 |
| 3.2.1 | Cálculo da Parte Ativa | 37 |
| 3.2.2 | Segurança da Informação | 38 |
| 3.2.3 | Controle de Versões | 39 |
| 3.2.4 | Software Expansível e de Fácil Manutenção | 39 |
| 3.2.5 | Exibir Apenas Informações Necessárias na Interface | 41 |
| 4 | RESULTADOS | 43 |
| 4.1 | ARQUITETURA DO MODELO DE CÁLCULO DO SOFTWARE | 43 |

| | | |
|-----|--|-----------|
| 4.2 | EXEMPLOS DOS CÓDIGOS IMPLEMENTADOS | 53 |
| 4.3 | INTERFACE DO USUÁRIO | 54 |
| 5 | CONCLUSÃO | 55 |
| | REFERÊNCIAS | 56 |

1 INTRODUÇÃO

A pesquisa e desenvolvimento que são detalhadas neste documento foram executadas em um empresa tradicional fabricante de transformadores para instrumentos e conjuntos de medição, sendo essa organização presente no fornecimento às empresas, principalmente, de energia elétrica no Brasil e em diversos países.

De acordo com Brom (2021), os transformadores para instrumentos são utilizados para reduzir altas tensões e correntes, como nas redes de transmissão ou distribuição, além de outras aplicações de média tensão, para níveis que podem ser medidos pelos dispositivos instalados. A precisão e a confiabilidade desses equipamentos são muito importantes para fins de faturamento, proteção e monitoramento da estabilidade e qualidade dos sistemas em que estão inseridos.

Dentre os transformadores para instrumentos que a organização em questão comercializa, tem-se os Transformadores de Corrente (TC), que são projetados, desenvolvidos e fabricados na planta. Como já introduzido anteriormente, eles são utilizados para reduzir os níveis de correntes elétricas de forma segura e precisa, permitindo que as informações sejam lidas pelos instrumentos de medição e/ou relés, sem que eles estejam conectados diretamente ao circuito de alta-tensão. Como citado por Paic (2005), transformadores para instrumentos são produtos feitos de acordo com as demandas do cliente. Geralmente, uma das demandas mais importantes que devem ser cumpridas é o curto prazo de entrega.

Para atender às especificações de cada cliente e o prazo de entrega, a empresa utiliza uma planilha excel, com diversas macros, que automatiza o processo de cálculo da parte ativa do TC. A saída dessa aplicação contempla itens como a carta de fabricação, que será utilizada na fábrica para a produção do transformador, gráficos com características referentes aos secundários, sendo a parte do dispositivo responsável por fornecer a corrente proporcional ao campo magnético gerado pelo enrolamento primário, e as listas de materiais que serão utilizadas na fabricação, podendo assim quantificar o preço final do produto.

Diante disso, a organização observou a necessidade de transcrição dessa planilha para dentro de um outro sistema já robusto, pois esse sistema, já contendo a característica de expansibilidade, facilita a implementação de diversos desenvolvimentos no futuro, como, por exemplo, a integração com o sistema de gerenciamento. Além disso, aumenta-se a segurança referente a roubo e a possíveis alterações que podem ser feitas por usuários no caso da planilha excel. Por fim, também contaria com uma equipe de desenvolvedores à disposição para darem suporte, mantendo o devido controle de versões através da ferramenta GIT.

Portanto, iniciou-se um novo projeto de engenharia. Geralmente, grandes empresas seguem um processo estruturado para a iniciação de projetos. Esse processo pode variar, dependendo da empresa e do tipo de projeto. As demandas e inícios de projetos nessa empresa, por exemplo, variam de setor para setor com algumas similiaridades.

A necessidade do projeto foi estabelecida pelos líderes e trata-se da elaboração de um software de engenharia. O autor, sendo o principal desenvolvedor do projeto, foi realocado para planta com o intuito de trabalhar em colaboração com as partes interessadas para definir o escopo do projeto, entender sobre o produto, levantar de forma clara os requisitos de negócio e projetar a arquitetura do software, as funcionalidades e as interfaces de usuário com base nos requisitos levantados.

Com o auxílio das equipes de Engenharia e Pesquisa e Desenvolvimento (P&D), foi executado o plano de ação, arquitetando e codificando o software, realizando testes e integrações e criando documentação. Também foi realizado o monitoramento do projeto para garantir que ele estivesse avançando de acordo com o plano de ação.

1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver um software que automatiza o processo de cálculo de transformadores de corrente. Busca-se estudar e analisar uma tecnologia já existente e trazê-la para um ambiente previamente estruturado, pois, dentro desse ambiente, facilita-se desenvolver diversas melhorias que possam, principalmente, diminuir custos durante o processo de projeto e produção do produto.

Especicamente, a fim de alcançar o objetivo principal mencionado, são necessários os seguintes objetivos específicos a serem cumpridos:

1. Revisão dos conceitos necessários para o trabalho;
2. Estudo e compreensão do modelo e sistema atual utilizado na empresa;
3. Levantar os requisitos de negócio;
4. Arquitetar e desenvolver a interface do software baseando-se nos requisitos e padrões de arquitetura;
5. Desenvolvimento do código na linguagem C# baseando-se em boas práticas de programação.

1.2 DESCRIÇÃO DOS CAPÍTULOS

Este documento contempla o detalhamento de como é organizado e feito um dos projetos de engenharia de uma grande empresa. O resultado será focado na arquitetura definida para o software, que é uma das etapas mais importantes no ciclo de desenvolvimento, pois, para esta etapa, é necessário um amplo conhecimento técnico do produto e das necessidades dos usuários. Portanto o documento será dividido em 4 capítulos além da Introdução:

- Fundamentação Teórica, onde é discutido sobre o produto, requisitos de negócio, metodologias e padrões de arquitetura para a estruturação do software;

- Desenvolvimento, onde contempla as decisões e o desenvolver daquilo que foi estudado, como os requisitos de negócios levantados e a elaboração deles;
- Resultados, onde é retratado os resultados daquilo que foi desenvolvido no projeto;
- Conclusão, contemplando uma discussão final das etapas e objetivos alcançados.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentado a fundamentação teórica que embasa o presente documento, fornecendo uma base conceitual para a compreensão do tema em estudo e realizando uma revisão abrangente da literatura.

2.1 TRANSFORMADOR DE CORRENTE

Um transformador de corrente, também conhecido como TC, é um dispositivo elétrico utilizado para medir a corrente elétrica em um circuito. Ele é amplamente utilizado em sistemas de medição e proteção, especialmente em aplicações de alta tensão e corrente, onde tensões e/ou correntes são muito altas para que as correntes sejam medidas diretamente pelos instrumentos de medição convencionais.

O funcionamento de um transformador de corrente baseia-se nos princípios do eletromagnetismo. Ele consiste em um enrolamento primário, que é conectado em série com o circuito a ser medido, e um enrolamento secundário, que é conectado aos instrumentos de medição ou proteção. O número de espiras no enrolamento primário é projetado para permitir que uma corrente muito alta circule por ele, enquanto o número de espiras no enrolamento secundário é projetado para produzir uma corrente proporcionalmente menor que seja adequada para medição (BHIM SINGH; CHANDRA; KAMAL, 2013).

O secundário de um transformador de corrente é composto pelo núcleo e pelos enrolamentos. O núcleo é feito de material ferromagnético, como ferro-silício, e tem a função de canalizar e direcionar o fluxo magnético gerado pela corrente no enrolamento primário. Os enrolamentos secundários, por sua vez, são constituídos por espiras de fio condutor e são responsáveis por receber a corrente induzida no núcleo do transformador e transmiti-la aos instrumentos de medição ou proteção (HÉROUX, P., 2011).

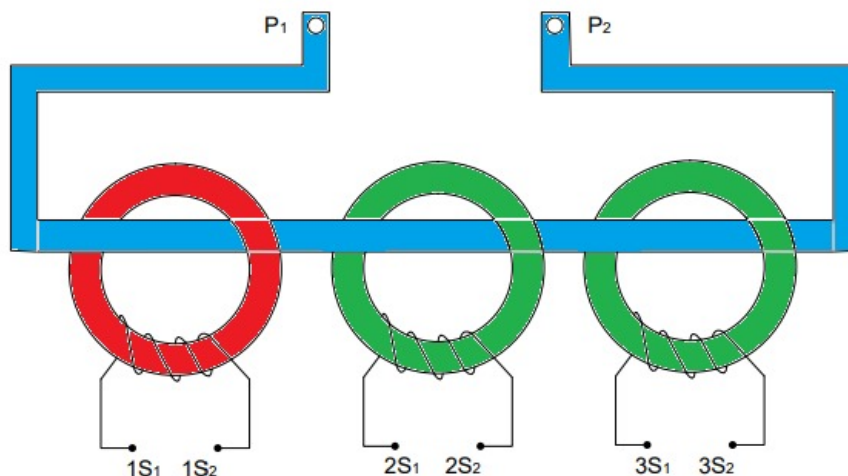
Os transformadores de corrente são projetados para operar com alta precisão e confiabilidade. Eles são calibrados para fornecer uma relação de transformação precisa e compensar quaisquer erros ou distorções introduzidos pelo dispositivo. Além disso, eles são projetados para ter alta imunidade a interferências eletromagnéticas e outras influências externas que possam afetar as medições (HÉROUX, Pierre, 2011).

A parte ativa de um transformador de corrente é composta pelo primário, pelos núcleos e seus enrolamentos (secundários) e pelo isolamento. Sendo na parte ativa justamente onde acontece as transformações citadas acima.

Existem dois tipos de secundários com diferentes finalidades, na Figura 1 por exemplo, tem-se a ilustração de um circuito de um TC, o primário destacado em azul, um secundário de medição em vermelho e dois secundários de proteção em verde. Nesse exemplo, o TC contém mais de um secundário, o que é comum em aplicações de alta tensão.

A função do secundário de medição é fornecer uma corrente proporcional à corrente

Figura 1 – Exemplo de circuito de um TC.



Fonte: Do autor.

medida no enrolamento primário para os instrumentos de medição. Esse secundário é utilizado em aplicações em que a corrente precisa ser monitorada e medida com precisão para fins de monitoramento, controle, faturamento ou análise. Os instrumentos de medição, como amperímetros e sistemas de aquisição de dados, são conectados ao secundário de medição para obter as leituras corretas da corrente elétrica no circuito.

Por outro lado, o secundário de proteção tem a função de fornecer uma corrente proporcional à corrente de falha ou corrente de curto-circuito no enrolamento primário para acionar dispositivos de proteção, como relés de sobrecorrente ou disjuntores. Esse secundário é projetado para operar em situações de falha, onde é necessário detectar e interromper rapidamente a corrente excessiva ou anormal para proteger o equipamento e a segurança do sistema elétrico. Os relés de proteção são conectados ao secundário de proteção para receber o sinal e atuar de acordo com os parâmetros de proteção definidos.

2.1.1 Requisitos de projeto de Transformadores de Corrente

Os requisitos de projeto dos transformadores de corrente englobam uma variedade de aspectos técnicos, normativos e operacionais que devem ser atendidos para garantir seu correto funcionamento. Esses requisitos abrangem desde especificações elétricas, como a relação de transformação, precisão da medição, faixa de corrente nominal e resposta em frequência, até requisitos mecânicos, como isolamento adequado, dissipação de calor e compatibilidade eletromagnética.

Além disso, os transformadores de corrente devem cumprir normas e regulamentos específicos, que variam de acordo com o país e a aplicação em que são utilizados. Essas normas estabelecem critérios de desempenho, segurança e qualidade que os transformadores

devem atender para serem considerados adequados para uso.

2.1.2 Normas

A concepção e fabricação de transformadores de corrente envolvem a consideração de normas e padrões específicos que estabelecem diretrizes para garantir a qualidade, desempenho e segurança desses dispositivos. Serão abordadas algumas das principais normas que regem o projeto de transformadores de corrente, sendo elas:

- IEC 60044-1, Instrument Transformers - Part 1: Current Transformers;
- IEC 60044-6, Instrument Transformers - Part 6: Requirements for Protective Current Transformers for Transient Performance;
- IEC 61869-2, Instrument Transformers - Part 2: Additional Requirements for Current Transformers;
- NBR 6856, Transformador de Corrente com Isolação Sólida para Tensão Máxima Igual ou Inferior a 52 kV.

A norma IEC 60044-1 estabelece os requisitos gerais para transformadores de corrente utilizados em medição e proteção de sistemas de energia elétrica. A norma define critérios para a relação de transformação, precisão da medição, faixa de corrente nominal, resposta em frequência e características mecânicas dos transformadores de corrente. Além disso, ela aborda aspectos relacionados à temperatura ambiente, isolamento elétrico, resistência a curto-circuito e requisitos de segurança. A IEC 60044-1 é essencial para garantir que os transformadores de corrente atendam aos padrões de desempenho e confiabilidade necessários para sua aplicação em sistemas elétricos de diferentes setores, como distribuição de energia, indústria e infraestrutura. Sua adoção proporciona uma base para o projeto, fabricação e utilização desses dispositivos (IEC, 1996).

A norma IEC 60044-6 (1992) estabelece os requisitos técnicos específicos para esses transformadores, garantindo que atendam aos critérios de desempenho e confiabilidade necessários para a detecção e isolamento de falhas elétricas. A norma abrange aspectos como a relação de transformação, classe de precisão, características de saturação, resposta em frequência e tolerâncias dos transformadores de corrente para proteção. Além disso, também aborda requisitos de isolamento, temperatura ambiente, segurança elétrica e ensaios necessários para garantir a conformidade desses transformadores com os padrões estabelecidos. Ela fornece orientações valiosas para o projeto e fabricação de transformadores de corrente de proteção, permitindo que sejam dimensionados e configurados adequadamente para a detecção de correntes anormais e a operação confiável dos sistemas de proteção. A adoção dessa norma auxilia engenheiros, projetistas e fabricantes na seleção e aplicação correta desses dispositivos, assegurando que atendam aos requisitos de segurança e desempenho exigidos. Ao seguir as diretrizes estabelecidas pela IEC 60044-6

(1992), é possível garantir a proteção efetiva dos sistemas elétricos, minimizando os riscos de falhas e danos aos equipamentos (IEC, 1992).

A norma IEC 61869-2 (2012) aborda aspectos como a precisão da medição, resposta em frequência, isolamento elétrica, sobrecarga, características mecânicas e tolerâncias dos transformadores de corrente. Ao seguir as diretrizes da IEC 61869-2 (2012), fabricantes e projetistas podem assegurar que os transformadores de corrente atendam aos critérios de desempenho necessários para uma medição precisa e confiável. A norma fornece uma base sólida para o projeto e fabricação desses dispositivos, considerando fatores essenciais como a exatidão das medições, a linearidade da resposta em frequência e a compatibilidade eletromagnética. A adoção dessa norma contribui para a melhoria da qualidade dos sistemas elétricos, promovendo a confiabilidade das medições e o adequado funcionamento dos sistemas de controle e proteção (IEC, 2012).

A NBR 6856 abrange diversos aspectos do projeto e fabricação dos transformadores de corrente, incluindo características elétricas, mecânicas e de segurança. Ela estabelece requisitos para a relação de transformação, classe de precisão, faixa de corrente nominal, resposta em frequência, entre outros parâmetros elétricos essenciais. Além disso, a norma também trata de aspectos mecânicos, como a construção, o isolamento, a resistência mecânica e a proteção contra curto-circuito. A adoção da NBR 6856 proporciona uma referência para o projeto e fabricação de transformadores de corrente no Brasil. Ela assegura que esses dispositivos atendam aos critérios técnicos específicos do país, levando em consideração as características dos sistemas elétricos e as normas regulatórias brasileiras. A conformidade com a NBR 6856 é essencial para garantir a qualidade dos transformadores de corrente utilizados nas redes elétricas brasileiras, promovendo a segurança operacional e a confiabilidade dos sistemas de energia. (ABNT, 1992)

2.1.3 Parâmetros comuns para Transformadores de Corrente de Medição e Proteção

Para ambos transformadores de corrente de medição e proteção, existem parâmetros comuns que precisam ser considerados durante o projeto e seleção desses equipamentos. Esses parâmetros incluem as correntes nominais do primário e do secundário. As correntes nominais do primário estão associadas às correntes nominais dos circuitos onde os transformadores serão aplicados. Por sua vez, as correntes nominais do secundário são definidas pelos instrumentos e equipamentos que serão conectados aos transformadores (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017a).

É importante considerar as sobrecargas permitidas no sistema elétrico. No geral, a estrutura física do primário, como barras ou cabos, é a mesma para os secundários de medição e proteção. Portanto, as correntes nominais primárias são as mesmas para ambas as aplicações.

A corrente secundária é definida pelos instrumentos e equipamentos aos quais

eles serão conectados. Geralmente, são adotados valores de 1, 2 ou 5 A como correntes nominais. Historicamente, no Brasil, o valor mais comum tem sido de 5 A (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017a).

Conhecendo as necessidades da corrente nominal do primário e da corrente nominal do secundário, pode-se definir a relação de transformação. Algumas normas apresentam diversos valores para que os usuários se enquadrem com o intuito de viabilizar praticamente todas as necessidades práticas dos sistemas de proteção e medição. A tabela 1 é um exemplo de relações nominais.

Tabela 1 – Exemplo de relações nominais (NBR 6856).

| Corrente | Esquema | Corrente primária nominal | Relação Nominal | Derivações secundárias |
|---------------------------------|---------|---------------------------|-----------------|------------------------|
| 1000/1500 /4000/6000 – 5A | | 500 | 100:1 | S2 – S3 |
| | | 1000 | 200:1 | S1 – S2 |
| | | 1500 | 300:1 | S1 – S3 |
| | | 2000 | 400:1 | S4 – S5 |
| | | 2500 | 500:1 | S3 – S4 |
| | | 3000 | 600:1 | S2 – S4 |
| | | 4000 | 800:1 | S1 – S4 |
| | | 4500 | 900:1 | S3 – S5 |
| | | 5000 | 1000:1 | S2 – S5 |
| | | 6000 | 1200:1 | S1 – S5 |

Fonte: (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017a)

2.1.4 Parâmetros específicos para Transformadores de Corrente de Medição

Os transformadores de corrente de medição desempenham um papel fundamental no faturamento de energia elétrica. Esses transformadores são projetados levando em consideração parâmetros específicos que garantem a precisão e a confiabilidade das medições realizadas. Neste capítulo, serão discutidos os parâmetros específicos aplicáveis aos transformadores de corrente de medição.

Para os transformadores de corrente de medição, é essencial garantir níveis de precisão nas medições realizadas, especialmente para fins de faturamento de energia. As classes de precisão definem os limites de erro aceitáveis para esses transformadores. A norma IEC 61869-2 estabelece classes de precisão variando de 0,1% a 5%. Já as normas brasileiras, NBR 6856 de 1992 e 2015, consideram transformadores de corrente de medição com precisões de até 3% (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017a).

Além das classes de precisão mencionadas, a norma IEC 61869-2 também introduz duas classes especiais, denominadas 0,2S e 0,5S, para os transformadores de corrente tipo "S" (Special). A norma NBR 6856 define as classes 0,3S e 0,6S para transformadores de corrente de medição. Essas classes especiais indicam que esses transformadores foram

projetados para atender a requisitos mais rigorosos de precisão, garantindo resultados ainda mais confiáveis em medições de energia elétrica.

Algumas normas estabelecem o conceito de paralelogramos de erros para os transformadores de corrente de medição. Esses paralelogramos representam os limites de tolerância para os erros de amplitude e fase em diferentes condições de carga. A norma NBR 6856 apresenta dois paralelogramos de erro para cada classe de precisão (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017a).

2.1.5 Parâmetros específicos para Transformadores de Corrente de Proteção

Os transformadores de corrente de proteção são projetados para a detecção e proteção contra faltas (desvio das condições operacionais padrão) e sobrecorrentes em sistemas elétricos. Diferentemente dos transformadores de corrente de medição, os transformadores de corrente de proteção possuem parâmetros específicos que garantem o desempenho adequado durante condições de falta e sobrecorrente. Nesta seção serão discutidos os parâmetros específicos aplicáveis aos transformadores de corrente de proteção (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017b).

O Fator Limite de Exatidão (FLE) é utilizado para definir o limite de corrente multiplicado pela corrente primária nominal do transformador de corrente, considerando características senoidais. Esse fator é conhecido popularmente como fator de sobrecorrente. O valor do fator FLE está relacionado à tensão necessária no secundário do transformador para reproduzir a corrente primária sem distorções, mantendo a precisão das medições.

Ao contrário dos transformadores de corrente de medição, os transformadores de corrente de proteção não requerem a mesma precisão nas medições. No entanto, é crucial que esses transformadores não saturem durante condições de falta, o que exige a definição da tensão de "joelho" para lidar com transientes causados por curto-circuitos. A NBR 6856 de 2015 caracteriza os TC's como P, PR, PXR e PXS, sendo:

- Classe P: Indica transformadores de corrente que não possuem controle do fluxo residual e têm sua especificação de tensão de saturação definida para a corrente de curto-circuito simétrico, que é determinada pelo fator de limite de precisão especificado;
- Classe PR: Caracteriza transformadores de corrente com controle do fluxo residual (inferior a 10%) e especificação de tensão de saturação definida para a corrente de curto-circuito simétrico associada ao fator de limite de precisão. Esses transformadores possuem uma pequena folga de ar para reduzir o fluxo residual;
- Classe PX: Caracteriza transformadores de corrente de baixa reatância cujas características são definidas pelo usuário, incluindo corrente nominal do primário e secundário, número de espiras, tensão de saturação e corrente de excitação

associadas, fator de limite de precisão, resistência do secundário e resistência da carga adotada. A norma sugere a definição da tensão de saturação considerando apenas a componente AC da corrente de falta;

- Classe PXR: A especificação é semelhante à classe PX, porém, considera-se que existe controle do fluxo residual por meio da introdução de uma pequena folga de ar no núcleo, mantendo o fluxo residual abaixo de 10%.

Para as classes P e PR, a norma IEC 61869-2 define erros de 5% e 10% como os erros compostos para a corrente no valor limite, múltiplos da corrente nominal para a qual o transformador de corrente foi construído. Esses múltiplos estão associados ao fator de limite de exatidão (FLE). A carga padrão é definida com fator de potência de 0,8, exceto quando a carga for inferior a 5 VA, caso em que um fator de potência unitário pode ser adotado. Para os transformadores do tipo PR, o limite de fluxo residual é de 10% (PEREIRA; PEREIRA JUNIOR; LOURENÇO, 2017b).

A norma também descreve os transformadores de corrente de proteção com especificações para faltas transitórias. Os tipos TPX e TPY têm sua definição de erro considerando as componentes AC e DC da corrente de falta, enquanto o tipo TPZ tem o erro referido apenas à componente AC, pois tende a filtrar a componente DC da corrente de falta devido à sua baixa impedância de magnetização, que inclui uma folga de ar maior em comparação com o tipo TPY.

2.2 REQUISITOS DE NEGÓCIO

Os requisitos de negócio desempenham um papel fundamental no processo de desenvolvimento de software, pois são eles que definem as necessidades e expectativas do cliente. Eles são a base para a criação de um produto ou sistema que atenda aos objetivos e requisitos específicos de uma organização. Portanto, entender e capturar de forma precisa os requisitos de negócio é essencial para o sucesso de qualquer projeto. Eles referem-se às funcionalidades, características e restrições que o sistema ou produto deve possuir para atender às necessidades do cliente e do negócio em geral. Eles podem abranger desde os requisitos funcionais, que descrevem as ações e comportamentos que o sistema deve executar, até os requisitos não funcionais, que descrevem aspectos como desempenho, segurança, usabilidade e escalabilidade do sistema (WIEGERS; BEATTY, 2013).

Um dos maiores desafios no gerenciamento de requisitos de negócio é garantir que eles sejam corretamente identificados, compreendidos e documentados. Isso requer uma comunicação clara e efetiva entre as partes interessadas, incluindo o cliente, usuários finais, analistas de negócio e desenvolvedores. A colaboração e o trabalho em equipe são fundamentais para garantir que todas as necessidades sejam identificadas e traduzidas em requisitos claros e coerentes.

Além disso, a gestão dos requisitos de negócio é um processo contínuo ao longo

do ciclo de vida do projeto. À medida que o projeto avança, é comum que os requisitos evoluam e sejam refinados à medida que se aprende mais sobre o domínio do negócio e as necessidades dos usuários. Portanto, é fundamental estabelecer mecanismos para rastrear, analisar e gerenciar as mudanças nos requisitos, garantindo que o produto final atenda aos objetivos iniciais (ROBERTSON, S.; ROBERTSON, J., 2012).

2.2.1 Etapas Para Levantamento de Requisitos

O processo de gerenciamento de requisitos de negócio envolve várias etapas cruciais para garantir que as necessidades e expectativas do cliente sejam adequadamente identificadas, compreendidas e documentadas. Essas etapas abrangem desde a elicitação inicial dos requisitos até a validação final, passando pela análise detalhada e especificação clara dos requisitos. As quatro etapas são:

- Elicitação;
- Análise;
- Especificação;
- Validação.

A elicitação de requisitos é o processo de identificar, coletar e compreender as necessidades do cliente e do negócio. Envolve a interação com as partes interessadas relevantes, como clientes, usuários finais e especialistas de domínio, para obter informações sobre o sistema ou produto a ser desenvolvido. Técnicas comuns de elicitação incluem entrevistas, *workshops*, questionários e observação direta. O objetivo é obter uma compreensão clara dos requisitos e seus contextos, identificando funcionalidades, restrições e necessidades específicas (WIEGERS; BEATTY, 2013).

A análise de requisitos é a etapa em que os requisitos coletados são analisados e refinados para garantir sua viabilidade técnica, clareza e consistência. Envolve a identificação de requisitos duplicados, conflitantes ou incompletos, bem como a priorização dos requisitos com base em sua importância e impacto no projeto. Durante essa etapa, podem ser utilizadas técnicas como modelagem de negócios, diagramas de caso de uso, diagramas de classe e prototipação para ajudar a visualizar e entender melhor os requisitos (ROBERTSON, S.; ROBERTSON, J., 2012).

A especificação de requisitos é o processo de documentar os requisitos de negócio de forma clara e completa. Envolve a criação de documentos, como a especificação de requisitos, que descrevem de maneira detalhada as funcionalidades, características e restrições do sistema ou produto. A especificação de requisitos deve ser precisa, consistente e compreensível para todas as partes interessadas envolvidas no projeto. Ela serve como base para o desenvolvimento, teste e validação do sistema (WIEGERS; BEATTY, 2013).

A validação de requisitos é a etapa final, na qual os requisitos documentados são revisados, verificados e validados para garantir que atendam aos objetivos do negócio

e às expectativas do cliente. Isso envolve a revisão cuidadosa dos requisitos em busca de erros, inconsistências ou omissões, bem como a verificação de sua rastreabilidade e alinhamento com os objetivos do projeto. A validação também pode incluir a realização de revisões formais, prototipação, testes de aceitação e validação por parte dos usuários finais (ROBERTSON, S.; ROBERTSON, J., 2012).

Essas etapas são interdependentes e devem ser realizadas de forma iterativa ao longo do ciclo de vida do projeto. A colaboração entre todas as partes interessadas é fundamental para garantir que os requisitos de negócio sejam adequadamente identificados, compreendidos e validados, resultando em um produto ou sistema que atenda às expectativas e necessidades do cliente.

2.2.2 Tipos de Requisitos

No processo de gerenciamento de requisitos, é importante distinguir diferentes tipos de requisitos que devem ser considerados ao desenvolver um produto ou sistema de software. Cada tipo de requisito aborda uma área específica e contribui para a definição completa e precisa do que o produto deve realizar. Vamos explorar os principais tipos de requisitos:

- Requisitos do Produto;
- Requisitos de Projeto e de Processo;
- Requisitos Funcionais;
- Requisitos Não Funcionais ou da Qualidade;
- Requisitos de Processo ou Restrições.

Os requisitos do produto descrevem as funcionalidades e características que o produto final deve ter para atender às necessidades do cliente. Eles definem o escopo do produto e geralmente são expressos em termos de ações que o sistema deve executar ou resultados que deve produzir. Os requisitos do produto são orientados para o usuário e descrevem o que o sistema deve fazer para satisfazer as necessidades e expectativas dos usuários finais (WIEGERS; BEATTY, 2013).

Os requisitos de projeto e de processo são aqueles que especificam as restrições e diretrizes a serem seguidas durante o desenvolvimento do produto. Eles podem incluir requisitos de design, restrições de hardware ou software, padrões de codificação, restrições de desempenho, restrições de segurança e outras considerações técnicas que influenciam a implementação do sistema. Esses requisitos são essenciais para garantir que o produto seja desenvolvido de acordo com as melhores práticas e atenda aos requisitos de qualidade e eficiência (LEFFINGWELL; WIDRIG, 2003).

Os requisitos funcionais descrevem as ações específicas que o sistema deve ser capaz de executar. Eles definem as funcionalidades, os comportamentos e as interações que o sistema deve suportar. Esses requisitos são expressos em termos de entradas, saídas e

comportamentos desejados do sistema. Os requisitos funcionais descrevem o que o sistema deve fazer em termos de processamento de dados, cálculos, interações com o usuário e integração com outros sistemas (ROBERTSON, S.; ROBERTSON, J., 2012).

Os requisitos não funcionais, também conhecidos como requisitos de qualidade, descrevem as características e atributos do sistema que não estão diretamente relacionados às funcionalidades específicas, mas que são essenciais para a qualidade, desempenho e usabilidade do produto. Eles incluem requisitos relacionados à segurança, desempenho, usabilidade, confiabilidade, escalabilidade, manutenibilidade, disponibilidade, entre outros. Esses requisitos estabelecem os critérios pelos quais o produto será avaliado em termos de sua eficácia e capacidade de atender às necessidades do usuário (GOTTESDIENER; GORMAN, 2012).

Os requisitos de processo ou restrições referem-se às necessidades e restrições relacionadas à forma como o projeto é conduzido e ao processo utilizado para desenvolver o produto. Eles podem incluir requisitos relacionados à metodologia de desenvolvimento, ferramentas de desenvolvimento, controle de versão, documentação, colaboração entre equipes, entre outros. Esses requisitos garantem que o projeto seja executado de maneira eficiente e eficaz, seguindo as melhores práticas de gerenciamento de projetos e garantindo a qualidade do resultado final (WIEGERS; BEATTY, 2013).

Ao identificar e documentar esses diferentes tipos de requisitos, as equipes de desenvolvimento podem garantir uma compreensão clara das necessidades e expectativas do cliente, bem como orientar todo o processo de desenvolvimento, desde a concepção até a implementação e entrega do produto final.

2.2.3 Técnicas de Elicitação de Requisitos

A elicitação de requisitos permite a identificação e compreensão das necessidades dos usuários e *stakeholders*. Existem várias técnicas de elicitação que auxiliam nesse processo, possibilitando a coleta de informações relevantes para a definição dos requisitos do sistema. A seguir, discutiremos algumas das principais técnicas de elicitação de requisitos:

- Entrevista;
- Reunião;
- *Brainstorming*;
- Observação;
- Questionário;
- *Workshop*.

A entrevista é uma técnica amplamente utilizada, na qual um analista de requisitos realiza uma série de perguntas aos usuários, especialistas ou outras partes interessadas para obter informações sobre o sistema. Ela permite uma interação direta e aprofundada,

possibilitando a exploração de necessidades, desejos e restrições dos *stakeholders* (POHL; RUPP, 2010).

As reuniões são encontros formais ou informais entre a equipe de desenvolvimento e os *stakeholders*. Elas permitem a troca de informações, esclarecimento de dúvidas e discussões em grupo sobre os requisitos do sistema. As reuniões podem envolver a participação de vários *stakeholders* e são uma oportunidade para identificar requisitos, alinhar expectativas e obter consenso (WIEGERS; BEATTY, 2013).

O *brainstorming* é uma técnica que visa gerar ideias criativas e inovadoras. Envolve a reunião de um grupo de pessoas, onde todos são encorajados a contribuir com ideias livremente, sem críticas ou julgamentos. O objetivo é estimular o pensamento colaborativo e identificar requisitos que possam não ter sido considerados inicialmente. O *brainstorming* pode ser especialmente útil para estimular a criatividade e explorar diferentes perspectivas (O'CONNOR; LAPLANTE, 2012).

A observação envolve a observação direta dos usuários ou *stakeholders* em seu ambiente de trabalho, permitindo ao analista de requisitos entender como eles realizam suas tarefas e identificar possíveis requisitos. Essa técnica fornece informações valiosas sobre o contexto de uso, desafios enfrentados e necessidades reais dos usuários. A observação pode ser realizada de forma participativa, na qual o analista acompanha o usuário durante suas atividades, ou de forma não participativa, onde o analista observa de forma discreta (ROBERTSON, S.; ROBERTSON, J., 2012).

O questionário é uma técnica que envolve a criação de um conjunto de perguntas estruturadas que são enviadas aos *stakeholders* para coletar informações sobre suas necessidades e expectativas em relação ao sistema. Essa técnica é útil quando é necessário coletar dados de um grande número de pessoas ou quando a interação direta não é possível. No entanto, é importante projetar o questionário de maneira clara e objetiva para garantir que as respostas sejam relevantes (SOMMERVILLE, 2016).

O *workshop* é uma técnica que envolve a realização de sessões colaborativas e interativas com uma variedade de *stakeholders*. Durante o *workshop*, diferentes técnicas podem ser utilizadas, como atividades em grupo, exercícios práticos e discussões. O objetivo é estimular a participação ativa dos *stakeholders*, compartilhar conhecimentos e ideias, e obter um entendimento comum dos requisitos do sistema. Os *workshops* podem ser conduzidos de forma presencial ou virtual, dependendo da disponibilidade e localização dos participantes (WIEGERS; BEATTY, 2013).

O grupo focal é uma técnica que envolve a reunião de um grupo de usuários ou *stakeholders* para discutir e explorar temas específicos relacionados aos requisitos do sistema. Durante a sessão, o facilitador estimula a discussão e busca insights valiosos dos participantes. O grupo focal é útil para obter diferentes perspectivas e aprofundar a compreensão das necessidades dos usuários. É importante garantir a diversidade dos participantes para obter uma visão abrangente dos requisitos (GOTTESDIENER; GORMAN, 2012).

Essas técnicas de elicitação de requisitos fornecem abordagens diferentes para coletar informações valiosas sobre as necessidades e expectativas dos usuários e *stakeholders*. É importante selecionar as técnicas adequadas com base no contexto do projeto, nos recursos disponíveis e nas características dos *stakeholders* envolvidos.

2.3 ENGENHARIA NO DESENVOLVIMENTO DE SOFTWARE

Uma equipe de desenvolvimento de software é responsável por criar soluções computacionais para resolver problemas complexos no campo da engenharia. Nesse contexto, o papel do engenheiro é de trazer conhecimento técnico especializado e experiência na aplicação de princípios de engenharia para o desenvolvimento de software.

Um dos principais papéis do engenheiro em uma equipe de desenvolvimento de software de cálculo/engenharia é a análise de requisitos. O engenheiro tem a tarefa de entender as necessidades dos usuários e traduzi-las em requisitos técnicos claros e precisos. Isso envolve a compreensão profunda do problema de engenharia em questão, a identificação das funcionalidades necessárias e a definição dos critérios de desempenho que o software deve atender.

Um dos papéis do engenheiro, também, é conduzir testes e depuração do software. Ele desenvolve casos de teste para verificar se o software está funcionando corretamente e atendendo aos requisitos estabelecidos. Um outro aspecto fundamental do engenheiro é a colaboração efetiva. Ele deve interagir com outros membros da equipe, como analistas de requisitos, designers de interface, desenvolvedores e testadores, a fim de garantir a integração adequada de todas as partes do projeto. A comunicação clara e a capacidade de trabalhar em equipe são habilidades essenciais para o sucesso do projeto.

2.4 ARQUITETURA DE SOFTWARE

A arquitetura de software fornece a estrutura e a organização necessárias para suportar os requisitos do sistema. Ela define a maneira como os componentes do software se relacionam entre si, como as informações fluem e como as funcionalidades são implementadas. A arquitetura pode ser definida como a estrutura fundamental do sistema, composta por componentes, suas interações e os princípios que guiam sua organização. Ela fornece uma visão abstrata do sistema, descrevendo os principais elementos e suas relações, e serve como um guia para o projeto, desenvolvimento e evolução do software (BASS; CLEMENTS; KAZMAN, 2012).

2.4.1 Princípios

Os princípios de arquitetura de software são diretrizes que orientam o design e a estrutura de sistemas de software. Eles fornecem uma base sólida para a criação

de arquiteturas que são modulares, flexíveis, escaláveis e fáceis de manter. A seguir, apresentamos alguns dos principais princípios de arquitetura de software:

- **Modularidade:** O princípio da modularidade busca dividir o sistema em módulos independentes e coesos. Cada módulo deve ser responsável por uma funcionalidade específica e interagir com outros módulos de forma clara e bem definida. A modularidade permite o reuso de componentes, facilita a manutenção e promove a escalabilidade do sistema (BASS; CLEMENTS; KAZMAN, 2012);
- **Reusabilidade:** O princípio da reusabilidade incentiva a criação de componentes e serviços que possam ser utilizados em diferentes partes do sistema ou em outros sistemas. Componentes reutilizáveis reduzem a redundância de código, melhoram a consistência e a qualidade do software, além de economizarem tempo e esforço no desenvolvimento (MARTIN, 2003);
- **Escalabilidade:** O princípio da escalabilidade visa projetar o sistema de forma que possa lidar com o aumento da carga de trabalho e do volume de dados sem comprometer seu desempenho. Uma arquitetura escalável permite a adição de recursos adicionais, como servidores ou instâncias de aplicativos, para atender às demandas crescentes do sistema (BASS; CLEMENTS; KAZMAN, 2012);
- **Flexibilidade:** O princípio da flexibilidade busca criar um sistema capaz de se adaptar a mudanças e evoluções futuras. Isso inclui a capacidade de incorporar novos requisitos de negócios, acomodar mudanças tecnológicas e permitir a extensibilidade do sistema. Uma arquitetura flexível facilita a manutenção e a evolução contínua do software;
- **Coesão e Baixo Acoplamento:** Os princípios da coesão e baixo acoplamento visam garantir que os componentes do sistema sejam altamente coesos, ou seja, cada componente deve ter uma única responsabilidade bem definida. Além disso, os componentes devem ter um baixo acoplamento, ou seja, devem depender o mínimo possível de outros componentes. Esses princípios promovem a independência dos componentes e tornam o sistema mais flexível e adaptável (ROZANSKI; WOODS, 2011).

2.4.2 UML

A UML (*Unified Modeling Language*) é uma linguagem de modelagem visual utilizada para representar e visualizar diferentes aspectos de sistemas de software e processos relacionados. A UML fornece um conjunto de notações gráficas padronizadas e amplamente adotadas, permitindo aos desenvolvedores e analistas descrever, documentar, projetar e comunicar efetivamente a estrutura, o comportamento e as interações de um sistema. Os diagramas podem ser usados em várias fases do ciclo de vida de desenvolvimento de software,

desde a análise de requisitos até o projeto detalhado e a implementação (RUMBAUGH; JACOBSON; BOOCH, 1999).

A UML é composto por vários tipos de diagramas, cada um com uma finalidade específica. Os diagramas mais comuns são:

- Diagrama de Caso de Uso: Representa as interações entre atores externos e o sistema em estudo, mostrando os diferentes cenários de uso;
- Diagrama de Classes: Descreve a estrutura estática do sistema, exibindo as classes, seus atributos, métodos e relacionamentos;
- Diagrama de Sequência: Ilustra a interação entre objetos ao longo do tempo, mostrando a ordem das mensagens trocadas entre eles;
- Diagrama de Atividade: Modela o fluxo de controle do sistema, descrevendo as atividades e as decisões tomadas em um processo;
- Diagrama de Componentes: Apresenta as partes do sistema e suas dependências, mostrando como os componentes se relacionam e se comunicam entre si;
- Diagrama de Implantação: Descreve a disposição física dos componentes de um sistema, incluindo hardware, software e conexões de rede.

O presente trabalho utiliza o diagrama de classes para representar a arquitetura do modelo de cálculo do software, já que esse diagrama possui diversas características distintas que auxiliam a visualização e entendimento da estrutura do sistema.

O diagrama de classes, como já mencionado anteriormente, evidencia as classes que compõem o sistema, destacando, principalmente, seus atributos e métodos. Além disso, esse diagrama exhibe os relacionamentos entre as classes, incluindo heranças, associações, agregações e composições.

A herança permite que uma classe herde os atributos e métodos de outra classe, possibilitando que essas subclasses reutilizem o código da superclasse, estendendo ou modificando seu comportamento conforme necessário.

A associação representa uma conexão entre duas classes, onde uma classe possui uma instância de outra classe como um de seus atributos. No geral, utiliza-se o termo "associação" quando duas classes são completamente independentes entre si mas eventualmente estão relacionadas. Um exemplo usual de associação seria a relação entre um professor e alunos. Um aluno pode ter vários professores e um professor pode ter vários alunos, eles não dependem entre si para existirem.

A agregação é um tipo de associação, sendo um relacionamento de um para muitos, ou seja, indicando que uma classe contém ou é composta por outras classes. Nesse caso, uma classe é proprietária de outras mas não há dependência, então ambas podem existir mesmo que a relação não se estabeleça.

A composição é um tipo específico de agregação que possui dependência entre as classes. Nesse sentido, indica que as classes que compõe a classe principal só existem

dentro do contexto do todo, ou seja, se não houver a classe principal, as que a compõe também não existirão.

Por fim, o diagrama de classes também permite a abstração, ou seja, a capacidade de mostrar apenas os aspectos relevantes do sistema, ocultando detalhes desnecessários e concentrando-se nos conceitos mais importantes.

2.4.3 DAO e DTO

Os padrões DAO (*Data Access Object*) e DTO (*Data Transfer Object*) no contexto do desenvolvimento de software são amplamente utilizados na construção de sistemas para fornecer uma separação clara e eficiente entre a camada de acesso a dados e a camada de negócios, facilitando a manutenção, a modularidade e a escalabilidade do código.

O padrão DAO é projetado para encapsular a lógica de acesso a dados em uma camada separada da camada de negócios. O DAO atua como uma interface entre o código de negócios e as operações de leitura e gravação de dados no banco de dados. Ele abstrai os detalhes de persistência, fornecendo métodos para recuperar, inserir, atualizar e excluir dados de forma transparente. O uso do padrão DAO promove a modularidade do código, permitindo que as operações de acesso a dados sejam facilmente substituídas ou atualizadas sem afetar a lógica de negócios (BAUER; KING, 2002).

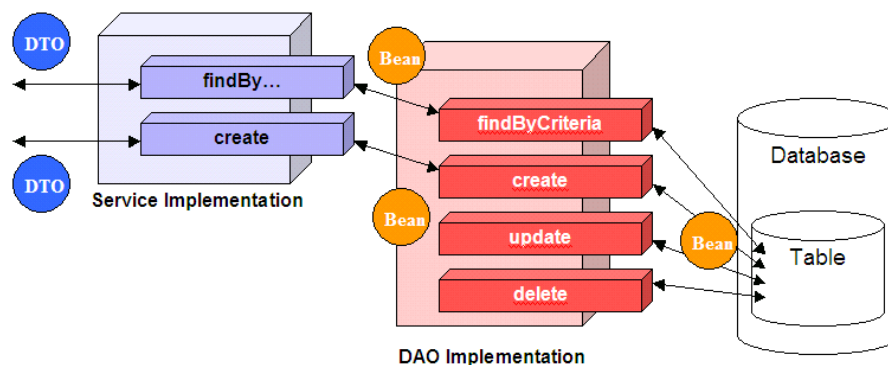
O padrão DTO é utilizado para transferir dados entre diferentes camadas ou componentes de um sistema. Ele encapsula um conjunto de dados relacionados em uma estrutura simples e independente de qualquer lógica de negócios. O DTO é geralmente usado para reduzir o acoplamento entre os diferentes componentes do sistema, evitando a exposição direta dos objetos de domínio. Além disso, o DTO permite otimizar a transferência de dados, selecionando apenas as informações relevantes para cada contexto (ALUR; CRUPI; MALKS, 2003).

Resumindo o que foi dito acima, ao utilizar o padrão DAO, a lógica de acesso a dados é separada da lógica de negócios, permitindo uma melhor organização e manutenção do código. O DAO abstrai as operações de leitura e gravação de dados, oferecendo uma interface consistente e simplificada para a camada de negócios interagir com o banco de dados. Já o padrão DTO permite uma transferência eficiente de dados entre diferentes componentes do sistema. Ele fornece uma estrutura de dados simples e padronizada para evitar a exposição desnecessária de informações sensíveis ou complexas. O uso do DTO ajuda a reduzir o acoplamento entre os componentes, tornando o sistema mais flexível e adaptável a mudanças futuras. Por fim, a Figura 2 deixa claro, por exemplo, a forma como o padrão DTO otimiza a transferência de dados.

2.4.4 Padrões Criacionais

Os padrões de projeto desempenham um papel fundamental na construção de sistemas bem estruturados, modularizados e de fácil manutenção. Esses padrões são soluções

Figura 2 – Exemplo de arquitetura que utiliza DAO e DTO.



Fonte: (MVVM. . . , 2022).

comprovadas para problemas recorrentes no desenvolvimento de software.

Os padrões criacionais fornecem mecanismos para a criação de objetos de forma flexível e encapsulada. Eles se concentram na forma como os objetos são instanciados e configurados, promovendo a reutilização e a flexibilidade do código. Alguns exemplos de padrões criacionais incluem o *Singleton*, o *Factory Method*, o *Abstract Factory* e o *Builder*. Esses padrões permitem que os desenvolvedores criem objetos de maneira consistente e modular, facilitando a manutenção e a evolução do sistema (GAMMA *et al.*, 1994).

Utilizando como exemplo um dos padrões amplamente utilizados no mundo do desenvolvimento, tem-se o *Singleton*. O padrão *Singleton* é um padrão criacional que garante que uma classe tenha apenas uma única instância em todo o sistema. Ele fornece um ponto centralizado de acesso a essa instância, permitindo que os objetos compartilhem um estado comum e evitando a criação excessiva de instâncias. Sua utilização está bem ligada a cenários como gerenciadores de recursos ou caches de dados (ALUR; CRUPI; MALKS, 2003).

2.4.5 Modelo MVVM

O *Model-View-ViewModel* (MVVM) é um padrão de arquitetura de software amplamente utilizado no desenvolvimento de interfaces de usuário. Ele proporciona uma separação clara das responsabilidades, facilitando a manutenção, testabilidade e escalabilidade do código.

O MVVM divide uma aplicação em três componentes principais: *Model* (Modelo), *View* (Visualização) e *ViewModel* (Modelo de Visualização). O *Model* representa os dados e a lógica de negócios da aplicação, a *View* é responsável pela interface de usuário e o *ViewModel* atua como intermediário entre o *Model* e a *View*, fornecendo os dados e comportamentos necessários para a interação com a interface (GOSSMAN; SHURTS, 2012).

Uma das principais vantagens do MVVM é a separação clara das responsabilidades entre os componentes. Isso permite que cada componente seja desenvolvido e mantido de forma independente, facilitando a colaboração entre os membros da equipe e a reutilização de código. A separação de preocupações também melhora a testabilidade, uma vez que os diferentes componentes podem ser testados isoladamente (SMITH, 2016).

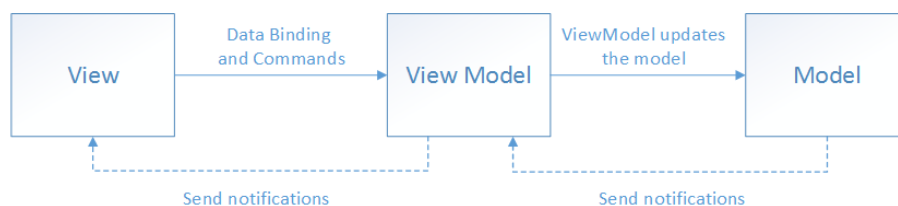
A seguir será detalhado os três componentes desse padrão de arquitetura:

- *Model* (Modelo): O *Model* representa os dados e a lógica de negócios da aplicação. Ele encapsula a funcionalidade central da aplicação, como acesso a banco de dados, manipulação de dados e regras de negócio. O *Model* é independente da interface de usuário e não possui conhecimento sobre a *View* ou o *ViewModel*. Ele é responsável por fornecer os dados e os métodos necessários para a aplicação funcionar corretamente.
- *View* (Visualização): A *View* é responsável pela apresentação da interface de usuário ao usuário final. Ela exibe os elementos visuais, como botões, campos de entrada e listas, e permite a interação com o usuário. A *View* não contém lógica de negócios, apenas reflete o estado dos dados do *ViewModel* e permite que o usuário interaja com esses dados. A *View* também recebe notificações do *ViewModel* quando os dados são atualizados e exibe essas atualizações na interface.
- *ViewModel* (Modelo de Visualização): O *ViewModel* atua como um intermediário entre o *Model* e a *View*. Ele fornece os dados e a lógica de apresentação necessários para a *View*. O *ViewModel* é responsável por obter os dados do *Model*, transformá-los em uma forma adequada para a *View* e fornecê-los para a interface. Ele também contém a lógica de manipulação dos dados e ações acionadas pelo usuário. O *ViewModel* mantém uma representação do estado da *View* e notifica a *View* quando ocorrem alterações nos dados.

O MVVM utiliza amplamente o conceito de *data binding* para estabelecer a ligação entre a *View* e o *ViewModel*. O *data binding* permite que as alterações nos dados do *ViewModel* sejam refletidas automaticamente na interface de usuário, eliminando a necessidade de atualizações manuais. Isso simplifica o desenvolvimento e melhora a responsividade da aplicação (LIPPERT, 2010).

O MVVM também introduz o conceito de comandos, que são objetos responsáveis por representar as ações acionadas pela interação do usuário na interface. Os comandos permitem que a *View* informe ao *ViewModel* sobre as ações realizadas e acionem a lógica correspondente. Isso ajuda a manter a separação entre a *View* e o *ViewModel* e facilita o tratamento de eventos e interações do usuário (GEHTLAND, 2011). A Figura 3 ilustra a arquitetura MVVM, onde é possível visualizar os conceitos de *data binding* e comandos citados anteriormente.

Figura 3 – Modelo MVVM.



Fonte: (MVVM. . . , 2023).

Resumindo, ao adotar o MVVM, os desenvolvedores podem obter uma arquitetura mais modular, flexível e de fácil manutenção. A separação clara de responsabilidades entre *Model*, *View* e *ViewModel* permite uma melhor colaboração entre a equipe de desenvolvimento e facilita a evolução do software ao longo do tempo.

2.4.6 SOLID

Os Princípios SOLID são um conjunto de diretrizes de design de software que ajudam a criar sistemas mais flexíveis, escaláveis e fáceis de manter. Esses princípios são amplamente adotados pela comunidade de desenvolvimento de software e foram estabelecidos por Robert C. Martin, sendo ele um engenheiro de software e uma grande personalidade da área que publicou diversos livros e atua no âmbito de desenvolvimento de software desde 1970. Ao seguir os princípios SOLID, podemos criar sistemas mais flexíveis, fáceis de entender e de manter. Esses princípios nos ajudam a evitar problemas comuns de design, como acoplamento excessivo, classes monolíticas e código difícil de modificar. Eles promovem a coesão, a reusabilidade e a escalabilidade do software.

O SRP (*Single Responsibility Principle*) afirma que uma classe ou módulo deve ter apenas uma única responsabilidade. Isso significa que uma classe deve ter apenas um motivo para mudar. Ao aderir a esse princípio, podemos evitar classes que têm muitas responsabilidades e tornar nosso código mais coeso e de fácil manutenção (MARTIN, 2002).

O OCP (*Open/Closed Principle*) estabelece que as entidades de software devem ser abertas para extensão, mas fechadas para modificação. Isso significa que devemos projetar nosso código de forma que seja possível adicionar novas funcionalidades sem precisar modificar o código existente. O uso de abstrações, interfaces e polimorfismo são técnicas comuns para alcançar esse princípio (MARTIN, 2002).

O LSP (*Liskov Substitution Principle*) afirma que as classes derivadas devem ser substituíveis por suas classes base sem afetar a integridade do programa. Isso significa que, se uma classe base é esperada em um determinado contexto, sua classe derivada também deve ser capaz de ser usada nesse contexto sem causar problemas. O LSP promove o uso adequado de herança e polimorfismo, garantindo a consistência e a segurança do código

(MARTIN, 2002).

O ISP (*Interface Segregation Principle*) preconiza que as interfaces devem ser específicas e direcionadas para as necessidades dos clientes. Em vez de ter interfaces grandes e genéricas, devemos criar interfaces menores e coesas que atendam às necessidades específicas de cada cliente. Isso evita o acoplamento desnecessário e melhora a coesão das classes (MARTIN, 2002).

O DIP (*Dependency Inversion Principle*) propõe que as dependências devem ser baseadas em abstrações e não em implementações concretas. Isso significa que as classes de alto nível não devem depender diretamente de classes de baixo nível, mas sim de abstrações comuns. O uso de interfaces e inversão de controle (IoC) são técnicas frequentemente utilizadas para aplicar esse princípio, promovendo a flexibilidade e a reutilização do código.

3 DESENVOLVIMENTO

Neste capítulo será abordado o desenvolvimento do presente trabalho, fornecendo uma visão detalhada das etapas realizadas e dos métodos empregados. O objetivo é apresentar uma sequência lógica de atividades que permita alcançar os objetivos estabelecidos.

3.1 REQUISITOS DE NEGÓCIO

Para se definir os requisitos de negócio, as etapas descritas na Seção 2.2.1 serviram como base. Durante a etapa de elicitação, foi necessário ter acesso a toda documentação, procedimentos e normas que envolvem a produção do TC da organização, além de realizar, também, diversas visitas técnicas às fábricas. Esse estudo do produto é crucial para levantamento dos requisitos do produto, conforme descrito na Seção 2.2.2.

A partir desse estudo também foi possível definir os requisitos funcionais, já que entender a parte teórica do transformador e o fluxo de informações da empresa é essencial no desenvolvimento dos cálculos e na definição das saídas do software, pois, como se trata de um software de cálculo, o sistema deve automatizar o processo de cálculo dos usuários. Diante disso, surge a necessidade do desenvolvimento do software ser composta por ao menos um engenheiro, pois, conforme descrito na Seção 2.3, a participação do engenheiro é fundamental para a análise de requisitos e, nesse caso, desenvolvimento dos cálculos que contemplam o projeto do TC.

Diante das técnicas descritas na Seção 2.2.3, foram conduzidas diversas reuniões em grupos e separadamente com os usuários finais para levantar os requisitos não funcionais, sendo eles relacionados à usabilidade do software, disposição de informações na tela e desacoplamento daquilo que é geral e específico do projeto, como, por exemplo, a frequência de operação do transformador é única e portanto deve estar numa tela geral do projeto, enquanto que o material do núcleo pode variar de acordo com o tipo de secundário, logo a definição desse parâmetro deve estar disponível em cada tela de projeto de secundário.

Por fim, os requisitos de projeto e processo vieram como padronização da empresa e não precisaram ser levantados, pois, por exemplo, o design do software é baseado nas cores da empresa, as restrições de seguranças e os padrões de codificação do software já são definidas.

3.1.1 Análise Fit/Gap

A análise Fit/Gap, na Tabela 2, envolve a comparação dos requisitos de negócio de uma organização com as funcionalidades fornecidas pelo sistema. O objetivo dessa análise é identificar a diferença entre os requisitos funcionais e os recursos disponíveis no sistema a ser implementado. Essa análise é dividida em três itens:

- *Fit*: O sistema atual atende aos requisitos de negócio. Ou seja, a planilha excel utilizada e desenvolvida pela organização contempla os requisitos.
- *Partial Fit*: O sistema atual atende parcialmente aos requisitos de negócio.
- *Gap*: O sistema atual não atende aos requisitos de negócio.

Tabela 2 – Análise Fit/Gap

| Requisitos de Negócio | Fit | Partial Gap | Gap |
|---|-----|-------------|-----|
| Cálculo da Parte Ativa | X | | |
| Visualização Prévia dos Secundários na Caixa de Blindagem | X | | |
| Gerar Carta de Fabricação | X | | |
| Gerar Gráficos com Características dos Secundários | X | | |
| Segurança da Informação | | X | |
| Controle de Versões | | X | |
| Automatizar a Entrada de Informações | | | X |
| Exibir Apenas Informações Necessárias na Interface | | | X |
| Software Expansível e de Fácil Manutenção | | | X |

Fonte: Do autor

3.1.2 Detalhamento dos Requisitos

Os requisitos de negócios podem incluir funcionalidades, desempenho, segurança, usabilidade, confiabilidade e outros atributos que sejam importantes para o sucesso do projeto. Portanto, a partir do que foi mencionado na análise *Fit/Gap*, cada requisito será descrito brevemente nos itens a seguir:

- **Cálculo da Parte Ativa:** para que seja feito o projeto da parte ativa do TC, são necessários os campos de entrada gerais, que o projetista preenche de acordo com a Ordem de Fabricação (OF), como a norma, frequência, quantidade e tipos de secundários, relação de transformação, entre outros. Também há os campos de entrada referente aos cálculos, preenchidos pelo projetista, que deve atender as especificações do cliente com o menor custo possível. A partir disso necessita-se dos campos de saída onde o calculista possa verificar os resultados.
- **Visualização Prévia dos Secundários na Caixa de Blindagem:** nesse contexto, a caixa de blindagem é o invólucro dos secundários, também conhecido como virola, dos TCs de alta tensão. A partir das características de cada caixa de blindagem, necessita-se fazer um desenho dos núcleos e da caixa. Essa ilustração é utilizada tanto para os projetistas verificarem se os secundários projetados

cabem na caixa, como, também, um guia para a fábrica sobre como deve ser a disposição de cada um dentro da caixa de blindagem.

- Gerar Carta de Fabricação: sendo uma das saídas do software, é através dela que o colaborador na fábrica terá informações necessárias para produzir o transformador. Como, por exemplo, a quantidade de secundários, os tipos de fios, as dimensões dos núcleos, a quantidade de fios por camadas etc. Além de que o projetista complementa a lista de materiais no sistema de gerenciamento através do conteúdo dessa carta.
- Gerar Gráficos com Características dos Secundários: eventualmente, alguns clientes fazem o pedido de determinados gráficos, sendo eles: Curva de Magnetização, Ciclo de Operação, Ciclo de Operação noutra Derivação, Erro e Defasagem pelo Fator de Sobrecorrente (Medição), Erro pelo Fator de Sobrecorrente (Proteção), Limites da Classe de Exatidão. Portanto, necessita-se que o programa gere esses gráficos para que sejam enviados aos clientes que os requisitarem.
- Segurança da Informação: como a ferramenta de automatização de cálculo de TC atual se trata de uma planilha excel, a proteção contra roubo não é robusta, sendo uma das preocupações da empresa.
- Controle de Versões: para a organização das versões, é preciso permitir que os desenvolvedores rastreiem alterações, revertam para versões anteriores, ramifiquem o desenvolvimento do software para trabalhar em recursos diferentes e, em seguida, mesclarem esses ramos novamente, além de também disporem de recursos avançados de colaboração, como revisão de código e gerenciamento de conflitos.
- Automatizar a Entrada de Informações: na planilha excel, os projetistas replicam as informações obtidas da OF, um PDF gerado pelo sistema de gerenciamento. Para automatizar esse processo, o software desenvolvido deverá importar o arquivo com as informações do cliente, ser capaz de lê-lo e preencher determinados campos do projeto de cálculo.
- Exibir Apenas Informações Necessárias na Interface: na interface atual, no projeto de cálculo do transformador, o projetista lida com uma tela contendo diversos tipos de informações, como, por exemplo, caso ele esteja fazendo um projeto de média tensão, ele tem acesso às entradas e saídas da alta tensão, ou então caso ele esteja fazendo o projeto de um secundário de medição, informações do secundário de proteção são exibidas a ele. Logo, há diversos pontos positivos em separar adequadamente tais informações. Sendo um deles, por exemplo, no momento de ensinar um novo usuário, como a empresa está em constante crescimento, há sempre pessoas novas para aprender a trabalhar com aquela

ferramenta. Diante disso, ter uma interface limpa e amigável torna o processo de aprendizagem mais suave.

- Software Expansível e de Fácil Manutenção: sendo um dos requisitos de desenvolvimento e padronização da organização, o software deve ser expansível e de fácil manutenção, significando que caso haja uma alteração no produto, seja um acréscimo, mudança ou melhoria, o software deve ser capaz de contemplar essa alteração com facilidade. Além disso, outros desenvolvedores devem ser capazes de dar manutenção no software mesmo não tendo contato técnico com o produto em questão.

3.2 DESENVOLVIMENTO DOS REQUISITOS

Dentre os requisitos citados na Tabela 2, este documento contemplará apenas o desenvolvimento dos requisitos abaixo:

- Cálculo da Parte Ativa;
- Segurança da Informação;
- Controle de Versões;
- Exibir Apenas Informações Necessárias na Interface;
- Software Expansível e de Fácil Manutenção.

Foi estimado, pela equipe de P&D, cerca de 1048 horas trabalhadas, ou seja, cerca de 5 meses e meio de desenvolvimento voltados aos cinco itens acima.

Os demais requisitos, com exceção da Automatização da Entrada de Informações, são referentes à saída do software, ou seja, serão desenvolvidas após a interface e estrutura de cálculo estiverem funcionando adequadamente.

3.2.1 Cálculo da Parte Ativa

Para o desenvolvimento do cálculo da parte ativa, foi necessário compreender e documentar cada variável de entrada, de cálculo intermediário e de saída. Então, a partir desse documento, poder estruturar, nomear e organizar as informações adequadamente. Um dos exemplos de variáveis de entrada, seria o *Fator Térmico*, que é especificado pelo cliente e remete ao que o TC possa suportar referente às correntes de sobrecargas do sistema sem ser danificado.

Figura 4 – Cálculo da Indução Nominal do Núcleo.

| |
|--|
| BN= <input type="text" value="3787"/> |
| =K*VAT*10000*(1.05)^CORETYPE/HZNOM/R17/H!AN12/H!AN11/SE(CHAPA="E5";0.78;SE(CHAPA="A6";0.75;0.94533)) |

Fonte: Do autor.

Tomando, como exemplo, uma variável de saída da planilha, ela é descrita conforme a Figura 4. Sendo a parte superior a forma como o usuário visualiza a informação e a inferior o cálculo relacionado à variável. Portanto, para documentá-la, a mesma variável foi descrita conforme a equação (1) e o item a seguir:

- *BN: Indução nominal do núcleo.*

Tem a seguinte equação com os termos " if_n " dependendo de situações distintas:

$$BN = \frac{K \cdot VAT \cdot 10^4 \cdot if_1}{Hz \cdot AE \cdot L[H] \cdot E[H] \cdot if_2} \quad (1)$$

- Se o núcleo for de medição $\rightarrow if_1 = 1,05$;
Senão $\rightarrow if_1 = 1$;
- Se a Chapa for A6 $\rightarrow if_2 = 0,75$;
Se a Chapa for E5 $\rightarrow if_2 = 0,78$;
Senão $\rightarrow if_2 = 0,94533$;

Sendo os termos da equação outras variáveis de entrada e cálculo da planilha, *CORETYPE*, por exemplo, é o campo referente ao tipo de secundário, sendo ele de proteção ou medição, o campo recebe como entrada "0" ou "1". Nota-se que a forma como a variável foi descrita facilita a transcrição para qualquer tipo de linguagem de programação. Outro detalhe a se comentar é o fato de o conhecimento de cálculo ser um fator crucial para esse desenvolvimento.

Um outro exemplo de variável, tensão de circuito aberto na Figura 5, sendo agora de cálculo intermediário, na qual os usuários tem acesso a visualizá-la, porém não se baseiam nela durante o projeto, mas sim em outras saídas que dependem dela.

Figura 5 – Cálculo da Tensão de Circuito Aberto.

| |
|---|
| EEF= 373 |
| =EXP(-0.235027+0.49352*LN(\$AI\$20))*\$AI\$18 |

Fonte: Do autor.

Seguindo o mesmo exemplo da indução nominal, a variável referente à tensão de circuito aberto foi documentada conforme a equação (2) e o item a seguir:

- *EEF: Tensão de circuito aberto.*

$$EEF = e^{[-0,235027+0,49352 \cdot \ln(K)]} \cdot EM \quad (2)$$

3.2.2 Segurança da Informação

A organização contém toda uma estrutura de segurança da informação gerenciada pela equipe de TI em sua planta matriz. Portanto, desenvolver o novo software dentro desse sistema trará maior segurança para a aplicação, já que conta com uma equipe focada apenas em garantir tal segurança. Além disso, na nova aplicação, o usuário não terá acesso ao código fonte.

3.2.3 Controle de Versões

Referente à integridade do software, o controle de versões é feito utilizando a ferramenta GIT, que é um sistema amplamente utilizado na área de desenvolvimento de software. Ele permite que os desenvolvedores acompanhem as alterações feitas em seus projetos ao longo do tempo e trabalhem de forma colaborativa em equipes.

3.2.4 Software Expansível e de Fácil Manutenção

O padrão de arquitetura utilizado no desenvolvimento do software é o MVVM discutido na Seção 2.4.5. Nesse sentido, surge a necessidade de definir a estrutura do modelo, sendo um dos requisitos de negócio, o software deve ser expansível e de fácil manutenção, tornando essa etapa uma das mais importantes do projeto, pois caso seja feita erroneamente, seria necessário dedicar vários esforços no futuro para reestruturar o modelo.

O modelo é arquitetado seguindo os princípios citados nas Seções 2.4.1 e 2.4.6, sendo uma tarefa difícil e trabalhosa, mas com uma grande recompensa. Utilizando os conceitos de programação orientada a objetos, as classes devem ser separadas em camadas, nesse ponto, nota-se a importância dos conhecimentos de engenharia e do produto, pois mesmo que o software a ser desenvolvido tenha como foco o projeto da parte ativa de um transformador de corrente, deve-se conter classes mais gerais que estejam prontas para uma possível expansão.

O departamento de engenharia da organização é dividido entre média e alta tensão. Cada uma dessas áreas contém projetistas, calculistas e um supervisor especializados para essas classes de tensões.

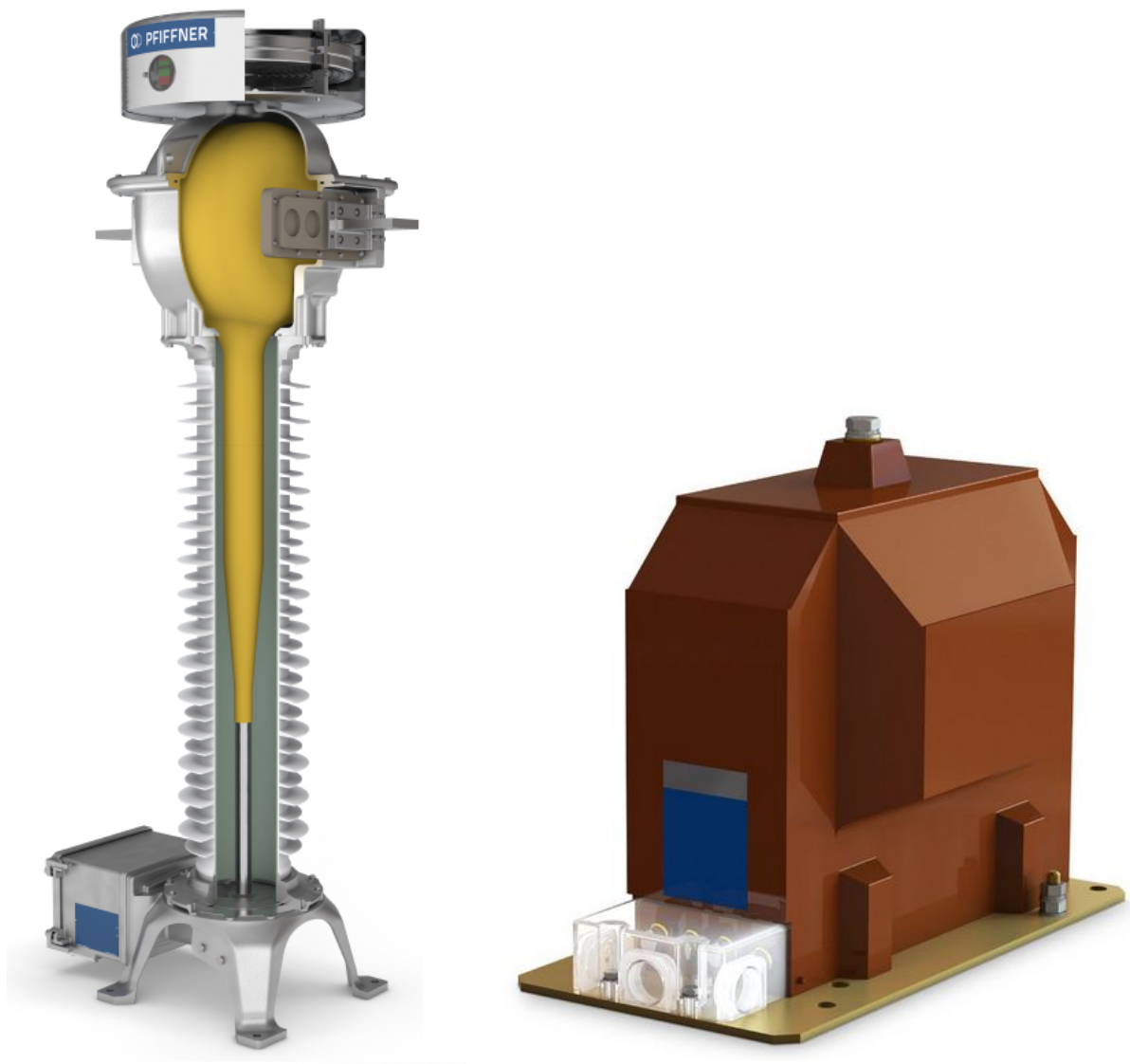
Na Figura 6, fica claro que os produtos, TCs de alta e média tensão, contêm características distintas. Portanto, na estrutura do software, o TC de média tensão não deve enxergar as entradas, métodos e saídas do TC de alta tensão, logo as classes devem ser separadas e cada uma conter suas devidas características exclusivas.

Na abertura do projeto, a aplicação deve reconhecer qual o tipo de projeto e instanciar corretamente seu modelo, seja ele de alta ou média tensão. Para isso é utilizado um padrão criacional chamado *Factory*, ele é brevemente citado na Seção 2.4.4.

A parte ativa de um TC é constituída por um único primário e por um ou mais secundários, como descrito na Seção 2.1. Por conta disso, deve-se separar as classes referentes ao primário dos secundários, já que o cálculo do primário será feito apenas uma vez, diferentemente do cálculo dos secundários.

Indo mais afundo, há também a diferença entre os secundários de medição e proteção. Por mais que eles compartilhem de características em comum, há diversas particularidades que os separam, como por exemplo a possibilidade fazer o projeto de núcleo com entreferro no secundário de proteção, ou chapas específicas que constituem o núcleo de

Figura 6 – Transformador de Alta e Média Tensão.



Fonte: (PFIFFNER, 2023)

medição para alcançar classes de exatidão definidas por normas como ABNT ou IEC.

Outro ponto importante no desenvolvimento é a forma como se armazena características dos materiais obtidos de fornecedores, como fios, chapas que compõem os núcleos, etc. Para isso tem-se o DAO e DTO descritos na Seção 2.4.3. Não é viável colocar tais características direto no código do software, pois se caso um fornecedor altere algum material, ou talvez a empresa troque de fornecedor, seria necessário liberar uma versão do software apenas para contemplar tal mudança.

Por conta disso, tais informações são guardadas num banco de dados, os objetos do tipo DAO são responsáveis pelo acesso ao BD e por isso contém atributos como a data

de criação e modificação daquele registro, além do usuário que o inseriu. Nota-se que essas informações não são necessárias para a parte de projeto do TC, apenas para controle e integridade do banco.

Logo, surgem os objetos DTO que serão preenchidos apenas com as informações necessárias do banco, conseqüentemente otimizando o tempo de download e armazenamento em cache dos registros. Para o acesso a esse objeto, utiliza-se um outro padrão criacional chamado *Singleton*, garantindo uma única instância compartilhada e um acesso global a ela no código.

A linguagem de programação utilizada para a codificação do software é C#, sendo ela orientada a objetos. Ela é uma linguagem desenvolvida pela *Microsoft* que se assemelha bastante ao Java.

Há diversas práticas que visam um código limpo, legível e de fácil manutenção. No caso da organização em questão, tais práticas são padrões a ser seguidos, toda a codificação deve seguir rigorosamente esse padrões, já que outros desenvolvedores precisam ler e dar manutenção nesses códigos.

Alguns exemplos são:

- Todos atributos e métodos do tipo *bool* devem começar com a palavra *Is*. Ex.: *IsHighVoltage*
- Todos atributos e métodos do tipo *Action* devem finalizar com a palavra *Action*. Ex.: *CloseWindowAction*
- Toda interface deve iniciar com a letra *I*. Ex.: *IProject*.
- Toda classe abstrata deve iniciar com a palavra *base*. Ex.: *BaseInstrumentTransformer*.
- Toda classe que tem a função de transportar dados do BD devem finalizar com DTO. Ex.: *AWGWireDTO*.

Além disso, a nomeação de todos os itens no código são em inglês e devem descrever de forma simples e direta sua função, não importando o tamanho. Um exemplo seria *CoreOuterDiameter*, referindo-se ao diâmetro externo do núcleo.

3.2.5 Exibir Apenas Informações Necessárias na Interface

Quando se trata de exibir apenas as informações necessárias na interface de um software, existem algumas abordagens que podem ser adotadas. Em primeiro lugar, é importante identificar as informações essenciais para a tarefa ou funcionalidade em questão. Isso pode ser feito por meio de pesquisa de usuário, análise de requisitos e compreensão das metas do usuário. Ao priorizar as informações principais, é possível destacá-las na interface, garantindo que sejam facilmente visíveis e acessíveis.

Em seguida, a simplicidade é um princípio-chave. Evitar sobrecarregar a interface com informações excessivas, pois isso pode confundir os usuários e dificultar a compreensão.

Ao manter uma abordagem minimalista, com uma quantidade adequada de informações, os usuários conseguem focar nas tarefas importantes sem distrações desnecessárias. Isso também contribui para uma experiência de usuário mais fluida e agradável.

Organizar as informações em grupos lógicos e coerentes é outra estratégia eficaz. Agrupar informações relacionadas ajuda os usuários a encontrarem facilmente o que estão procurando e entenderem a relação entre os diferentes elementos da interface. Ao criar uma estrutura organizada, os usuários podem navegar de forma mais intuitiva e eficiente, tornando o software mais fácil de usar.

Por fim, é importante realizar testes de usabilidade com usuários reais. Esses testes fornecem feedback valioso sobre a eficácia da interface em exibir as informações necessárias. Com base nos resultados dos testes, é possível fazer ajustes e melhorias para garantir que a interface esteja atendendo adequadamente às necessidades dos usuários e exibindo apenas as informações relevantes de maneira clara e concisa.

4 RESULTADOS

Este capítulo apresenta o fruto daquilo que foi obtido a partir da análise dos dados coletados nesta pesquisa. Sendo esses dados os requisitos de negócio levantados, o programa já existente, visitas técnicas a fábrica e diversas reuniões com as equipes de engenharia (usuários) e P&D (responsáveis pelo produto atual).




4.1 ARQUITETURA DO MODELO DE CÁLCULO DO SOFTWARE

Nesta seção será exposto o Model da MVVM, somente ao que se refere a parte de cálculo. Posto isto, será feita uma análise separada de diferentes classes e, por fim, o modelo completo. Essa análise contemplará, também, os relacionamentos de herança, associação, agregação e composição entre as classes conforme citado na Seção 2.4.2.

Para representar o modelo construído, utilizou-se um diagrama de classes adaptado à linguagem C#. Nesse sentido, o cabeçalho é o nome da classe, *Fields* são os atributos privados, *Properties* são as propriedades com *Get* e *Set* e *Methods* são os métodos. As propriedades apenas com *Get* são chamadas de *read only*, ou seja, apenas leitura. Como se trata de um modelo de cálculo, essas propriedades de apenas leitura estão atreladas, no geral, a um cálculo.

As propriedades e métodos que estão em itálicos, nos diagramas, significam que foram definidos como abstrato na classe base, forçando sua implementação nas classes que o herdam.

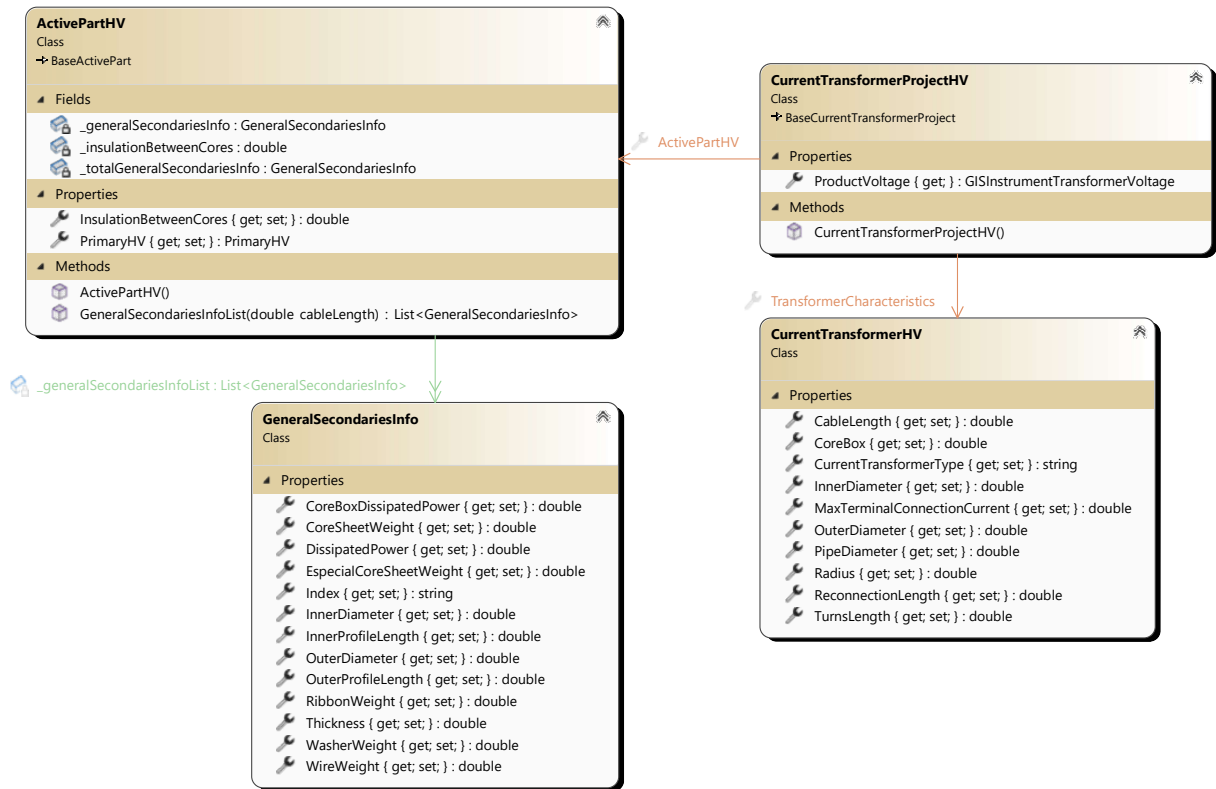
As setas presente nos diagramas desta seção são definidas da seguinte forma:

-  : representa o relacionamento de herança.
-  : representa os relacionamentos de associação, agregação ou composição.
-  : representa, através de uma lista, os relacionamentos de associação, agregação ou composição de um para muitos.

As cores das setas foram definidas como laranja para quando a associação é feita com uma propriedade, ou seja, uma classe foi definida como propriedade de uma outra classe. Já a cor verde significa que esse relacionamento é através de um atributo. Além disso, a ilustração de um cadeado nos atributos e métodos representam que são privados.

Na Figura 7 tem-se as classes referentes à Alta Tensão. Um transformador contém tanto uma parte ativa como parte mecânica, porém, mesmo que o software não contemple o projeto da parte mecânica, o modelo foi pensado na possibilidade de tal implementação. A classe *ActivePartHV* compõe a *CurrentTransformerProjectHV*, já que o projeto do transformador de corrente é incoerente sem uma parte ativa. O projeto da parte ativa, também, não existe sem o projeto de TC, respeitando o relacionamento de composição que é feito através de uma propriedade.

Figura 7 – Arquitetura do Modelo - Alta Tensão.



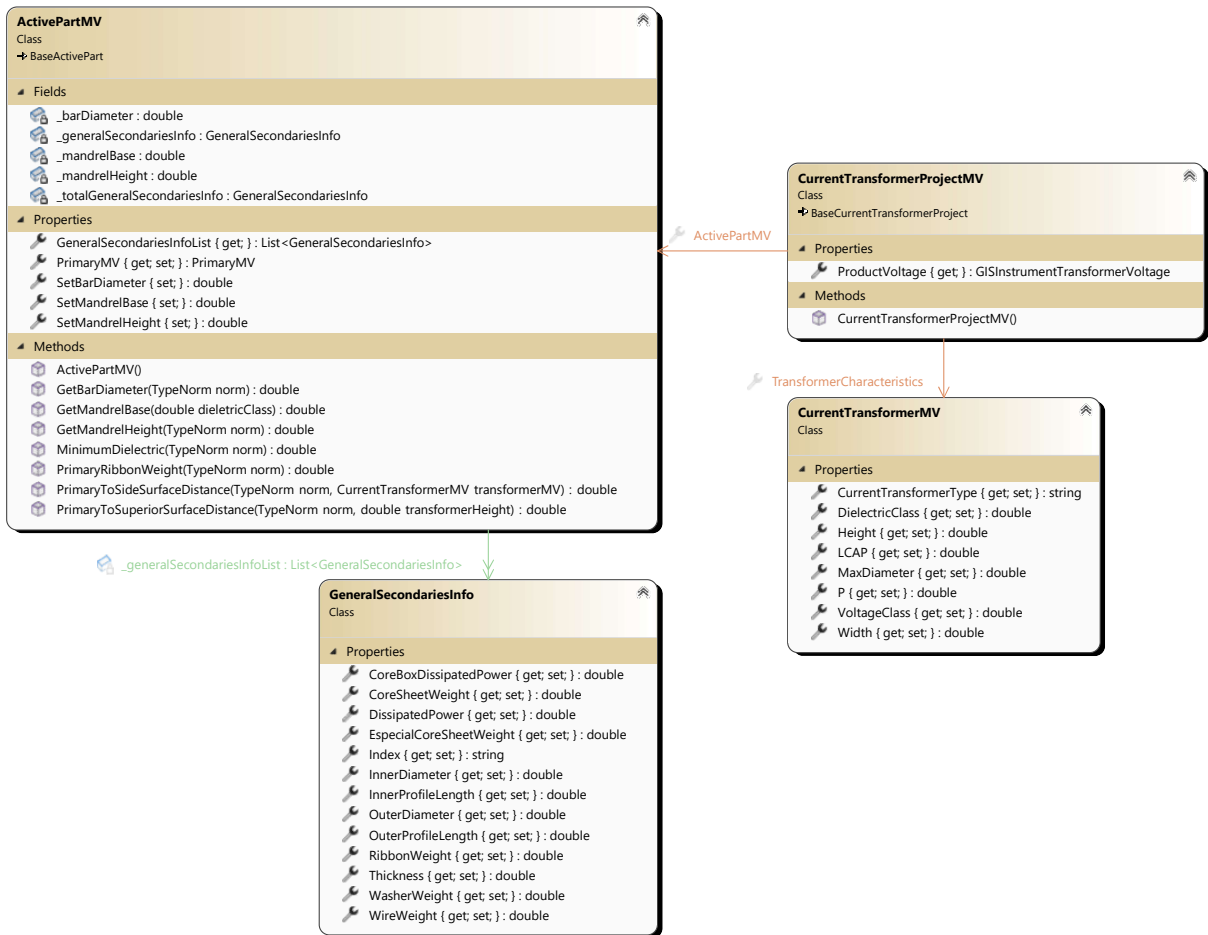
Fonte: Do autor

Como visto na Figura 6, ao definir o tipo de transformador, ele vem com características da parte ativa e parte mecânica atreladas, portanto a classe *CurrentTransformerHV* é responsável por estes componentes, sendo "HV" a abreviação para *High Voltage*. Essas características são tabeladas e serão armazenadas em um banco de dados, então, cabe ao usuário definir o tipo de transformador e esse objeto será preenchido e será utilizado para o projeto da parte ativa. Diante disso, essa classe compõe o projeto de transformadores de alta tensão, já que sem as características do transformador não há como fazer o projeto.

A classe *GeneralSecondaryInfo* representa as informações gerais do secundário. O método *GeneralSecondariesInfoList* é responsável por percorrer todos os secundários coletando tais características e, por fim, também devolve um valor total para cada uma delas. Tendo como exemplo o peso do fio, *Wire Weight*, o calculista necessita do peso total dos fios de cobre dos secundários para se colocar na lista de materiais daquele projeto, que é administrada pelo sistema de gerenciamento da empresa. O relacionamento entre a parte ativa e as informações dos secundários é de composição, já que essas informações são necessárias para todo projeto da parte ativa. Essa associação é representada por um atributo privado, onde o método *GeneralSecondariesInfoList* o retorna após preenchê-lo.

Já para a média tensão, na Figura 8, nota-se que as características se diferenciam da alta tensão. Nesse caso, por exemplo, o calculista tem que se atentar com o dielétrico entre o primário e o secundário. O *Minimum Dielectric* é um método que envolve percorrer

Figura 8 – Arquitetura do Modelo - Média Tensão.



Fonte: Do autor

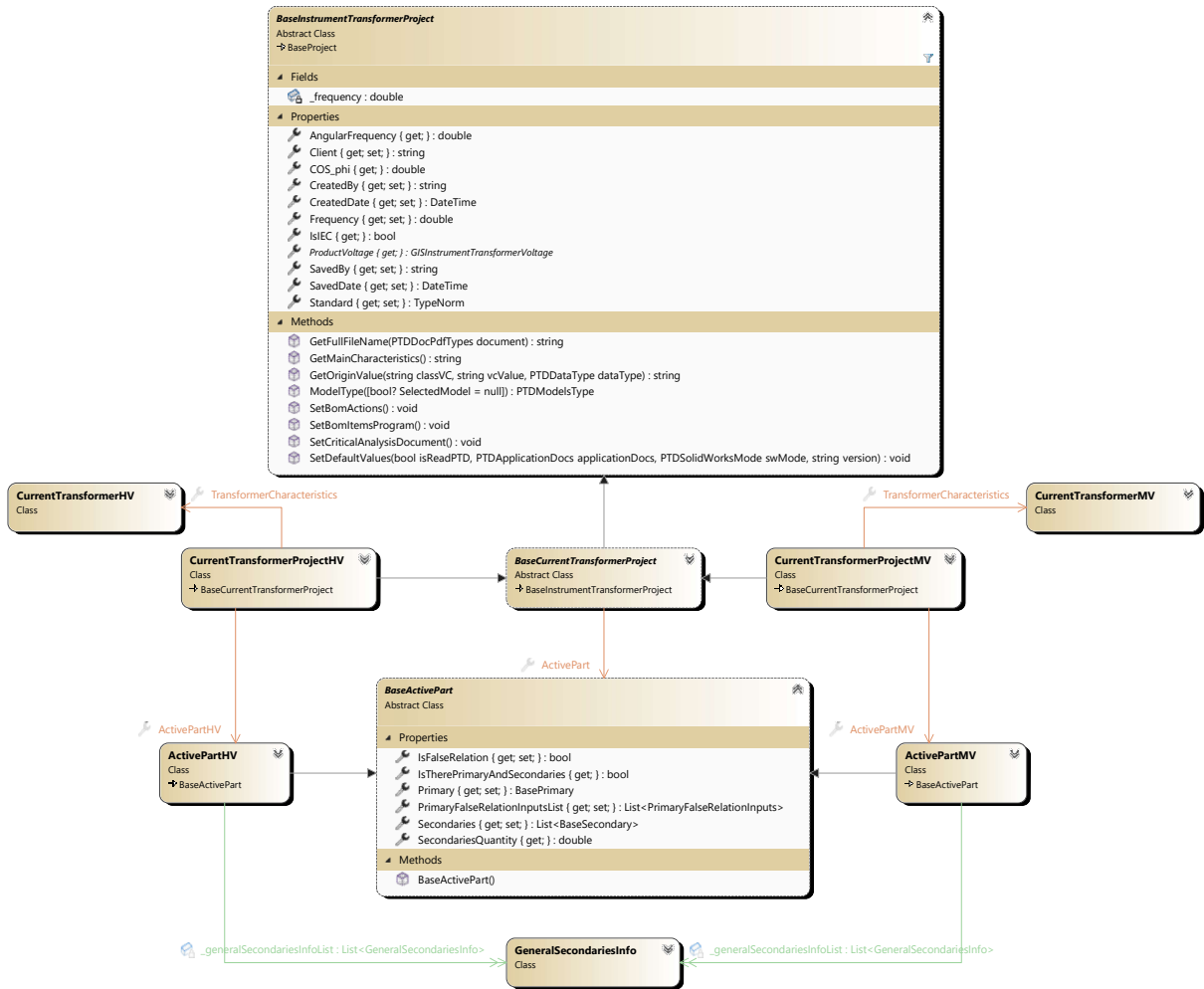
todos os secundários e devolver o menor dielétrico entre eles. O relacionamento entre as classes da Figura 8 respeitam o mesmos conceitos da alta tensão.

Na Figura 9 tem-se a composição do modelo de projeto com a parte ativa. As classes que foram discutidas anteriormente foram reduzidas para facilitar a visualização do conjunto. A classe *BaseInstrumentTransformerProject* foi criada para representar todas as características em comum entre os transformadores para instrumentos produzidos na planta. É possível visualizar, através da seta preenchida, que a classe *BaseCurrentTransformerProject* herda dela.

As classes referentes à parte ativa da média e alta tensão herdam da *BaseActivePart*, quem contém as características em comum entre elas. A quantidade de secundários, por exemplo, sendo uma propriedade em comum, é importante no momento de se definir a espessura final. Como há isolamento entre os secundários, a espessura final não é somente a soma das espessuras.

Outro ponto a se destacar seriam as propriedades *Primary* e *Secondaries*. Elas foram definidas através das classes bases do primário e secundário, então, a partir do polimorfismo, elas são utilizadas em cálculos específicos da média e alta tensão nas classes

Figura 9 – Arquitetura do Modelo - Parte Ativa.



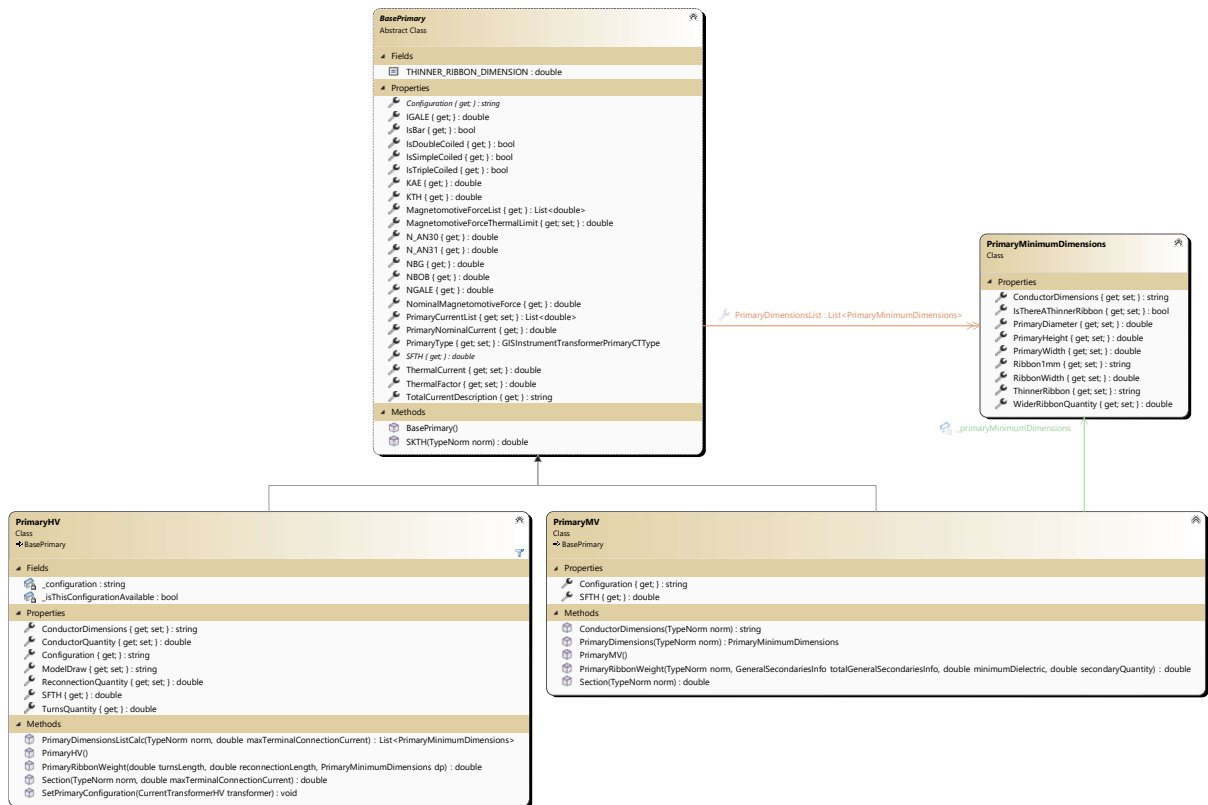
Fonte: Do autor

ActivePartHV e *ActivePartMV*.

Na Figura 10 tem-se a classe base do primário com as informações em comum do primário de média e alta tensão. É possível visualizar, também, a relação de herança entre elas. As propriedades dessa classe, com exceção do *PrimaryDimensionsList*, que contém *Get* e *Set* são entradas de projeto, sendo a corrente térmica, o fator térmico e a "lista de correntes do primário" especificadas pelo cliente. Na Seção 2.1.3 é comentado sobre as correntes nominais do primário. O número de derivações no secundário é determinado pelo número dessas correntes.

A propriedade *PrimaryDimensionsList* representa o relacionamento de composição com a classe *PrimaryMinimumDimensions*, pois é uma lista que é preenchida na classe base do primário e utilizada na média tensão para automatizar o projeto de algumas características do primário através do método *PrimaryDimensions(TypeNorm norm)*. Para a alta tensão, essa propriedade é utilizada no método *PrimaryDimensionsListCalc(TypeNorm norm, double maxTerminalConnectionCurrent)*, onde é retornado uma lista com características das fitas de cobre utilizadas no primário, então o projetista tem acesso a essas

Figura 10 – Arquitetura do Modelo - Primário.



Fonte: Do autor

informações e fica a critério dele escolher qual opção, dentre essa lista, que se encaixa melhor no projeto.

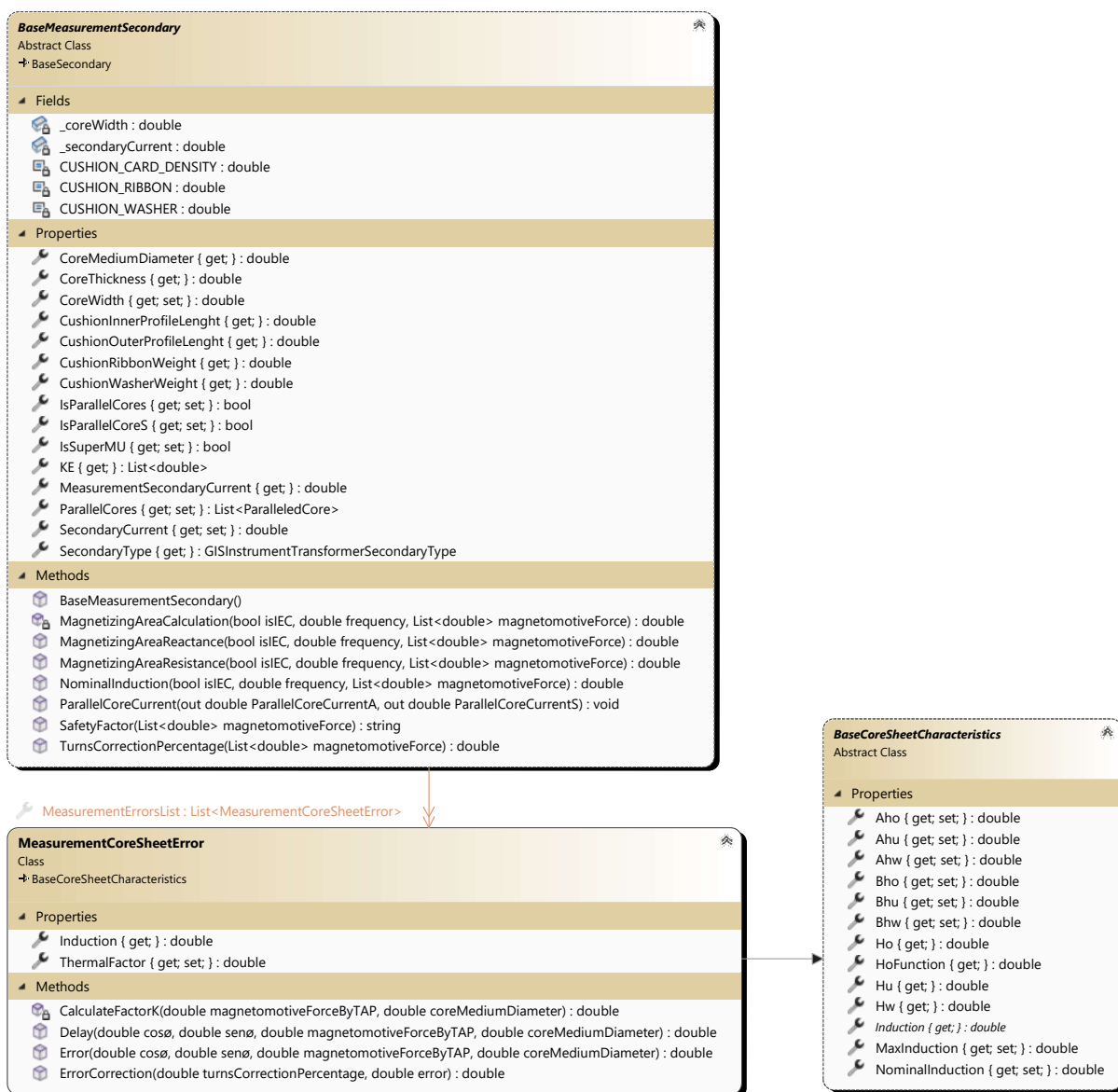
O peso do cobre do primário, *Primary Ribbon Weight*, no caso da média tensão, está diretamente ligado a características da parte ativa, então, mesmo sendo uma propriedade física em comum com o projeto de alta tensão, eles dependem de parâmetros distintos e, por conta disso, não são definidos numa classe base.

O primário de alta tensão também tem suas características específicas, como, por exemplo, o *ModelDraw*. Esse método retorna, através de cálculos e condições, o código do desenho dos enrolamentos do primário que é utilizado na fábrica para a produção do transformador.

Na Figura 11 tem-se a classe base do secundário de medição. Como dito na Seção 2.1.4, os critérios de precisão do núcleo de medição são mais rigorosos, portanto a qualidade do material que compõe o núcleo influencia diretamente nas medições.

A classe abstrata *BaseCoreSheetCharacteristics* contém as características da chapa do núcleo que são obtidas através dos fornecedores. Essas informações são armazenadas no banco de dados. As propriedades *read only* dessa classe estão atreladas a um cálculo. Cada classe que estende ela, como por exemplo a *MeasurementCoreSheetError*, tem uma forma específica de calcular a indução. A partir desse valor de indução, é feito uma busca no banco de dados com as características do núcleo referente àquele valor de indução

Figura 11 – Arquitetura do Modelo - Secundário de Medição.



Fonte: Do autor

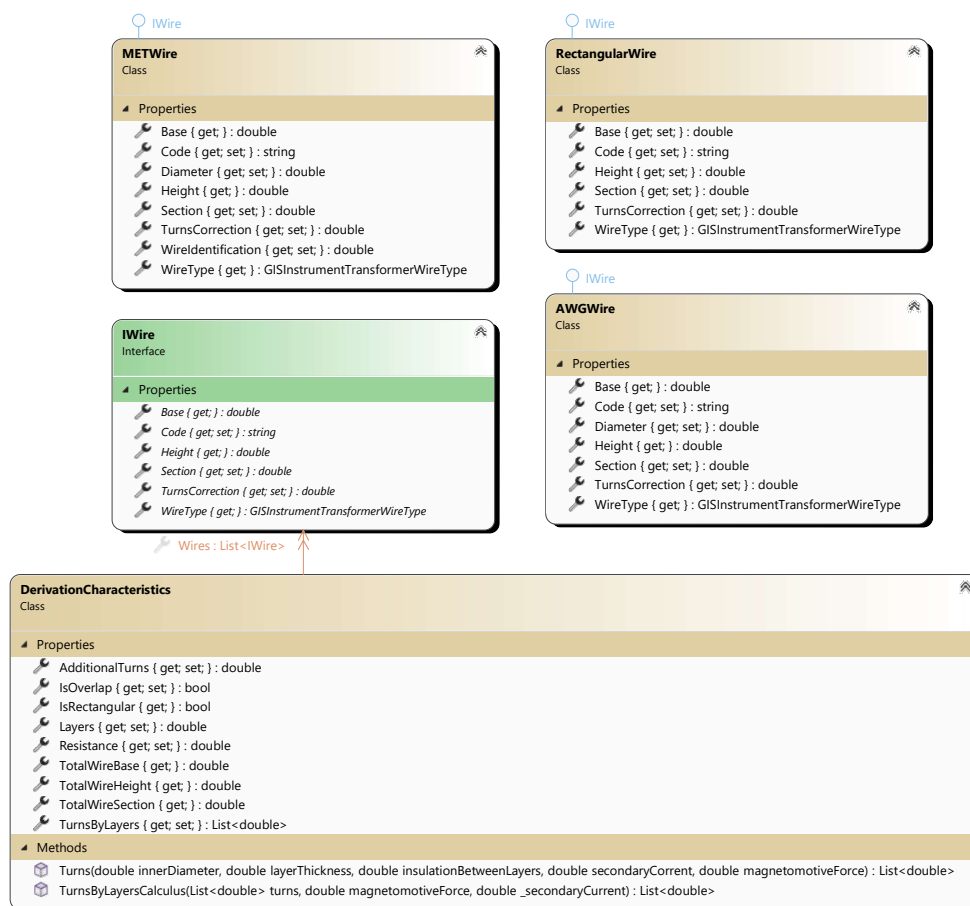
calculado.

No projeto do secundário de medição, há a relacionamento de composição com a classe *MeasurementCoreSheetError*, pois é necessário cumprir uma determinada exatidão para todos os fatores especificados pelas normas.

Durante o processo de cálculo do secundário de medição, o usuário tem diversas ferramentas para enquadrar seu projeto nas normas especificadas, como, por exemplo, correção do número de espiras. Essa correção é feita nos fios de cobre que bobinam o núcleo e sua representação no modelo pode ser vista na Figura 12. Como dito na Seção 2.1.3, as derivações dependem das relações de correntes do primário. Um exemplo de derivações pode ser vista na Tabela 1.

IWire é uma interface que contém as características em comuns entre os tipos

Figura 12 – Arquitetura do Modelo - Derivações.



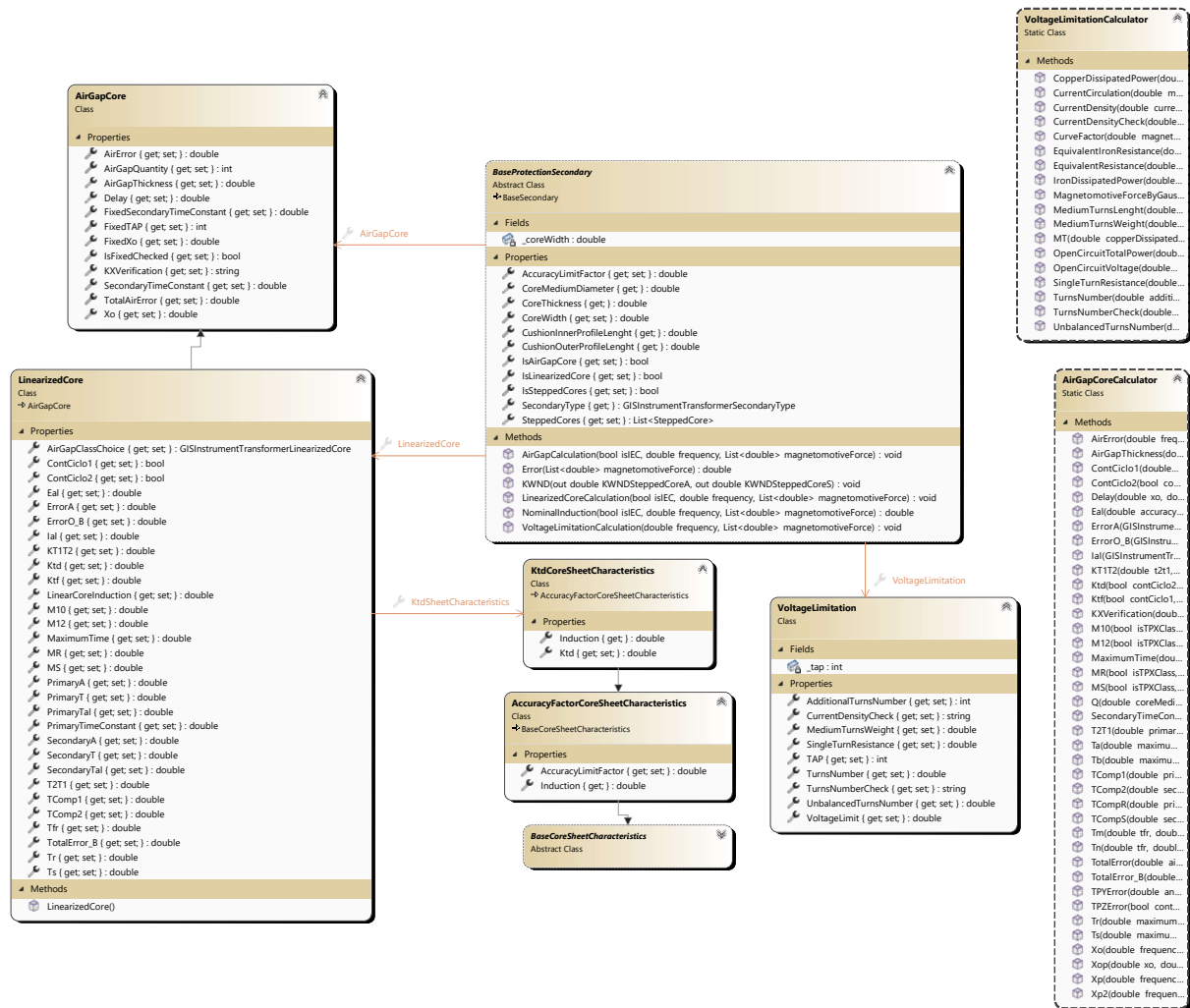
Fonte: Do autor

de fios. As classes *METWire*, *RectangularWire* e *AWGWire* implementam essa interface. Então, a partir do tipo de fio que o usuário escolher para aquela determinada derivação, o objeto será criado através do padrão de arquitetura criacional *Factory*. A propriedade *Wires* foi definida como uma lista de fios na classe das características da derivação, pois é possível unir e bobinar o núcleo com fios de diferentes dimensões, desde que sejam do mesmo tipo. Diante disso, a relação entre a classe de fios e derivações é de composição, já que é incoerente um existir sem o outro.

Na Figura 13 tem-se a classe base do secundário de proteção. Analisando a classe *LinearizedCore*, ela herda da classe *AirGapCore* e estão relacionadas ao núcleo com entreferro. Elas foram definidas separadamente na classe na *BaseProtectionSecondary* devido aos projetos de transformadores PR, PXR, TPX, TPY e TPZ descritos na Seção 2.1.5. Diante disso, o relacionamento delas com o secundário de proteção é de agregação, já que o projeto de núcleo com entreferro depende das especificações dos clientes.

Ao fazer um projeto de secundário de "núcleo linearizado", classes TPX, TPY e TPZ, há o interesse no controle do fluxo residual e do regime transitório durante um curto circuito na rede, essa nomenclatura é popularmente utilizada devido ao fato de a curva de magnetização do núcleo ter um comportamento próximo ao linear. O projeto apenas

Figura 13 – Arquitetura do Modelo - Secundário de Proteção.



Fonte: Do autor

com os entreferros, *AirGapCore*, classe PR ou PXR, não está interessado nesse regime transitório, tornando-o menos rigoroso e, conseqüentemente, mais barato.

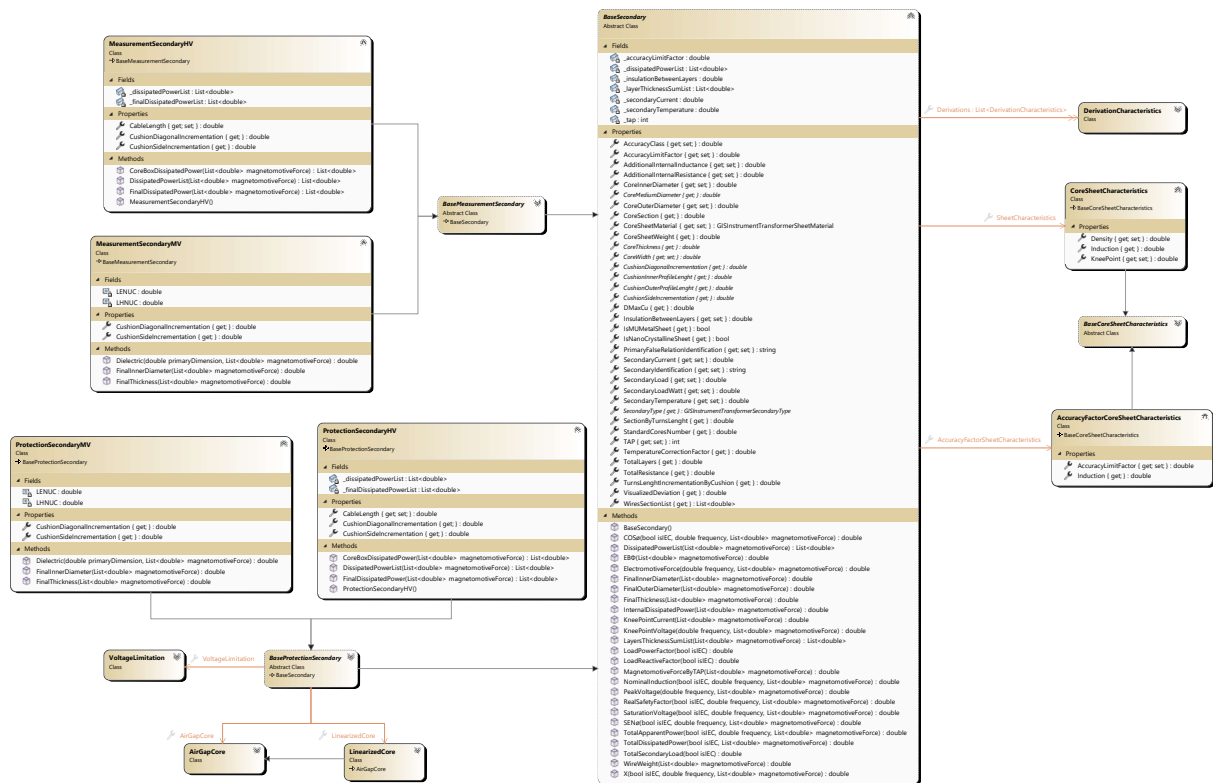
A classe *LinearizedCore* tem a relação de composição com a classe *KtdCoreSheetCharacteristics* e é representada pela propriedade *KtdSheetCharacteristics*. Essa classe herda da *AccuracyFactorCoreSheetCharacteristics* que estende da *BaseCoreSheetCharacteristics*. Essas camadas de relações se devem ao fato da classe *AccuracyFactorCoreSheetCharacteristics* conter uma propriedade que é utilizada em conjunto com a "Ktd" para fazer o cálculo da indução. Como descrito anteriormente, a partir desse cálculo da indução, é feito uma busca no banco de dados para obter as características do núcleo que equivalem a essa indução e, a partir disso, efetuar outros cálculos.

Já a limitação de tensão, *VoltageLimitation*, também é uma particularidade do secundário de proteção. Em transformadores de corrente é importante ter uma carga conectada aos secundários, pois, caso não haja, a tensão de circuito aberto pode chegar a grandes valores. Portanto, esse limitador de tensão tem uma função de segurança. Nesse

sentido, a classe *VoltageLimitation* tem o relacionamento de agregação com o secundário de proteção, já que não é todo projeto que contém a limitação de tensão.

O cálculo demonstrado na Figura 5 é referente ao projeto envolvendo o limitador de tensão, mas como ele é um cálculo intermediário que o usuário não está interessado e não é utilizado para fazer, por exemplo, o *plot* de um gráfico, ele não foi definido na classe. À direita, ainda na Figura 13, é possível visualizar as classes e seus métodos estáticos utilizados para os projetos que envolvem entreferro e/ou limitação de tensão.

Figura 14 – Arquitetura do Modelo - Secundários.



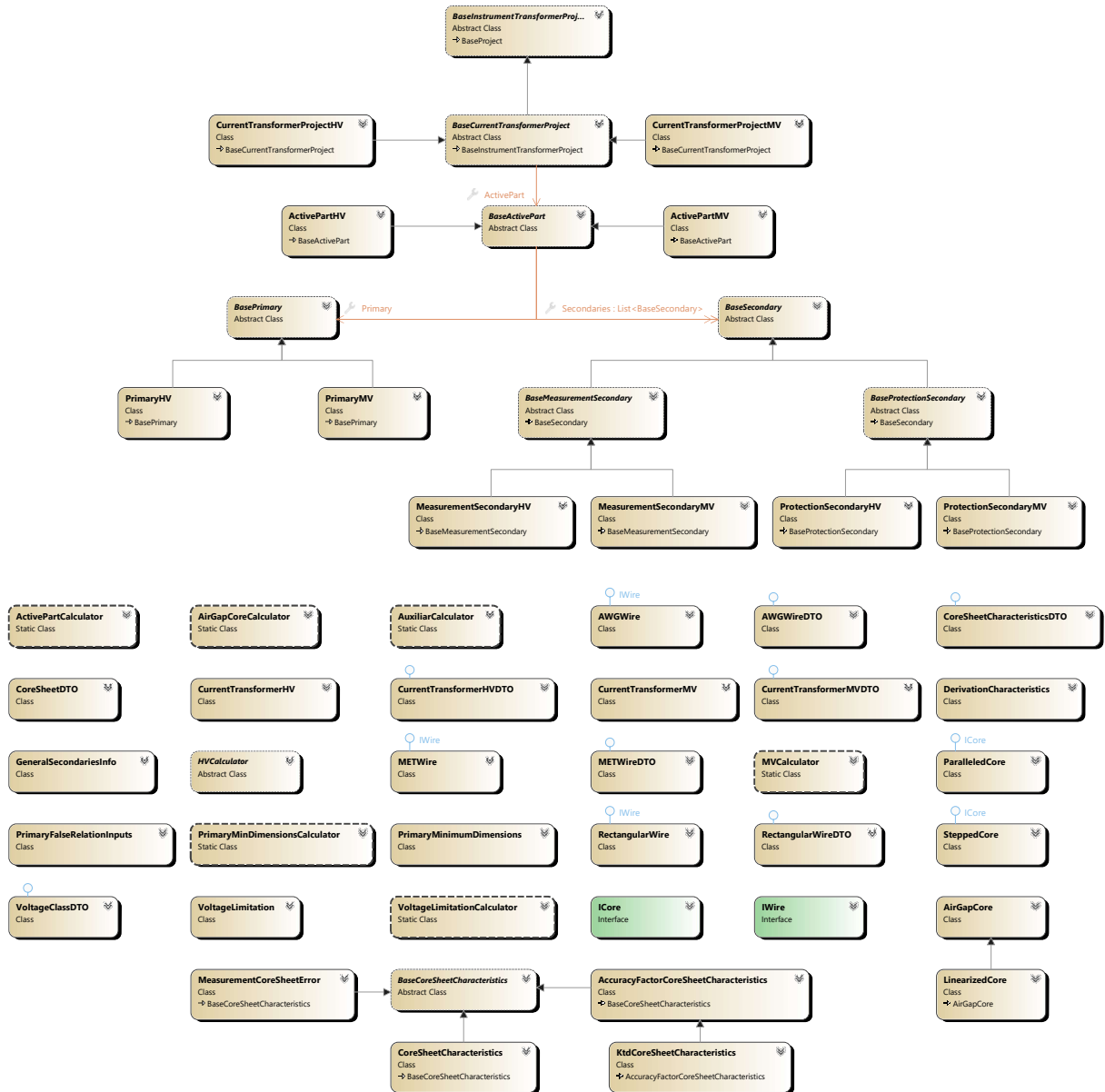
Fonte: Do autor

Referente aos secundários, tem-se a classe *BaseSecondary* onde contempla todas as características em comum dos secundários. Ela também define objetos do tipo *BaseCoreSheetCharacteristics*, sendo que a classe *CoreSheetCharacteristics* estende ela, porém, nesse contexto, a indução é baseada no ponto "joelho" da curva de saturação, enquanto que na classe *AccuracyFactorCoreSheetCharacteristics*, por exemplo, é baseado no fator limite de exatidão, ou seja, como evidenciado anteriormente essas características necessitam de valores distintos do fornecedor para se fazer os cálculos. A relação da classe *BaseSecondary* com as que foram citadas é de composição, já que o projeto dos secundários dependem dessas características para se fazer determinados cálculos.

Na Figura 14 é possível visualizar a relação da classe *BaseSecondary* com as características da derivação. Nesse sentido, a relação entre elas também é de composição, já que todo projeto de secundário contém ao menos uma derivação. É evidenciado, também, a

relação entre os secundários, sendo que as classes referentes a média e alta tensão estendem os secundários de proteção e medição que, por fim, herdam do *BaseSecondary*.

Figura 15 – Arquitetura do Modelo - Completa.



Fonte: Do autor

Por fim, na Figura 15, tem-se o modelo de cálculo completo do software de projeto de transformadores de corrente. Como dito anteriormente, a classe *BaseActivePart* define uma lista de *BaseSecondary* e uma propriedade de *Base Primary*, portanto, através do polimorfismo, as partes ativas de média e alta tensão acessam suas informações específicas.

A voltagem do projeto só pode ser definida uma vez, sendo ela na criação do projeto. Então, a partir disso, todo momento que o projeto for aberto por algum usuário, os secundários e primário serão instanciados, respeitando a voltagem do projeto. Dessa forma as informações e desenvolvimento são filtradas e separadas adequadamente.

Foi optado por não discutir, nessa Seção, algumas das classes que podem ser visualizadas na estrutura completa, já que algumas delas, por exemplo, envolvem ser informações sensíveis da empresa em questão. Além disso, nesse modelo, foi omitido os relacionamentos de associação entre as classes principais, como *Current Transformer Project*, *Active Part*, *Primary* e *Secondary*, e os objetos que as compõe. Essa abordagem foi adotada pois tais associações já foram discutidas previamente e, dessa forma, como na Figura 15, é possível visualizar claramente os relacionamentos entre as classes principais.

4.2 EXEMPLOS DOS CÓDIGOS IMPLEMENTADOS

A implementação do código da indução nominal do núcleo, descrita na Seção 3.2.1, pode ser vista no Código Fonte 1. Nota-se que os parâmetros de frequência e tipo de norma são de uma camada mais acima do projeto, enquanto que este cálculo está na camada de cálculo do secundário. Já o parâmetro da força magnetomotriz vem do projeto do primário. Além disso, foi utilizado o *override*, ou seja, o método foi sobrescrito da sua classe base.

Os cálculos referentes às classes de *Air Gap Core*, *Linearized Core* e *Voltage Limitation* foram implementados utilizando métodos estáticos ao invés do corpo da classe como a Indução Nominal. Como não são todos os projetos de secundário de proteção que envolvem essas características, tais objetos são instanciados apenas quando necessário.

Um detalhe a se notar na implementação do método da tensão de circuito aberto, no Código Fonte 2, é a importância dos conhecimentos de cálculo no desenvolvimento, pois a função logarítmica $\ln(x)$ é indeterminada para $x \leq 0$, portanto, ao colocar a condição de que o fator da curva tem que ser maior que zero, o software não irá fechar sozinho caso venha um valor inesperado.

Código Fonte 1 - Implementação em C# da Indução Nominal.

```

1 public override double NominalInduction(bool isIEC, double frequency,
2                                     List<double> magnetomotiveForce)
3 {
4     double calculation = 0;
5     double magnetomotiveForceByTap = MagnetomotiveForceByTap(magnetomotiveForce, TAP);
6
7     if (frequency != 0 && magnetomotiveForceByTap != 0 && CoreWidth != 0 &&
8         CoreThickness != 0)
9     {
10        calculation = K * TotalApparentPower(isIEC, frequency, magnetomotiveForce)
11                    * Math.Pow(10, 4) * 1.05
12                    / (frequency * magnetomotiveForceByTap * CoreWidth
13                      * CoreThickness);
14
15        double condition = IsNanoCrystallineSheet
16                          ? CRYSTALLINE : NO_CRYSTALLINE;
17
18        calculation = calculation / condition;
19    }
20    return calculation;
21 }

```

Código Fonte 2 - Implementação em C# da Tensão de Circuito Aberto.

```
1 public static double OpenCircuitVoltage(double GaussMagnetomotiveForce ,
2                                     double curveFactor)
3 {
4     double calculation = 0;
5
6     if (curveFactor > 0)
7     {
8         calculation = Math.Exp(-0.235027 + 0.49352 * Math.Log(curveFactor))
9             * GaussMagnetomotiveForce ;
10    }
11    return calculation ;
12 }
```

4.3 INTERFACE DO USUÁRIO

O estilo e design da interface do software é baseada nas cores da organização. Além disso, como no programa já existente, as cores nos campos de entrada e saída são diferentes, sendo fundamental na usabilidade e experiência do usuário. Essa diferenciação visual facilita a compreensão dos elementos interativos da interface. Ao utilizar cores distintas, os usuários conseguem identificar rapidamente quais campos são destinados à inserção de informações e quais são campos de exibição de resultados ou informações geradas pelo sistema.

Como dito no desenvolver do levantamento dos requisitos, agrupar informações relacionadas ajuda os usuários a encontrar facilmente o que estão procurando e entender a relação entre os diferentes elementos da interface, portanto, a interface foi separada em grupos utilizando *expanders* e em subgrupos referente às entradas e saídas.

O recurso *visibility* do *Windows Presentation Foundation* (WPF), sistema gráfico onde está foi desenvolvido o software, é uma ferramenta para controlar a visibilidade de elementos de interface, permitindo ocultar elementos de forma eficiente, liberando espaço na tela e mantendo a fluidez do layout da aplicação. Sua utilização adequada contribui para uma melhor experiência do usuário e uma interface mais flexível e adaptável. Ao utilizar o *visibility.collapsed*, que significa ocultar, na tela, um determinado elemento ou grupo de informações, o desempenho do programa não é afetado, já que a interface não carrega as informações daquilo que está oculto, dessa forma, por exemplo, pode-se criar apenas uma view para o projeto de secundários e utilizar dessa ferramenta para ocultar as informações que não são daquele cálculo.

5 CONCLUSÃO

Através dessa pesquisa, foi possível identificar os principais requisitos e funcionalidades que o software deve abranger para atender às necessidades específicas dos engenheiros e profissionais envolvidos no cálculo e dimensionamento da parte ativa de transformadores de corrente.

Ao compreender e detalhar os requisitos, no desenvolvimento desse trabalho, foi possível ter uma visão clara do escopo do software a ser construído, evitando equívocos, garantindo que todas as partes interessadas estejam alinhadas e ajudando a evitar retrabalhos desnecessários.

Em conclusão, essa pesquisa alcançou seus objetivos. O primeiro objetivo envolveu realizar uma revisão dos conceitos relevantes para o estudo, estabelecendo uma base teórica para as etapas subsequentes. O segundo objetivo concentrou-se em estudar e compreender o modelo e o sistema existentes utilizados pela empresa, fornecendo informações valiosas sobre as práticas atuais e áreas potenciais para melhorias.

O terceiro objetivo teve como meta coletar e identificar os requisitos de negócio, uma fase que permitiu uma definição clara das funcionalidades do software e seus aspectos de segurança. Esses requisitos foram então utilizados no quarto objetivo, levando à arquitetura do modelo de cálculo e ao desenvolvimento de uma interface intuitiva e amigável que seguiu padrões estabelecidos. Já o quinto objetivo teve seu foco no desenvolvimento do código do software utilizando a linguagem de programação C#, seguindo as boas práticas de programação.

Um ponto a se destacar é o fato de que é comum que empresas que desenvolvem software para uso interno tenham políticas de proteção de dados, especialmente quando se trata de informações confidenciais e proprietárias. Diante disso, não exibir, por exemplo, as telas do software foi uma medida para evitar divulgação não autorizada de informações.

Por fim, com base nos requisitos levantados, acredita-se que o software proposto terá um impacto significativo na empresa, proporcionando uma ferramenta valiosa para uma constante ampliação e diminuição de custos. Contribuindo, assim, para o avanço tecnológico e melhoria contínua do sistema em questão.

REFERÊNCIAS

- ALUR, Deepak; CRUPI, John; MALKS, Dan. **Core J2EE Patterns: Best Practices and Design Strategies**. [S.l.]: Prentice Hall, 2003.
- BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice (3rd Edition)**. [S.l.]: Addison-Wesley, 2012.
- BAUER, Christian; KING, Gavin. **Java Persistence with Hibernate**. [S.l.]: Manning Publications, 2002.
- BHIM SINGH, R.; CHANDRA, Ambrish; KAMAL, Sanjeev. **Power Electronics and Renewable Energy Systems: Proceedings of ICPERES 2014**. [S.l.]: Springer, 2013.
- BROM, Helko E. van den; LEEUWEN, Ronald van; RIETVELD, Gert; HOUTZAGER, Ernest. Voltage Dependence of the Reference System in Medium- and High-Voltage Current Transformer Calibrations. **IEEE Transactions on Instrumentation and Measurement**, v. 70, p. 1–8, 2021.
- GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. [S.l.]: Addison-Wesley Professional, 1994.
- GEHTLAND, Josh. **Windows Presentation Foundation 4.5 Cookbook**. [S.l.]: O'Reilly Media, 2011.
- GOSSMAN, John; SHURTS, Laurent. **Model-View-ViewModel (MVVM) Explained**. [S.l.]: Microsoft Press, 2012.
- GOTTESDIENER, Ellen; GORMAN, Mary. **Discover to Deliver: Agile Product Planning and Analysis**. [S.l.]: EBG Consulting, 2012.
- HÉROUX, P. **Power Transformer Handbook**. [S.l.]: Springer, 2011.
- HÉROUX, Pierre. **Power Transformer Handbook**. [S.l.]: Springer, 2011.
- IEC. **IEC 60044-1: Instrument Transformers - Part 1: Current Transformers**. [S.l.: s.n.], 1996. International Electrotechnical Commission.

- IEC. **IEC 60044-6: Instrument Transformers - Part 6: Requirements for protective current transformers for transient performance.** [*S.l.: s.n.*], 1992. International Electrotechnical Commission.
- IEC. **IEC 61869-2: Instrument Transformers - Part 2: Additional requirements for current transformers.** [*S.l.: s.n.*], 2012. International Electrotechnical Commission.
- KARNAL, Leandro. **O Coração das Coisas.** São Paulo: Editora Contexto, 2019.
- LEFFINGWELL, Dean; WIDRIG, Don. **Managing Software Requirements: A Unified Approach.** [*S.l.*]: Addison-Wesley Professional, 2003.
- LIPPERT, Scott. **Data Binding in Windows Presentation Foundation.** [*S.l.*]: Microsoft Press, 2010.
- MARTIN, Robert C. **Agile Software Development, Principles, Patterns, and Practices.** [*S.l.*]: Prentice Hall, 2002.
- MARTIN, Robert C. **Agile Software Development, Principles, Patterns, and Practices.** [*S.l.*]: Pearson Education, 2003.
- MVVM Pattern. [*S.l.: s.n.*], 2022. Disponível em: <https://i.stack.imgur.com/3XnBN.png>.
- MVVM Pattern. [*S.l.: s.n.*], 2023. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/architecture/maui/media/mvvm-pattern.png>.
- ABNT. **NBR 6856: Transformador de Corrente com Isolação Sólida para Tensão Máxima Igual ou Inferior a 52 kV - Especificação e ensaios.** [*S.l.: s.n.*], 1992.
- O'CONNOR, Richard V.; LAPLANTE, Phillip A. **A Guide to the Project Management Body of Knowledge (PMBOK Guide), 5th Edition.** [*S.l.*]: Project Management Institute, 2012.
- PAIC, I.; KRAJTNER, D.; NENADIC, Z. A new approach to design and construction process of instrument transformers. *In: EUROCON 2005 - The International Conference on "Computer as a Tool".* [*S.l.: s.n.*], 2005. v. 1, p. 527-530.

PEREIRA, Paulo Sérgio; PEREIRA JUNIOR, Paulo Sérgio; LOURENÇO, Gustavo Espinha. Parte 1 - Características de Transformadores de Corrente de Acordo com as Normas IEC 60044-1, 60044-6, 61869-2, IEEE C57.13, NBR 6856 1992 e 2015 e do Technical Report TR 61869-100. **CONPROVE Engenharia**, 2017.

PEREIRA, Paulo Sérgio; PEREIRA JUNIOR, Paulo Sérgio; LOURENÇO, Gustavo Espinha. Parte 2 - Características de Transformadores de Corrente de Acordo com as Normas IEC 60044-1, 60044-6, 61869-2, IEEE C57.13, NBR 6856 1992 e 2015 e do Technical Report TR 61869-100. **CONPROVE Engenharia**, 2017.

PIFFNER. **Transformador de Corrente Indutivo JOF**. Acessado em 10 de julho de 2023. 2023. Disponível em: <https://www.piffner-group.com/pt/produtos-e-solucoes/detalhes/transformador-de-corrente-indutivo-jof>.

POHL, Klaus; RUPP, Chris. **Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam**. [S.l.]: Rocky Nook, 2010.

ROBERTSON, Suzanne; ROBERTSON, James. **Mastering the Requirements Process: Getting Requirements Right**. [S.l.]: Addison-Wesley Professional, 2012.

ROZANSKI, Nick; WOODS, Eóin. **Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives**. [S.l.]: Addison-Wesley, 2011.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. **The Unified Modeling Language User Guide**. Reading, MA: Addison-Wesley, 1999.

SMITH, Jason. **Pro MVVM with Xamarin.Forms: Architecture and Patterns for Mobile Development**. [S.l.]: Apress, 2016.

SOMMERVILLE, Ian. **Software Engineering, 10th Edition**. [S.l.]: Pearson, 2016.

WIEGERS, Karl E.; BEATTY, Joy. **Software Requirements, 3rd Edition**. [S.l.]: Microsoft Press, 2013.