UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

CÁSSIO MIGUEL ENTRUDO

Uma suíte de ferramentas para apoio na criação de projetos de automação com foco em inspeção visual

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Profª. Dr. Renata Galante

Porto Alegre
2023

AGRADECIMENTOS

Esta longa caminhada finalmente chegou ao seu fim. Muitas vezes desacreditei que ela chegaria, porém eu nunca desisti. A frase mais clichê que existe: "Só eu sei o que eu passei", infelizmente terá de ser empregada neste momento. Apesar de tudo, eu sempre tive uma imensa sorte em minha vida. O destino sempre me rodeou e me presenteou com pessoas maravilhosas.

Agradeço aos meus pais, José Miguel e Célia Terezinha, que mesmo em suas simplicidades, sem mesmo entenderem o que eu estudo até hoje, me criaram e me deram tudo que estava ao alcance deles. Me ensiram os princípios da vida que contribuiram para eu ser a pessoa que hoje eu sou. Eu amo vocês! Além deles, agradeço à minha amada irmã Rita de Cássia, que nunca mediu esforços para me ajudar a alcançar meus sonhos.

Um agradecimento infinito à minha noiva, Gabriela Florisbal. Obrigado por tudo que fizeste e ainda fazes por mim. Obrigado por quebrar o gelo que me deixou congelado por diversos momentos. Obrigado por acreditar em mim, pegar na minha mão e não me deixar cair. Sem você eu jamais teria conseguido. Eu te amo! Um agradecimento também à toda tua família, que por muitas vezes me alegrou e tornou os momentos difícies mais leves.

Não poderia deixar de agradecer à professora Renata Galante, que me acolheu, acreditou em mim e me ajudou a tornar este trabalho possível. Mesmo com muita correria, sempre querida e disposta a ajudar. Obrigado, professora! :)

Agradeço também à todos amigos e colegas que fiz durante esta jornada. Colegas de trabalho, de faculdade, de treinos de CrossFit e de todos os lugares por onde passei. Por fim, agradeço à Deus por toda essa experiência que vivi. Com certeza tinha que ser dessa maneira para que eu me tornasse quem eu me tornei.

RESUMO

Os sistemas de automação com foco em inspeção visual implementados na indústria possuem muitas especificidades que dependem do produto final. Tal característica dificulta o seu desenvolvimento, fazendo com que cada novo projeto seja codificado do zero. Atualmente, existe uma dependência dos programadores para codificar cada novo sistema. Desta forma, o objetivo geral deste trabalho foi criar uma suíte de ferramentas que possibilite que projetistas, que não possuem conhecimento na área de programação, desenvolvam projetos de automação industrial com foco em processamento de imagens. Para isso, a suíte de ferramentas foi proposta dentro do processo de uma empresa do Sul do Brasil, que desenvolve soluções completas em sistemas para inspeção visual automatizada. A partir desta suíte, é possível adicionar e configurar câmeras, definir o formato da aquisição de imagens, adicionar módulos de processamento de imagens e determinar a aprovação e reprovação de produtos em uma linha de produção. As ferramentas foram desenvolvidas na linguagem C# e utilizaram protocolos de comunicação como WCF e Modbus TCP. Através de uma avaliação realizada com os colaboradores da empresa, foi possível analisar que a criação da suíte de ferramentas corrobora para a criação de projetos, facilitando a sua criação e configuração de maneira rápida e fácil, se comparada com a forma anterior de desenvolvimento dos projetos. Além disso, constatou-se a necessidade de treinamento e uma constante atualização de funcionalidades, devido à complexidade e especificidades dos projetos. A principal contribuição deste trabalho foi alterar a forma como a empresa desenvolve os seus projetos, construindo assim um produto no qual profissionais da área de integração de sistemas poderão realizar os projetos de forma autônoma.

Palavras-chave: No-Code. Inspeção visual. C#. MODBUS. Automação.

A suite of tools to support the creation of automation visual inspection projects

ABSTRACT

Automated visual inspection systems implemented in the industry have many specificities that depend on the final product. Such characteristic makes their development difficult, making each new project to be coded from scratch. Currently, one must rely on programmers to code each new system. Based on this, the general objective of this work was to create a suite of tools that enable designers, who have no programming knowledge, to develop industrial automation projects with a focus on image processing. For this purpose, we propose a suite of tools within the process of a Southern Brazilian company, which develops complete solutions for automated visual inspection systems. From this suite, it is possible to add and configure cameras, set the image acquisition format, add image processing modules, and define product approval and rejection in a production line. The tools were developed using C# language and communication protocols such as WCF and Modbus TCP. Through an evaluation conducted with the company employees, it was possible to analyze that the creation of the suite of tools corroborates for the creation of projects, making its creation and configuration easy and quick, if compared to the previous form of development of the projects. In addition, we verified the need for training and a constant updating of functionalities, due to the complexity and specificities of the projects. The main contribution of this work was to change the way the company develops its projects, thus building a product in which system integration professionals are able to autonomously execute the projects.

Keywords: No-Code. Visual inspection. C#. MODBUS. Automation.

LISTA DE ABREVIATURAS E SIGLAS

IDE	Integrated Development Environment
XAML	Extensible Application Markup Language
WPF	Windows Presentation Foundation
WCF	Windows Communication Foundation
TCP	Transmission Control Protocol
JSON	JavaScript Object Notation
TPM	Total Productive Management
PDSFO	Programa para Dimensionar um Sistema Fotovoltaico
DLL	Dynamic-link library

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Observando o cenário atual, percebe-se que empresas da área da indústria tem investido cada vez mais em processos e linhas de produção na fabricação de seus produtos. Algumas funcionam de forma manual, o que indica que há seres humanos realizando cada etapa necessária, mas a grande maioria trabalha de forma totalmente automatizada com esteiras, robôs e outros mecanismos autônomos. Este avanço tem início no século XVIII, com a Revolução Industrial, período que propiciou o desenvolvimento de tecnologias que contribuíram para um aumento da produtividade e precisão na fabricação dos produtos (ROGGIA; FUENTES, 2016).

A evolução da fabricação que tem início no século XVIII e se estende até os dias atuais é marcada por quatro grandes revoluções industriais, conhecidas também como Indústria 1.0, 2.0, 3.0 e 4.0. A Primeira Revolução Industrial é caracterizada pela evolução do trabalho manual por pessoas e animais para a mecanização da produção através de máquinas e motores (SAKURAI; ZUCHI, 2018). Mais adiante, no início do século XX ocorre a Segunda Revolução, marcada pela produção em massa por intermédio da eletricidade e do petróleo. Posteriormente, a Terceira Revolução Industrial, conhecida também como Revolução Técnico-Científica e Informacional são marcadas pelos avanços no campo da informática, robótica e biotecnologia. Desta maneira, estabelecendo a Indústria 3.0 como a era dos computadores e da automação industrial (ALMEIDA; ANDRADE, 2023).

Ao longo dos anos com o contínuo investimento em desenvolvimento tecnológico surge a Quarta Revolução Industrial, a Indústria 4.0. Combinando produção física e operações de tecnologia digital inteligente, aprendizado de máquina e *big data*, a Indústria 4.0 apresenta alguns benefícios, como a redução de desperdícios, o controle preciso das informações ao longo de todo o processo, a customização de produtos em grande escala, a tomada de decisão mais ágil e precisa, entre outras muitas vantagens (SAKURAI; ZUCHI, 2018) (ANTONIO et al., 2018).

Embora a implantação da indústria 4.0 apresente alguns benefícios, esta ainda segue em curso e por isso é importante atentar para as desvantagens e/ou dificuldades que são encontradas à medida que é implantada. Como o processo é totalmente automatizado, tanto em linhas de produção manuais quanto em automatizadas é sempre difícil garantir que cada um destes processos seja executado com qualidade. Isto porque, existe na maioria das vezes problemas mecânicos, elétricos, falhas nos processos e entre outros, que

podem ocasionar falhas na produção e/ou produtos com defeitos.

Alguns destes problemas podem gerar uma falta de confiança nos usuários ou até mesmo problemas de segurança, nos casos de produtos que possam comprometer a integridade física de pessoas. Atualmente, com a alta competitividade do mercado, a satisfação do cliente é algo que precisa ser mantido. Por isso, uma empresa que não possui ferramentas de qualidade poderá apresentar produtos com defeitos em seus processos de fabricação, resultando em um gasto excessivo de correções e, em muitos casos, até dobrar o custo inicial de produção (CORAL; SELIG, 1994).

Da mesma forma que nos processos de fabricação, alguns processos de qualidade também contam com seres humanos que utilizam de uma inspeção visual para garantir que os produtos não sejam aprovados com falhas. A inspeção visual é um processo muito utilizado e tem como objetivo procurar por imperfeições ou características diferentes das esperadas em algum produto (TAYLOR; MELLOY; DHARWADA, 2004).

Mesmo contribuindo para a melhora da qualidade, o método de inspeção visual não é 100% confiável, visto que mesmo sendo executado por profissionais competentes, erros podem acontecer. Muitas destas inspeções podem falhar ou em muitos casos, devido à velocidade das linhas de produção, não serem possíveis de serem executados. Cenários como estes, em que a velocidade de produção é um fator de risco, se torna necessário automatizar a tarefa de inspeção visual para a indústria.

Muitas empresas oferecem soluções em processamento de imagens como forma de garantir a qualidade em processos de fabricação em linhas de produção. Importante ressaltar, que cada cliente final possui um produto diferente e com processos distintos, o que torna este tipo de projeto bastante específico. Fatores como o tipo de material, o ambiente, a iluminação e a velocidade de fabricação tornam o desenvolvimento da solução algo complexo e repleto de especificidades.

Com o avanço da tecnologia na área da visão computacional, muitos recursos de processamento de imagem começaram a ser utilizados na indústria como forma de mitigar falhas em produtos. A utilização de câmeras de alta resolução juntamente com sofisticados algoritmos têm contribuído para a melhoria de processos e controles de qualidade.

No Sul do Brasil existe uma empresa com mais de 25 anos de atuação, que aplica soluções completas em sistemas de inspeção visual automatizada na resolução de problemas em processos complexos da indústria. Com clientes de diversos seguimentos, cada sistema desenvolvido requer uma customização específica que atenda o funcionamento de sua linha de produção. Tal fator resulta em muitas horas de desenvolvimento focada

em algoritmos de processamento de imagem, *softwares* de interface com usuário e integração com a automação da empresa, utilizando-se principalmente de redes de campo industriais. Observa-se uma carência de processo no desenvolvimento dos projetos, não havendo padrões na criação dos códigos-fontes, controle de versionamento inexistente e o ressurgimento de *bugs* que já haviam sido resolvidos em projetos anteriores.

Como tentativa de resolver alguns destes problemas de desenvolvimento, inicialmente foi criado uma espécie de *Framework* para que fosse utilizado no início de cada projeto. O *Framework* é composto por um código-fonte base, com definições comuns aos projetos e com uma interface básica, este contribuiu para a melhoria do processo. Todos os projetos tinham a mesma base de código-fonte, necessitando apenas do algoritmo específico de processamento de imagem e de regras de negócio de acordo com o fluxo de operação. Durante algum tempo a solução foi satisfatória, facilitando a criação de projetos e reduzindo a repetição de erros antigos. No entanto, mesmo com este avanço, ainda era necessário a alocação de programadores desde a criação até a implantação dos projetos, resultando em uma impossibilidade da empresa assumir novos projetos.

Diante deste cenário, entendeu-se a necessidade da autonomia dos integradores de sistemas no desenvolvimento e implantação dos projetos. Dessa forma, o presente estudo tem por objetivo criar uma suíte de ferramentas que possa auxiliar e facilitar a construção, o desenvolvimento e a implantação dos sistemas. A suíte de ferramentas é composta por quatro sistemas que funcionaram de forma independentes, porém tem uma comunicação entre si, seja por arquivos ou por algum protocolo existente. Espera-se, com esta suíte de ferramentas, realizar a configuração do projeto, o desenvolvimento de inspeções visuais através de processamento de imagem, a execução do que foi configurado e a visualização dos resultados.

Através de uma avaliação realizada com os colaboradores da empresa, foi possível analisar que a criação da suíte de ferramentas corrobora para a criação de projetos, facilitando a sua criação e configuração de maneira rápida e fácil, se comparada com a forma anterior de desenvolvimento dos projetos. Ainda através desta pesquisa, constatou-se a necessidade de treinamento e uma constante atualização de funcionalidades, devido à complexidade e especificidades dos projetos.

A importância da autonomia dos integradores de sistemas no desenvolvimento e implantação dos projetos ocorre também pelo fato de que existe uma complexidade na área de programação. Entende-se que a suíte de ferramentas surge como uma opção *No-Code*, que vem se tornando uma alternativa cada vez mais popular e que facilita o

acesso às técnicas e às tecnologias de programação. Conforme a NoCode.Tech (NOCODE.TECH, 2022), a programação *No-Code* tem o objetivo de auxiliar na construção de software sem necessitar da escrita de código. Importante ressaltar, como aponta (GARCIA, 2023) em seu trabalho, que mesmo não necessitando de experiência em desenvolvimento de *software*, ainda assim é necessário ter um mínimo entendimento de como um sistema computacional funciona.

O restante do trabalho está organizado da seguinte forma: o capítulo 2 define e detalhas todas as tecnologias que fundamentaram a criação da suíte de ferramentas. Em seguida, o capítulo 3, descreve as ferramentas com os objetivos similares e existentes ao trabalho proposto e compara os resultados encontrados. O capítulo 4 descreve o desenvolvimento da suíte, detalhando as escolhas de implementação e as metodologias utilizadas durante a realização do trabalho. O capítulo 5 é a demonstração da utilização da suíte de ferramentas, exibindo funcionalidades e telas. No capítulo 6 encontra-se a descrição da avaliação realizada com os usuários acerca da utilização da suíte de ferramentas. Por fim, o capítulo 6 apresenta as conclusões do trabalho e sugestões de melhorias para uma possível continuação da suíte de ferramentas.

2 TECNOLOGIAS UTILIZADAS

Neste capítulo encontra-se a descrição das tecnologias utilizadas no desenvolvimento das ferramentas, trazendo os principais conceitos sobre cada tecnologia e sua utilização. Para a escolha das tecnologias foi levado em consideração o prévio conhecimento dos desenvolvedores da empresa e as licenças de *software* já adquiridas.

2.1 C# .NET

C# (lê-se "C Sharp") é uma linguagem de programação de alto nível, fortemente tipada, orientada a objetos e componentes que tem sua origem na família de linguagens C. Desenvolvida pela *Microsoft* o C# é a principal linguagem da plataforma .NET, podendo ser utilizada para desenvolver sistemas para plataformas Web, Desktop e dispositivos móveis. Por ser uma linguagem simples e com uma baixa curva de aprendizagem, tornou-se muito popular na área de desenvolvimento de *software* (SHARP; JAGGER, 2003).

Durante a criação da suíte de ferramentas, o C# foi a linguagem de programação utilizada para codificar todas as ferramentas propostas. Além de ser uma tecnologia intuitiva e relativamente fácil para desenvolver, tanto projetos mais simples, quanto projetos complexos e multiplataforma. Também é a linguagem que a empresa disponibiliza a licença de uso da IDE e, dessa forma, a equipe de desenvolvimento tem conhecimento.

2.2 Visual Studio

O *Visual Studio* é uma *IDE*, do inglês *Integrated Development Environment*, que permite a criação de *softwares* da plataforma .NET. Através dele é possível desenvolver todo o ciclo de desenvolvimento, como criar, editar, testar, depurar, controlar a versão e até mesmo implantar na nuvem. O *Visual Studio* é considerado a principal plataforma de desenvolvimento de *softwares* feitos em C# (MICROSOFT, 2023c).

O *Visual Studio* é a IDE utilizada durante o desenvolvimento da suíte, pois a equipe de desenvolvimento já possui conhecimento de uso em projetos anteriores. Com a IDE foi feito a codificação tanto do *back-end* quanto do *front-end* das ferramentas que englobam a suíte.

2.3 Windows Forms

Windows Forms é a principal estrutura de desenvolvimento de interfaces com o usuário para aplicações desktop para *Windows*. Fornecido juntamente com o *Visual Studio*, utiliza-se de uma visão de designer de telas, que é possível criar controles visuais através de funcionalidades de posicionamento, como, por exemplo, arrastar e soltar objetos. Tais recursos facilitam um rápido aprendizado na criação de interfaces gráficas, tornando a tarefa intuitiva e simples de ser implementada (MICROSOFT, 2023b).

Todos os projetos anteriores da empresa tiveram as suas interfaces desenvolvidas utilizando-se do *Windows Forms*. No desenvolvimento das ferramentas do Configurador e do Designer também foi adotada esta tecnologia para a criação de interfaces. O *Windows Forms* também foi uma escolha por apresentar um rápido desenvolvimento de telas complexas e uma fácil integração com o *back-end*.

2.4 XAML (WPF .NET)

Extensible Application Markup Language (XAML) é uma linguagem declarativa baseada no XML e é utilizada para desenvolver interfaces de usuário em aplicativos WPF (*Windows Presentation Foundation*). Com o auxílio de ferramentas como o *Visual Studio* esta pode funcionar de maneira similar ao *Windows Forms*, utilizando-se do recurso de clicar e arrastar controles de interface. Embora, o formato mais usual seja editar diretamente o arquivo de texto (MICROSOFT, 2023d).

Na criação da suíte de ferramentas, o WPF foi utilizado para criar a interface de usuário da ferramenta *Viewer*. A escolha por esta tecnologia ocorreu pelo fato de seu *design* ser mais moderno comparado com o *Windows Forms*. Como o objetivo da ferramenta *Viewer* é ser executada como um visualizador na própria linha de produção, apostou-se em uma interface mais moderna com relação ao Configurador e o *Designer*.

2.5 WCF

Windows Communication Foundation (WCF) é um conjunto de recursos utilizados na criação de aplicativos orientados a serviços. Criado pela *Microsoft*, o WCF facilita a comunicação entre *softwares* da plataforma .NET. A partir de um contrato de serviço que

define os dados e operações que serão consumidos, qualquer aplicação cliente pode implementar a comunicação e consumir tais dados de maneira fácil e rápida (MICROSOFT, 2023a).

O protocolo WCF foi utilizado neste projeto como forma de comunicação entre as ferramentas *Runtime* e *Viewer*. Após o *Runtime* receber as imagens e executar o processamento, ele envia os resultados da inspeção para o *Viewer* através do WCF. Após o recebimento, os resultados são exibidos para o usuário na interface do *Viewer*.

2.6 MODBUS TCP

Criado pela empresa *Modicom*, *Modbus* é um protocolo de comunicação muito utilizado por equipamentos industriais. Utilizando-se de um modelo cliente-servidor e um protocolo TCP, redes de comunicação são criadas de maneira fácil e rápida. Amplamente utilizado em processos industriais, o protocolo *Modbus* permite a comunicação e monitoramento de componentes críticos dos sistemas. Pelo fato do *Modbus* ser um protocolo aberto, qualquer dispositivo pode implementá-lo (MODICON INC., 1996).

Na suíte de ferramentas, o protocolo *Modbus TCP* é utilizado como forma de informar os resultados da inspeção para dispositivos necessários. Caso o sistema de visão precise acionar algum dispositivo nos casos de reprovação, como parar uma esteira ou algum mecanismo de expulsador de peças, isto é realizado através do protocolo *Modbus*. Quando configurado na ferramenta do Configurador, o *Runtime* logo após realizar o processamento, envia os sinais para os dispositivos cadastrados.

2.7 JSON

JavaScript Object Notation (JSON) é um padrão de troca de dados entre sistemas no formato de texto. Responsável por representar as informações e mais compacto que o XML, o formato JSON é amplamente utilizado em sistemas que necessitam a transmissão de grandes volumes de dados (ECMA-404, 2017).

Na criação das ferramentas da suíte, o JSON foi utilizado para definir o formato do arquivo de configuração que é gerado pelo Configurador. Todas as definições feitas no Configurador são serializadas em um arquivo de projeto que posteriormente será consumido e executado pelo *Runtime*.

2.8 SQLite

SQLite é uma biblioteca em linguagem C que implementa uma base de dados SQL. Através do *SQlite* é possível que aplicações consigam, de maneira fácil e rápida, a criação de um simples banco de dados que não existam muitos acessos e nem um grande volume de dados. Além disso, a utilização do *SQLite* cria um arquivo em disco com todos os dados e o acesso é feito diretamente sem a necessidade de um Sistema de Gerenciamento de Banco de Dados (SGBD) (SQLITE, 2022).

O banco de dados SQLite foi utilizado na suíte de ferramentas pela ferramenta *Runtime*. Seu objetivo é registrar em um banco local o resultado de cada uma das inspeções. Como uma linha de produção geralmente opera em uma velocidade alta, escolheu-se o *SQLite* pela sua simplicidade e fácil implantação.

2.9 Git

O *Git* é um sistema de controle de versão distribuída e amplamente utilizada por equipes de desenvolvimento de *software*. Criada em 2005 pelo também criador do *kernel* do sistema operacional *Linux*, Linus Torvalds, o *Git* é um projeto de código-aberto que trabalha com o conceito de repositórios, que cada diretório do *Git* é um repositório com histórico completo de alterações (GIT, 2022).

O Git é o sistema de controle de código-fonte e versionamento da suíte de ferramentas. Foi construído um repositório único contendo os projetos de todas as ferramentas, possibilitando o controle e a visualização de toda evolução da suíte.

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados seis estudos encontrados a partir da revisão bibliográfica realizada sobre a temática do presente estudo. Foram selecionados quatro artigos científicos, um trabalho de conclusão de curso e uma dissertação de mestrado. Primeiramente, são abordados os trabalhos que geraram o desenvolvimento de ferramentas que contribuíram para o desenvolvimento de algum processo industrial, a partir de uma dificuldade existente. Logo após, são apresentados os trabalhos que tiveram o objetivo de propor um estudo que avalie a qualidade de um *software* e por fim, um estudo sobre a criação de ferramentas com foco em inspeção visual.

3.1 Descrição dos Trabalhos

O estudo realizado por (BARBOSA; GASPAROTTO, 2015) resultou no desenvolvimento de uma ferramenta que auxilia o gerenciamento do processo de manutenção das máquinas e equipamentos do setor de fabricação de micro e pequenas empresas. Partindo da análise de desperdícios, retrabalhos, perda de tempo e de esforço humano, que ocorrem em manutenções preventivas, os autores desenvolveram uma ferramenta baseada no método TPM. Este método tem como objetivos elevar a autonomia dos colaboradores no processo, melhorar a confiabilidade e a eficiência dos equipamentos, melhorar e implementar segurança no ambiente de trabalho.

Outro estudo que abordou o desenvolvimento de uma ferramenta foi realizado por (MEDEIROS; JúNIOR, 2020). Partindo da indisponibilidade de ferramentas que possam modelar sistemas fotovoltaicos nas universidades, foi desenvolvido um *software* didático capaz de dimensionar um sistema fotovoltaico e a proteção dele. Chamado de PDSFO (Programa para Dimensionar um Sistema Fotovoltaico), a ferramenta calcula a geração do sistema solar com base em dados fornecidos pelo usuário da unidade consumidora, levando em consideração as irradiações da cidade do projeto.

No quesito qualidade de *software*, (NETO, 2020) realizou um estudo de caso sobre a implantação de processos de qualidade no desenvolvimento da empresa, buscando aumentar a confiabilidade das soluções desenvolvidas. O estudo foi realizado em uma *startup* em passos iniciais de existência e foi observado que os desenvolvedores não encontraram ou não registraram *bugs* durante o processo de desenvolvimento de *features* da aplicação, porém os usuários em ambiente de produção reportaram um alto número de

incidentes.

Ainda abordando a qualidade de *software*, (OLIVEIRA et al., 2012) propõem um instrumento para avaliar a qualidade dos serviços prestados por uma fábrica de *software*. Focado na terceirização do desenvolvimento de sistemas, este estudo analisa o processo de desenvolvimento de *software* em uma organização e constrói, com base na teoria, um instrumento para avaliar a qualidade dos serviços.

Explorando o processamento de imagem, (VILELA et al., 2008) desenvolveram um protótipo composto por uma esteira, uma câmera e uma ferramenta com algoritmos de visão computacional capaz de identificar qual peça está visível na esteira. O objetivo deste trabalho foi construir um protótipo que possibilitasse o ensino dos fundamentos de reconhecimento de objetos e a realização de pequenos experimentos envolvendo controle e automação. No estudo de (COSTA, 2021), foi desenvolvida uma *toolbox* com soluções configuráveis para resolução de alguns problemas de inspeção visual utilizando processamento de imagem. Este trabalho utilizou-se do *software HALCON*, fazendo uso dos recursos disponíveis nesta ferramenta e focando em algoritmos de visão computacional para resolver problemas específicos encontrados nas imagens obtidas.

3.2 Análise Comparativa

Realizando uma comparação dos trabalhos descritos nesta seção, conforme a Tabela 3.1. Em ambos os estudos que resultaram no desenvolvimento de ferramentas de auxílio, observou-se a escolha pela linguagem C# da *Microsoft* e o objetivo de tornar mais acessível uma tarefa complexa. Apesar de (BARBOSA; GASPAROTTO, 2015) não ter indicado os resultados de uso da ferramenta proposta em seu estudo, (MEDEIROS; JúNIOR, 2020) mostraram que a ferramenta desenvolvida alcançou a meta final, que era calcular com sucesso todos os parâmetros esperados. Ambos os projetos geraram versões iniciais com a possibilidade de adição de novas funcionalidades, conforme Tabela 3.2. Os trabalhos com foco na qualidade de desenvolvimento mostraram claramente uma preocupação com a forma de desenvolvimento de *software* que as empresas possuem. A necessidade de registrar e avaliar incidências encontradas para evitar que comportamentos não esperados cheguem no usuário final do sistema, como mostra a Tabela 3.3. Por fim, os trabalhos que abordaram a visão computacional demonstraram que a maioria dos problemas de processamento de imagens são muito específicos, necessitando algoritmos dedicados e diversos ensaios.

Tabela 3.1: Tabela com análise comparativa entre os estudos

Temáticas	Desenvolvimento	Qualidade	Processamento de Imagem
Vilela et al., 2008			x
Oliveira et al., 2012		x	
Barbosa; Gasparotto, 2015	x		
Medeiros; Júnior, 2020	x		
Neto, 2020		x	
Costa, 2021			x

Fonte: O Autor

Tabela 3.2: Tabela com análise comparativa entre os estudos de desenvolvimento

	Linguagem C#	Resultados	Adição Funcionalidades
Barbosa; Gasparotto, 2015	x		x
Medeiros; Júnior, 2020	x	x	x

Fonte: O Autor

Tabela 3.3: Tabela com análise comparativa entre os estudos de qualidade

	Preocupação no desenvolvimento	Registro de incidência
Oliveira et al., 2012	x	x
Neto, 2020	x	x

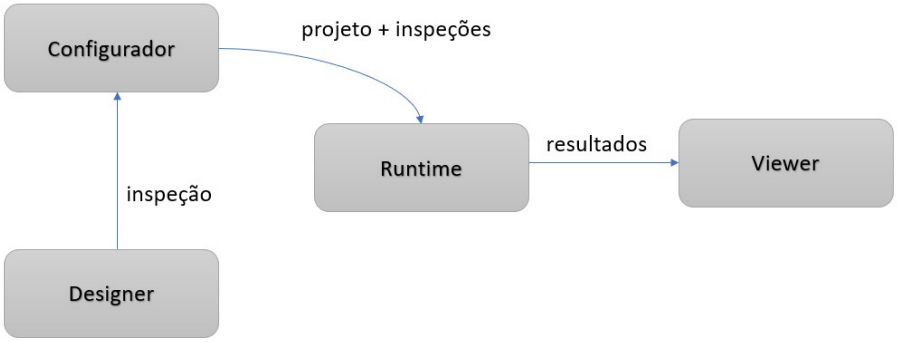
Fonte: O Autor

4 DESENVOLVIMENTO DA SUÍTE DE FERRAMENTAS

O presente capítulo tem o objetivo de apresentar e detalhar o processo de desenvolvimento das ferramentas que compõe a suíte. São quatro sistemas que funcionam de forma independente, porém têm uma comunicação entre si, seja por arquivos, ou por algum protocolo existente. A arquitetura da suíte está representada na Figura 4.1

Como as ferramentas possuem funcionalidades distintas, o público-alvo e o local que cada uma é executada também são diferentes. O Configurador e o Designer são sistemas que são executados exclusivamente por pessoas da empresa durante o desenvolvimento do projeto, rodando em seus próprios computadores. O *Runtime* tem um computador específico, pois é a parte mais crítica da arquitetura. Já o *Viewer* roda em um computador separado, servindo apenas como um visualizador de todo o sistema.

Figura 4.1: Arquitetura da suíte de ferramentas



Fonte: O Autor

Todas as ferramentas que fazem parte da suíte foram desenvolvidas utilizando o *Visual Studio* 2019 como *IDE*, sendo codificadas em C# e rodadas em um ambiente *Windows Desktop*. A escolha se deu devido ao prévio conhecimento da equipe de desenvolvedores e ao fato de que grande parte das linhas de produção da indústria possuem computadores dedicados somente para o controle das máquinas, não possuindo acesso à internet ou a qualquer outro tipo de rede. Incialmente, todas as *features* que foram implementadas em cada ferramenta foram escolhidas durante o próprio desenvolvimento da suíte, em reuniões semanais com a equipe de gerência.

4.1 Configurador

O desenvolvimento do Configurador teve como ponto de partida os elementos mínimos que são necessários para a criação de um projeto de automação com processa-

mento de imagem. Baseando-se em projetos já desenvolvidos pela empresa, percebeu-se que alguns elementos do sistema se repetiam, mantendo um padrão de arquitetura. Sendo assim, a ferramenta tem como objetivo a criação de um projeto, possibilitando ao usuário a adição e a configuração dos periféricos que serão utilizados, bem como, a definição de comportamentos em determinadas situações.

O *Template* utilizado na criação da ferramenta no *Visual Studio* foi do tipo *WindowsForms*, fornecendo a possibilidade de criação de formulários como interface gráfica de maneira rápida e simples. Para a parte de *front-end*, a arquitetura de desenvolvimento baseou-se inicialmente na criação de um objeto *TreeView* com todos os elementos do projeto, possibilitando ao usuário uma visão geral do sistema criado. Cada elemento da *TreeView* possui um formulário correspondente contendo os seus campos de configuração. A qualquer momento, o usuário pode optar por salvar as alterações realizadas, gerando assim a serialização dos campos em um arquivo que representa o projeto criado. O *back-end* foi desenvolvido de forma que os campos configurados na interface gráfica sejam um objeto interno chamado *RuntimeSettings* e que será serializado no formato *JSON*.

4.1.1 RuntimeSettings

O objeto interno *RuntimeSettings* representa toda a arquitetura do projeto de automação com processamento de imagem. Este objeto conta com todas as definições e parâmetros que devem ser carregados pela ferramenta que irá executar o projeto, conforme representado pela Figura 4.2 a seguir.

Figura 4.2: Definições do objeto *RuntimeSettings*

```
public List<CameraSettings> Cameras { get; set; } = new List<CameraSettings>();
public SequenceSettings Sequence { get; set; } = new SequenceSettings();
public List<ProcessorSettings> Processors { get; set; } = new List<ProcessorSettings>();
public List<ModelSettings> Models { get; set; } = new List<ModelSettings>();
public List<PartNumberParameter> PartNumberParameters { get; set; } = new List<PartNumberParameter>();
public List<MachineParameter> MachineParameters { get; set; } = new List<MachineParameter>();
public List<OutputParameter> OutputParameters { get; set; } = new List<OutputParameter>();
```

Fonte: O Autor

Os elementos de maior relevância que configuram e determinam como o sistema funciona, estão descritos abaixo:

- *CameraSettings*: Lista que define a quantidade de câmeras, modelo e parâmetros da imagem a ser recebida.

- *SequenceSettings*: Objeto que define a quantidade de imagens que cada câmera irá receber e que corresponde a uma peça inspecionada. Esse campo é necessário, pois em alguns sistemas uma peça inspecionada necessita de mais de uma foto para a realização da inspeção de processamento de imagem.
- *ProcessorSettings*: Lista que define a quantidade de *Processors* do projeto. Para cada câmera do projeto, existe um ou mais objetos *Processors* associados. Cada um destes objetos é como uma *Thread* que será responsável por executar o processamento da inspeção criada.
- *Models*: Lista que define as características comuns que determinadas peças da produção do cliente possuem. Em linhas de produção é muito comum as peças possuírem modelos distintos. Por isso, para o processamento de imagem, é importante a existência de parâmetros que ajudem na calibração da inspeção realizada.
- *PartNumerParameter*: Lista de códigos internos que representam peças reais de produção do cliente. Geralmente são códigos gerados pelo cliente e que estão atrelados a um modelo existente.
- *MachineParameter*: Lista com parâmetros importantes para o correto funcionamento do sistema. Dependendo da infraestrutura do sistema, alguns parâmetros de máquina são necessários.
- *OutputParameter*: Lista com dados que informam os resultados da inspeção de cada peça produzida. Nesta lista, devem conter todos os dados que o cliente final ou o operador deseja visualizar em tempo real no *Viewer*.

4.1.2 Arquivo de projeto

Toda a configuração realizada na interface gráfica do Configurador resulta no arquivo de projeto chamado *RuntimeSettings.json*. Para essa serialização foi utilizado um pacote *NuGet* chamado *Newtonsoft.json* versão 12.0.3. Com a utilização deste pacote é possível realizar tanto a serialização do objeto em arquivo, quanto a desserialização, no caso do usuário abrir projetos já desenvolvidos.

4.2 Designer

A ferramenta Designer foi desenvolvida de tal forma que funciona como uma interface de edição de arquivos de inspeção. Os arquivos de inspeção são arquivos de texto com um formato específico, que é interpretado pelo motor de processamento. Antes da criação da suíte, a maioria do desenvolvimento se dava através da edição manual destes arquivos. A ferramenta faz a leitura dos arquivos de módulos, que também são arquivos de texto, possibilitando a criação da receita e alterações dos parâmetros destes módulos. Cada parâmetro alterado realiza o salvamento desta informação nos arquivos, executa a chamada do motor de processamento e executa a leitura dos resultados e da imagem resultante.

4.3 Runtime

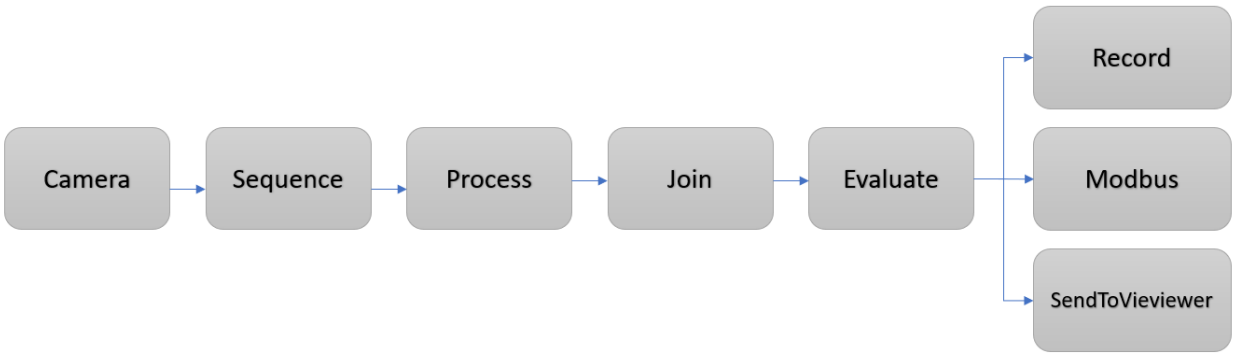
A Ferramenta *Runtime* é responsável por executar o projeto desenvolvido no Configurador. Diferentemente do Configurador, o *Template* utilizado no *Visual Studio* para a criação do *Runtime* foi o *ConsoleApplication*. Esta ferramenta não possui interface e roda pelo próprio terminal do *Windows*. Ao executar o *Runtime*, o usuário deve passar como parâmetro o caminho do arquivo de projeto criado pelo Configurador.

4.3.1 Rede de agentes

A arquitetura do *Runtime* é baseada em agentes. Cada agente funciona como uma *Thread* que possui uma fila de *Jobs*. Assim que a *Thread* percebe um novo *Job* em sua fila de entrada, esta retira da fila, executa, gera um resultado e repassa para o próximo agente. Assim que o arquivo de projeto é carregado, uma rede de agentes é construída com base nas definições deste arquivo. Na Figura 4.3, podemos observar uma ordem de ligação entre os agentes. Todo início se dá a partir do recebimento de imagens. Importante ressaltar, que a DLL de comunicação com o *Hardware* das câmeras foi desenvolvida em C++ e já existia.

- *Camera*: Para cada câmera do projeto existe um agente responsável por receber a imagem, carregá-la em memória e passá-la para o agente seguinte.
- *Sequence*: Agente único responsável por enumerar as imagens de cada peça inspe-

Figura 4.3: Arquitetura de agentes



Fonte: O Autor

cionada. Cada inspeção necessita de um número pré-definido de imagens de cada câmera que compõe o sistema. O sequenciamento realiza essa numeração para que o *Join* identifique se faltou alguma imagem da peça.

- *Process*: Para cada inspeção criada no Configurador e parametrizada pelo *Designer*, existe um agente *Process* responsável por executá-lo. Este agente é responsável por chamar o motor de processamento de imagens informando qual o arquivo de receita deve ser utilizado. Além disso, ao final do processamento ele deverá obter uma lista com os resultados obtidos. Importante ressaltar, que o motor de processamento de imagens é uma DLL que já existe, foi desenvolvida pela própria empresa e aqui será usada através de chamadas de processamento e leitura de resultados.
- *Join*: Este agente recebe todos os agentes Process que existem no projeto. Baseado em uma variável de *timeout*, o *Join* garante que todos os processamentos necessários da peça que está sendo produzida foram executados. Em alguns sistemas que possuem uma produção com esteiras, o resultado final da inspeção tem de ocorrer em tempo hábil de expulsá-la em casos de reprovação, portanto se por alguma razão algum processamento falhar ou demorar um longo tempo, este agente passa adiante a lista de *Process* recebidos e assinala um status de falha.
- *Evaluate*: Este agente é responsável por gerar o resultado final da peça produzida. Após o *Join* reunir todos *Process* referente a peça atual, ela os repassa para o *Evaluate* e este por sua vez analisa o resultado de cada *Process*, decidindo se a peça deve ser aprovada ou reprovada.
- *Modbus*: O agente *Modbus*, assim como o agente *Record*, também recebe como entrada o resultado da inspeção realizada. Caso o campo de endereço de dispositivo *Modbus* estiver configurado no projeto, um comando de aprovação ou reprovação

será enviado.

- SendToViewer: Agente responsável por enviar o resultado da inspeção e as imagens para o *Viewer*, através do protocolo WCF.
- *Record*: Agente que recebe como entrada o resultado da inspeção realizada e informações da peça. Gera uma estrutura chamada *ProductionData*, conforme Figura 4.4, e salva estas informações no banco de dados *SQLite*. A seguir, é detalhado cada um destes itens.
    - *Id*: Identificador do registro que contém o resultado da peça no banco de dados.
    - *PartNumberId*: Chave estrangeira para a tabela de *Partnumber* que contém o identificador do Partnumber da peça produzida.
    - *PartId*: Identificador da ordem da peça na linha na produção atual. Cada produção ao iniciar, zera sua contagem e, a cada peça inspecionada, incrementa o partId.
    - *InspectionTime*: Registro de data e hora da produção da peça inspecionada.
    - *Result*: Resultado da peça inspecionada. Cada peça produzida possui um resultado final, sendo o valor 0 para reprovada ou 1 para aprovada.

Figura 4.4: Estrutura ProductionData

```
public class ProductionData
{
    public long Id { get; set; }
    public long PartNumberId { get; set; }
    public int PartId { get; set; }
    public DateTime InspectionTime { get; set; }
    public int Result { get; set; }
}
```

Fonte: O Autor

4.3.2 Estados de operação

A rede de agentes existente no *Runtime* possui um estado de funcionamento. Ao executar o projeto, o *Runtime* realiza toda a configuração de sua rede, no entanto aguarda um comando de *Start* para realizar a conexão com as câmeras, disparar os agentes e aguardar as imagens. A Figura 4.5 exibe os possíveis estados da rede.

Figura 4.5: Estados de operação do *Runtime*

```
/// <summary>
/// Estrutura que representa o estado do Runtime.
/// </summary>
public enum State
{
    STOPPED,
    STARTING,
    RUNNING,
    STOPPING
}
```

Fonte: O Autor

- *STOPPED*: Rede de agentes parada, sem conexão com as câmeras e agentes parados.
- *STARTING*: Realizando conexões com as câmeras e disparando os agentes.
- *RUNNING*: Todas as conexões realizadas e todos agentes rodando. Aguardando as imagens das câmeras.
- *STOPPING*: Enviando comando de *Stop* para todas as câmeras e parando os agentes.

Os comandos que setam o estado do *Runtime* para *STARTING* e *STOPPING* são recebidos do *Viewer*, quando o usuário iniciar ou parar a produção.

4.3.3 Banco de dados

O *Runtime* é a ferramenta responsável por criar e realizar a inserção de itens no banco de dados. Na primeira execução do projeto, o *Runtime* cria localmente um banco de dados *SQLite*. Optou-se por esse tipo de banco, pois ele é usado somente para salvar os dados de produção e os modelos criados pelo usuário. Este é de fácil criação e não necessita de instalações adicionais.

4.3.4 Comunição WCF

O *Runtime* possui duas comunicações via protocolo WCF. Uma comunicação Cliente que possibilita enviar para o *Viewer* o resultado da inspeção juntamente com as imagens. A outra comunicação em que o *Runtime* funciona como servidor, recebendo do *Viewer* os comandos de *Start* e *Stop*. Além de receber estes dois comandos, o *Viewer*

pode também ler o status da rede de agentes do *Runtime*, pois existe a possibilidade do *Runtime* já estar em funcionamento quando o *Viewer* for aberto. Isto pode acontecer nos casos em que o computador que o *Viewer* estiver rodando necessite ser reinicializando ou o programa fechado acidentalmente. Desta forma, a comunicação entre *Runtime* e *Viewer* é restabelecida normalmente.

4.4 Viewer

A ferramenta *Viewer* foi a única que teve o seu desenvolvimento de forma diferente da especificada inicialmente. Com a evolução do desenvolvimento da suíte, gerou-se uma urgência em sua utilização, portanto decidiu-se por desenvolvê-la utilizando o próprio *WindowsForms* e não o WPF.

Ao inicializar, o *Viewer* realiza as conexões necessárias com o *Runtime* para iniciar a comunicação entre ambos. Após a comunicação ser estabelecida, o *Viewer* realiza a leitura do estado de operação do *Runtime* e o exibe na tela. Caso este estado seja diferente de *Running*, ele habilita um botão para o usuário poder iniciar o sistema. Em caso contrário, ele só estabelece a conexão para poder receber os resultados.

Com o *Runtime* em execução, assim que uma peça for produzida e o seu resultado for enviado para o Viewer, este por sua vez apenas exibe as imagens e o resultado final da inspeção na tela. Este processo é repetido a cada nova peça, sempre mantendo atualizado as imagens e resultado com a última peça produzida.

5 APLICAÇÃO DA SUÍTE DE FERRAMENTAS

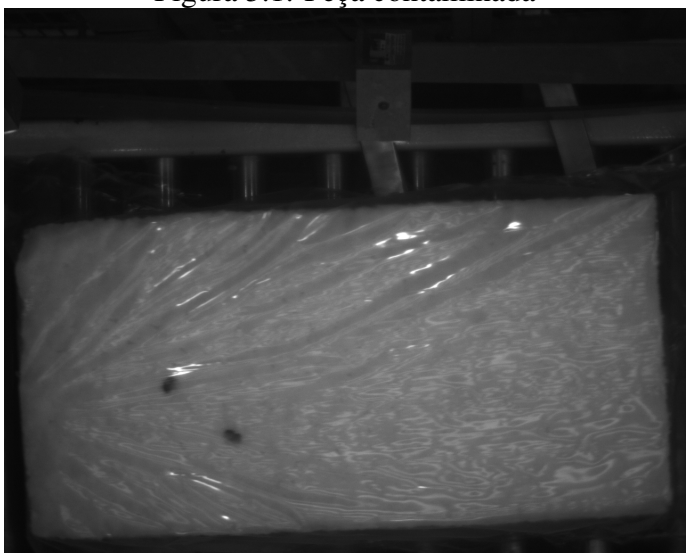
Este capítulo tem como objetivo demonstrar a utilização da suíte de ferramentas em um projeto. Inicia-se com a definição do projeto na Seção 5.1 e em seguida a criação do projeto com o Configurador e inspeção com o *Designer* na Seção 5.2. Após, na seção 5.3, está descrito a execução do *Runtime*, finalizando com o *Viewer* na Seção 5.4.

Para fins de execução e demonstração de todo o sistema em funcionamento, foi desenvolvido um exemplo que tem uma câmera que identifica contaminantes em um produto, assim simulando como as ferramentas se comunicam na prática. As imagens aqui utilizadas correspondem a um projeto já realizado pela empresa, porém não com a suíte de ferramentas.

5.1 Definição de um projeto de automação com inspeção visual

O projeto é baseado em uma empresa do ramo petroquímico que produz uma espuma de dimensões conhecidas. Devido ao formato do seu processo, o produto em questão sofre com a contaminação de algumas impurezas, assim gerando rejeitos na linha de produção. Foi realizado um sistema de automação com processamento de imagem, que utiliza uma câmera que, ao receber um sinal elétrico oriundo de um sensor, realiza uma captura superior da peça e roda um processamento que visa identificar se existem contaminantes presentes. Caso seja identificado a contaminação, um sinal é enviado para o acionamento de um expulsador que rejeita a peça. As Figuras 5.1 e 5.2 representam uma peça reprovada e uma peça aprovada, respectivamente.

Figura 5.1: Peça contaminada



Fonte: O Autor

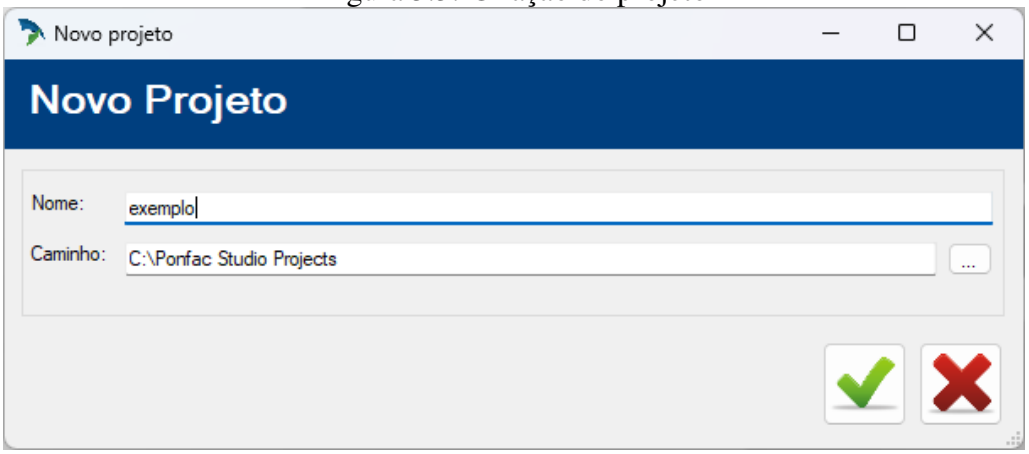
Figura 5.2: Peça sem contaminação

Fonte: O Autor

5.2 Criando o projeto com o Configurador e inspeção com o Designer

Primeiramente, devemos criar um projeto no Configurador. A Figura 5.3 exibe a tela de criação de um projeto.

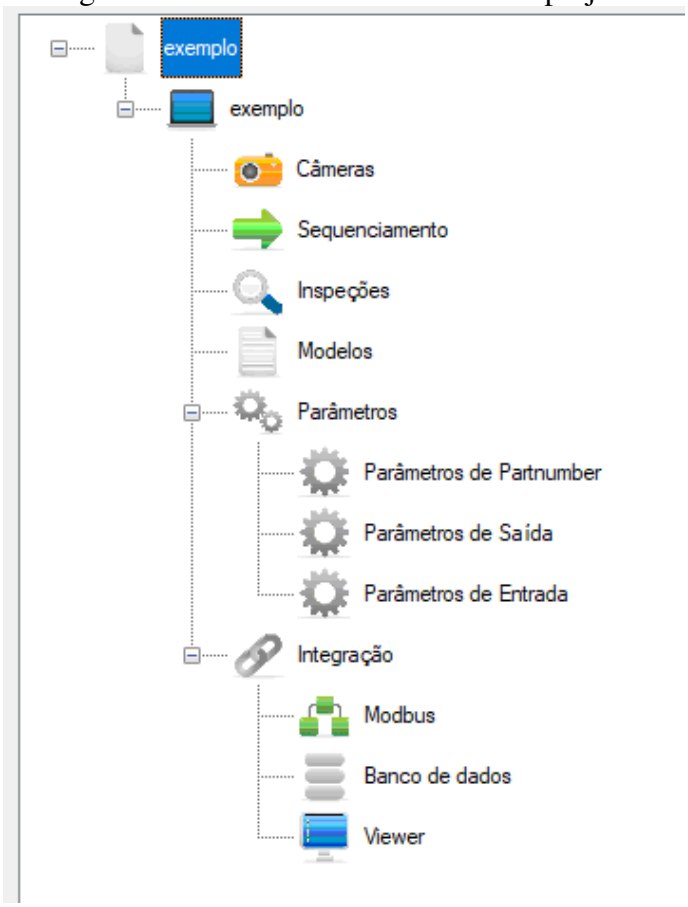
Figura 5.3: Criação do projeto



Fonte: O Autor

Após a criação, temos uma tela principal com uma árvore de itens à esquerda que demonstra a arquitetura do projeto. Na Figura 5.4 pode-se verificar a estrutura. A ordem de apresentação dos itens está disposta apenas para demonstrar o fluxo de operação do projeto, porém não existe uma obrigatoriedade de ser configurado nesta sequência.

Figura 5.4: Árvore com a estrutura do projeto

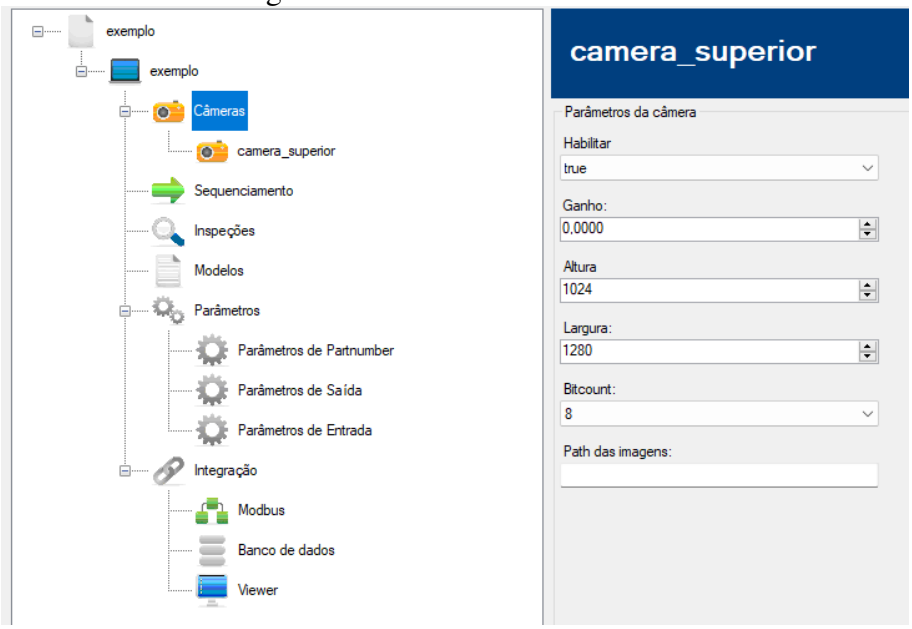


Fonte: O Autor

5.2.1 Câmera

O primeiro item a ser configurado é a câmera. Nesse exemplo, temos somente uma câmera superior, mas poderíamos ter seis câmeras, replicando a mesma inspeção em todas as faces da peça. A câmera utilizada neste trabalho é de simulação, deste modo, as imagens são carregadas do próprio computador. Ao adicionar a câmera, os principais parâmetros que devem ser configurados são a largura, a altura e o *bitcount*. A Figura 5.5 exibe a tela de configuração da câmera.

Figura 5.5: Parâmetros de câmera



Fonte: O Autor

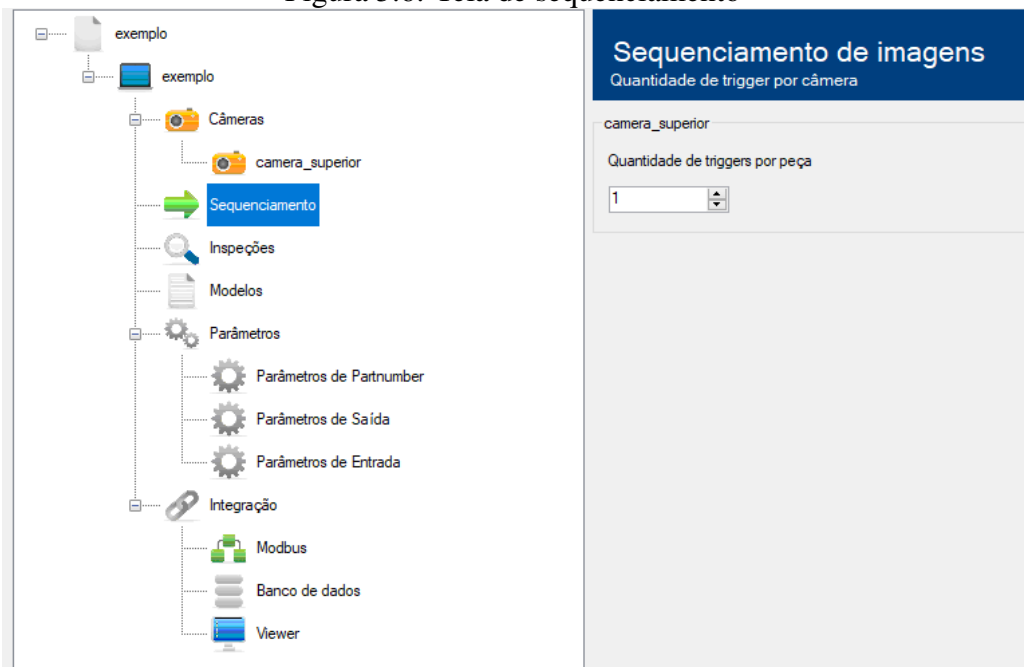
- Altura: Quantidade de *pixels* da altura da imagem.
- Largura: Quantidade de *pixels* da largura da imagem.
- Bitcount: Intensidade de *bits* da imagem.

Para o projeto utilizado, a imagem possui uma altura de 1024px, largura de 1280px e um *bitcount* de 8.

5.2.2 Sequenciamento

Para o projeto em questão, a captura de imagens se dá através de um *trigger* oriundo de um sinal elétrico de um sensor. Como será capturado apenas uma imagem para cada peça, deixamos a configuração como 1, conforme indicado na Figura 5.6.

Figura 5.6: Tela de sequenciamento

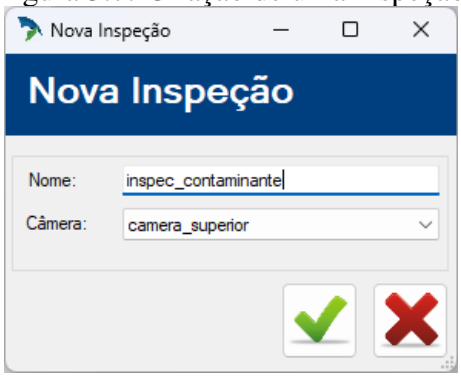


Fonte: O Autor

5.2.3 Inspeções

Como as inspeções são todas desenvolvidas e parametrizáveis pelo *Designer*, o único papel do Configurador é a criação do objeto na *Treeview*, a criação de um arquivo de receita no diretório do projeto e a chamada para execução do *Designer* passando o nome deste arquivo como parâmetro. A Figura 5.7 exibe a adição de uma inspeção. É necessário escolher a câmera do projeto. O arquivo de receita é apenas um arquivo vazio com extensão .ini e que será interpretado pelo motor de processamento.

Figura 5.7: Criação de uma inspeção

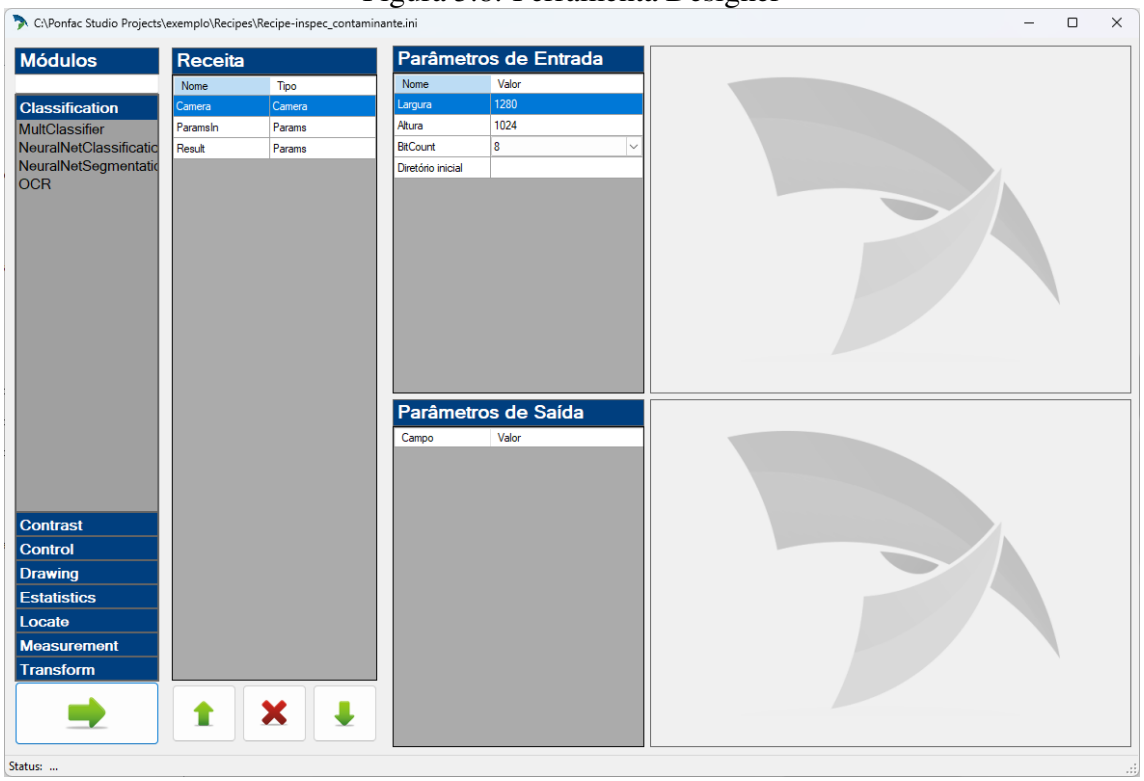


Fonte: O Autor

Ao criar a inspeção, automaticamente o Designer abre, como mostra a Figura 5.8. Esta ferramenta possui alguns campos importantes:

- Módulos: São ferramentas de processamento de imagem. Algoritmos desenvolvidos por uma equipe especialista em processamento de imagens.
- Receita: Receita é um arquivo que estão os módulos que o usuário utiliza em sua inspeção. De maneira geral, é o projeto de inspeção propriamente dito.
- Parâmetros de entrada: Cada módulo selecionado na área de receita, terá seus parâmetros carregados nos parâmetros de entrada. Desta forma, o usuário pode verificar e ajustar de maneira prática.
- Parâmetros de saída: Parâmetros de resultado do módulo selecionado. Cada alteração em algum parâmetro de entrada, irá gerar novos resultados no processamento. Portanto, o *Designer* funciona como um simulador do algoritmo de processamento criado. É possível escolher os módulos, editar seus parâmetros e verificar o efeito na imagem de resultado.

Figura 5.8: Ferramenta Designer

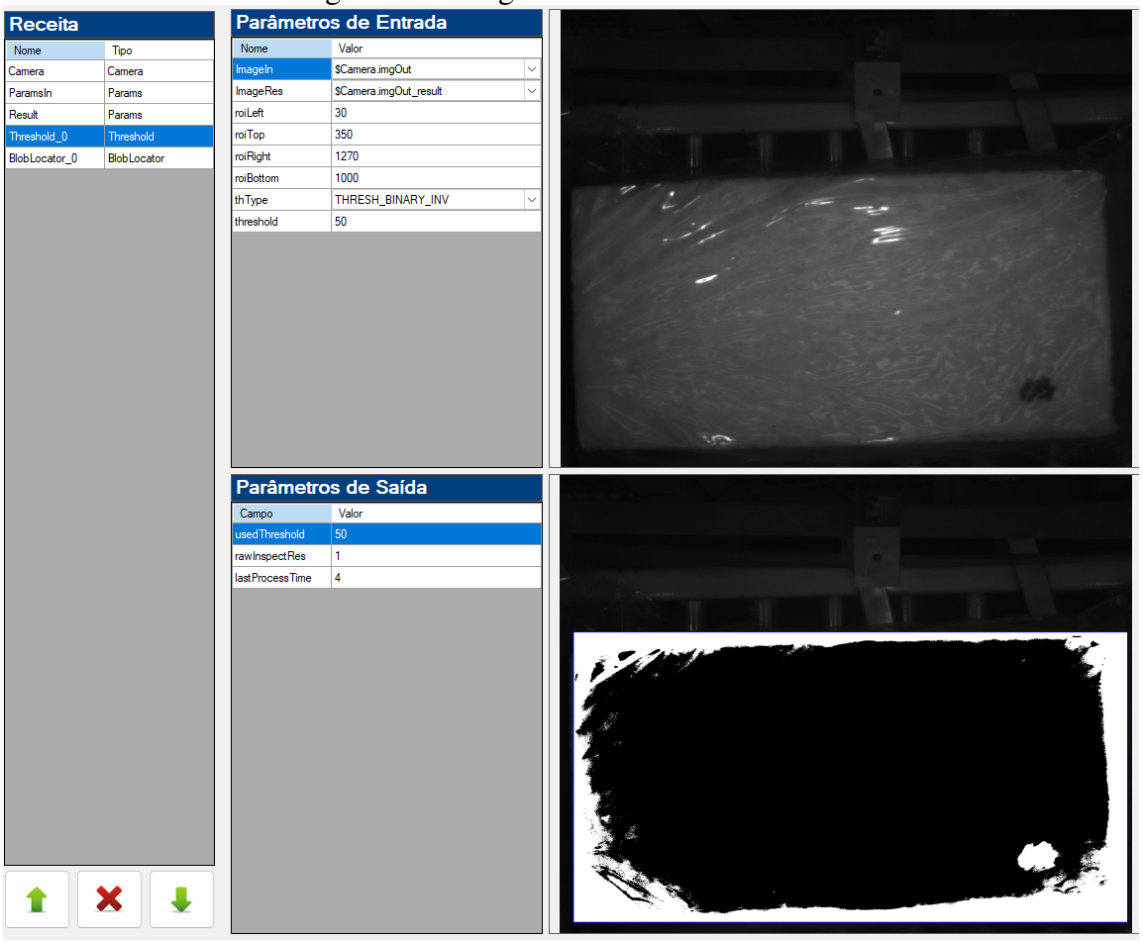


Fonte: O Autor

Além dos itens acima, existem duas áreas de imagens à direita da tela do *Designer*. Na parte superior, será exibido a imagem bruta vinda da câmera e abaixo, terá a imagem

resultante da execução da receita criada. No exemplo, o objetivo é identificar os contaminantes na peça, para isso será feito uma simples inspeção. Primeiramente, adicionamos na receita um módulo do tipo *Threshold* para realizar uma limiarização na imagem recebida da câmera. A Figura 5.9 mostra o resultado obtido após a execução deste módulo.

Figura 5.9: Imagem da câmera limiarizada



Fonte: O Autor

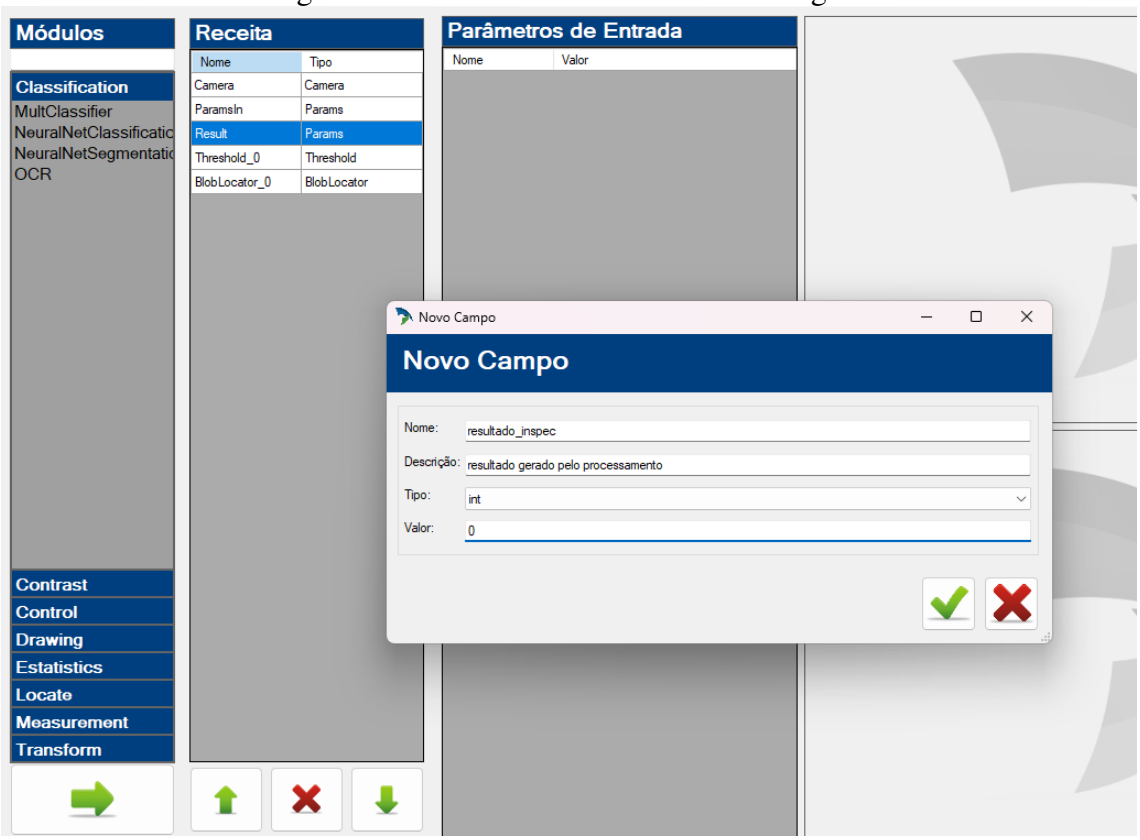
Conforme visto acima, que após a limiarização o defeito ficou mais aparente. Agora, basta adicionarmos um módulo de detecção de *blobs*. A Figura 5.10 mostra o resultado final da execução da receita.

Figura 5.10: Imagem da detecção do contaminante

Fonte: O Autor

Após a receita estar pronta, ainda é necessário disponibilizar o resultado final da inspeção como um parâmetro de saída da receita, para que o *Runtime* possa enviá-la via *Modbus* para acionamento de algum expulsador e também para que o usuário consiga visualizar este resultado no *Viewer*. A Figura 5.11 exibe a adição deste parâmetro. Foi configurado com um valor inicial de 0, que é interpretado como Reprovação, desta forma se houver algum tipo de falha no recebimento do resultado, a peça produzida será reprovada e expulsa da linha por segurança.

Figura 5.11: Variável de resultado no Designer

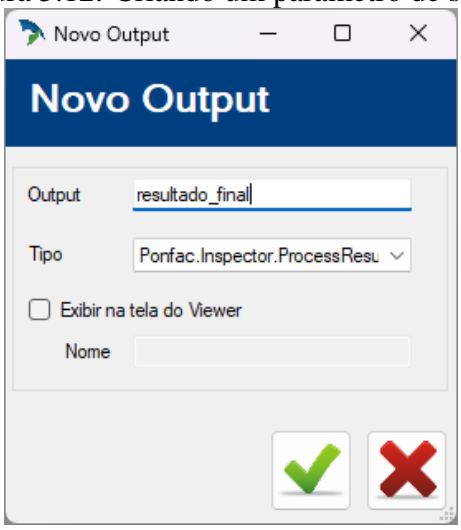


Fonte: O Autor

5.2.4 Parâmetros de Saída

Neste exemplo, foi criado apenas um parâmetro de saída que é utilizado para receber o resultado final da inspeção do motor de processamento. Já existe uma variável criada no *Designer* que possui este resultado, porém é necessário realizar um mapeamento, pois as variáveis no *Designer* são apenas utilizadas na receita e, portanto, só existem dentro do motor. Este mapeamento será feito na parte de Modelos. A Figura 5.12 mostra a variável sendo criada.

Figura 5.12: Criando um parâmetro de saída

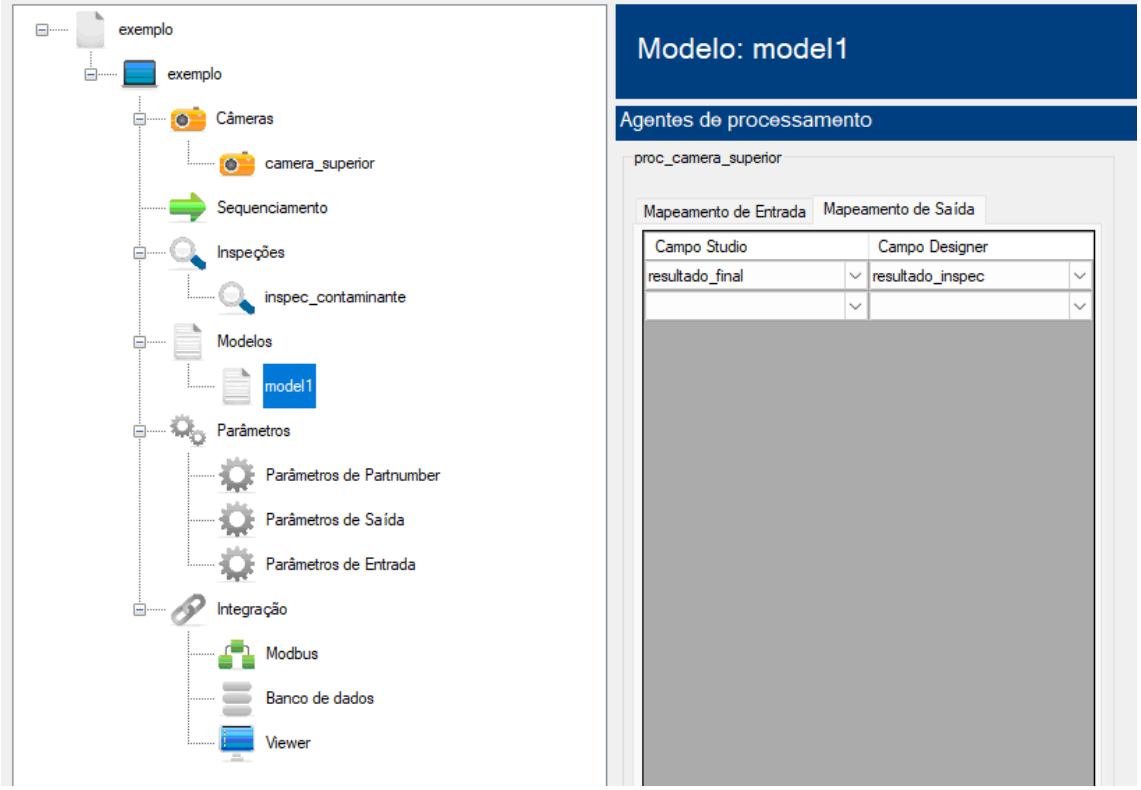


Fonte: O Autor

5.2.5 Modelos

No exemplo, criou-se um modelo apenas para realizar um mapeamento entre variáveis. Como a Figura 5.13 exibe, é criado um mapeamento de saída que relaciona a variável de resultado criada no *Designer* com a criada no Configurador. Será através desse mapeamento que é possível obter o resultado da inspeção do motor de processamento.

Figura 5.13: Mapeamento entre Designer e Configurador

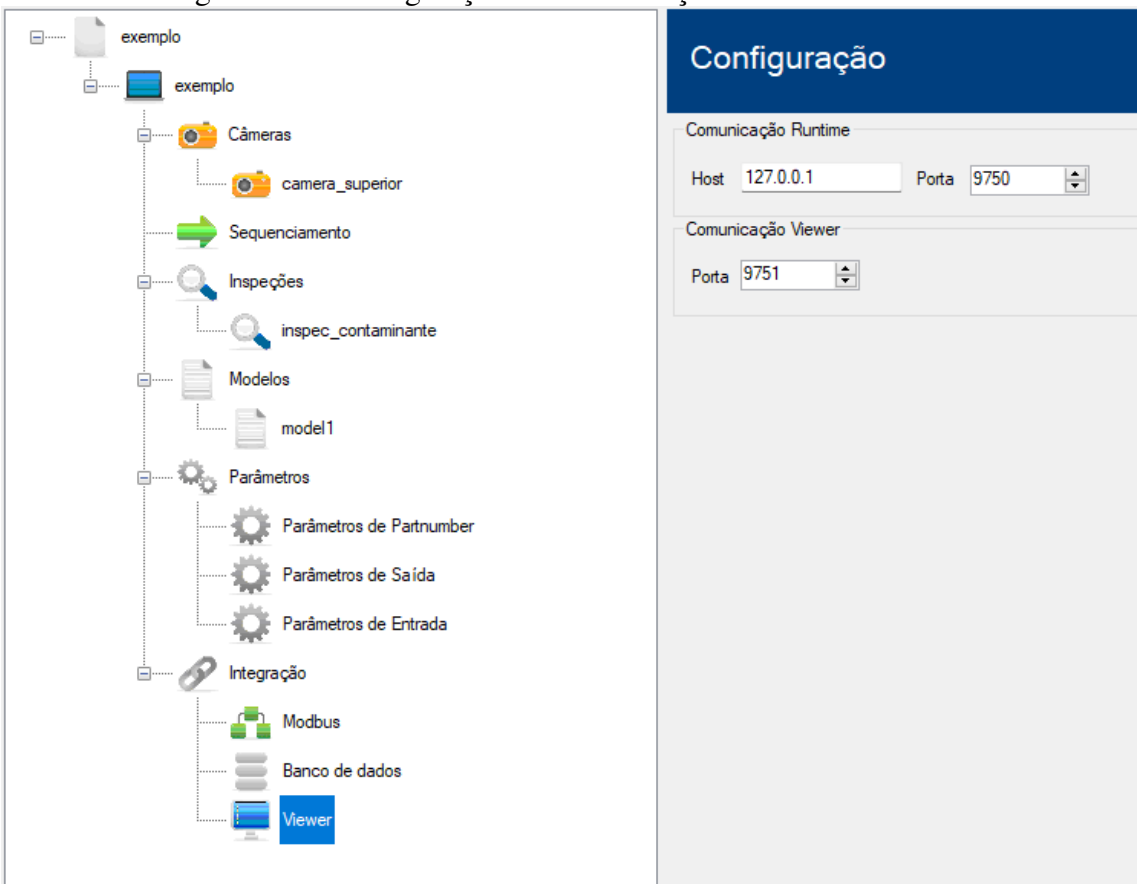


Fonte: O Autor

5.2.6 Comunicação com Viewer

Para a comunicação do *Runtime* com o Viewer, tanto para o envio dos resultados, quanto para o recebimento de comandos de *Start* e *Stop*, deve ser realizado a configuração dos parâmetros que definem a comunicação. A Figura 5.14 exibe estes parâmetros.

Figura 5.14: Configuração de comunicação *Runtime<->Viewer*



Fonte: O Autor

5.3 Executando o Runtime

Para rodar o *Runtime*, deve-se executá-lo informando qual o arquivo de projeto ele deverá carregar. O caminho e nome do arquivo se dão através do argumento *–JsonPath*. A Figura 5.15 exibe a chamada de execução.

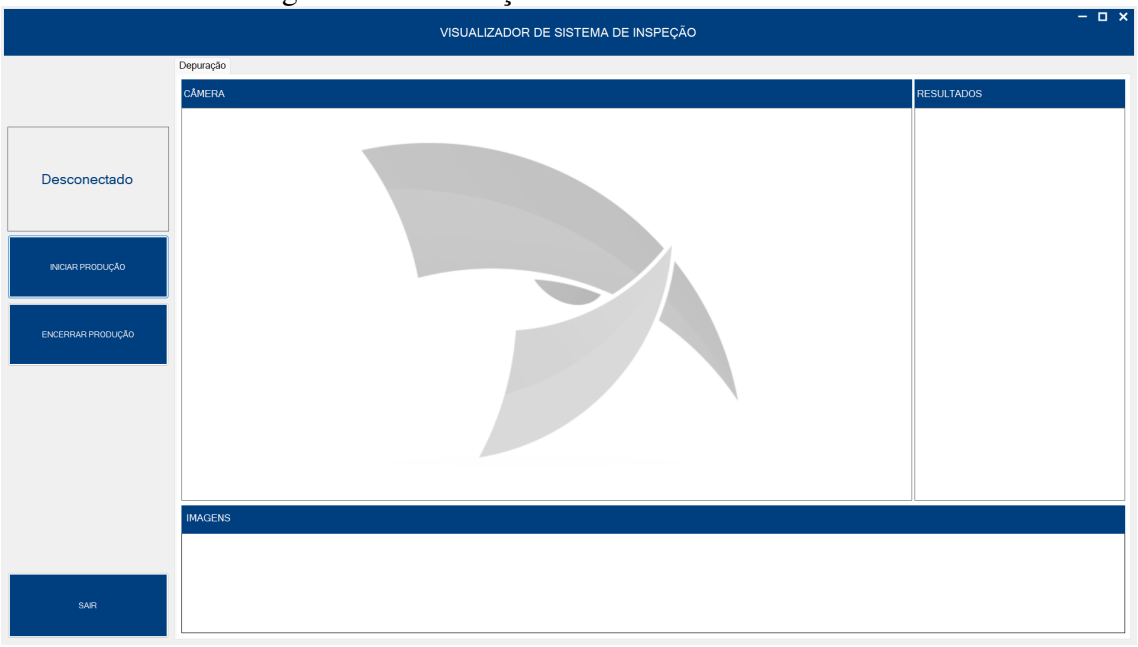
Figura 5.15: Executando o Runtime

Fonte: O Autor

5.4 Viewer

Por fim, utilizou-se o Viewer para enviar os comandos de *Start* e *Stop* para o *Runtime* e visualizar as inspeções e os seus resultados. Ao iniciar a ferramenta, percebe-se o status como "Desconectado", isto ocorre porque não foi possível estabelecer conexão com o *Runtime*. A Figura 5.16 exibe este cenário.

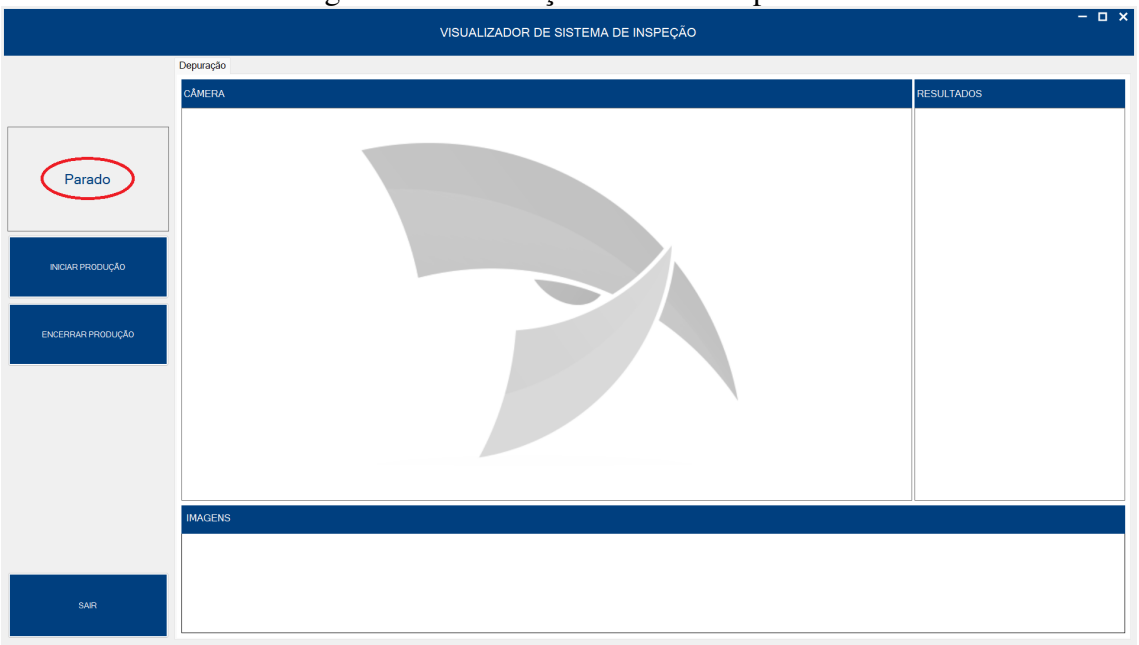
Figura 5.16: Execução do Viewer - desconectado



Fonte: O Autor

Após iniciar o *Runtime*, pode-se identificar que o estado informado na tela já atualizou para "Parado", conforme a Figura 5.17, pois a conexão entre *Runtime* e *Viewer* foi estabelecida com sucesso. Neste momento, O *Runtime* ainda está parado, aguardando um comando de *Start*.

Figura 5.17: Execução do Viewer - parado

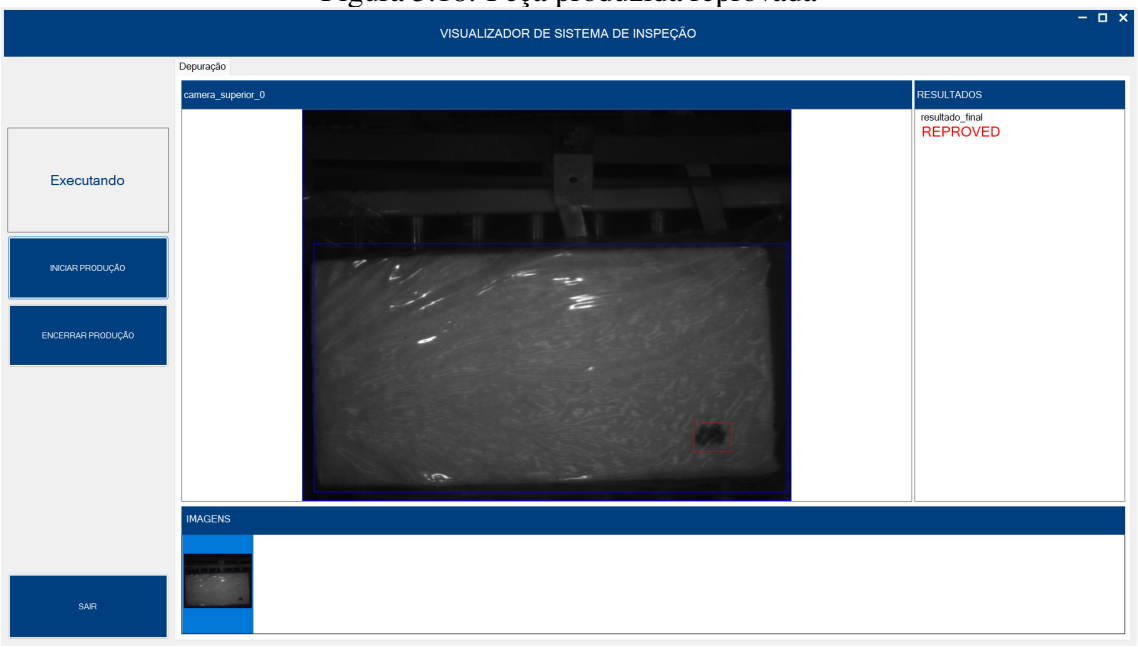


Fonte: O Autor

Assim que o usuário clicar no botão de "Iniciar Produção", o comando de *Start* é enviado e, se não ocorrer nenhuma falha na inicialização do *Runtime*, o estado na tela é atualizado para "Executando". Neste momento, o sistema está pronto para produção, aguardando que as peças sejam posicionadas na linha e que a captura se inicie. Assim que o *Runtime* receber da câmera a imagem da peça atual, todo o fluxo do projeto acontece.

No exemplo utilizado, realizou-se a simulação de duas peças, uma devendo ser reprovada, devido ao contaminante presente e outra peça devendo ser aprovada, pois não consta contaminante. A Figura 5.18 e Figura 5.19 representam, respectivamente, o resultado dessa simulação.

Figura 5.18: Peça produzida reprovada



Fonte: O Autor

Figura 5.19: Peça produzida aprovada

Fonte: O Autor

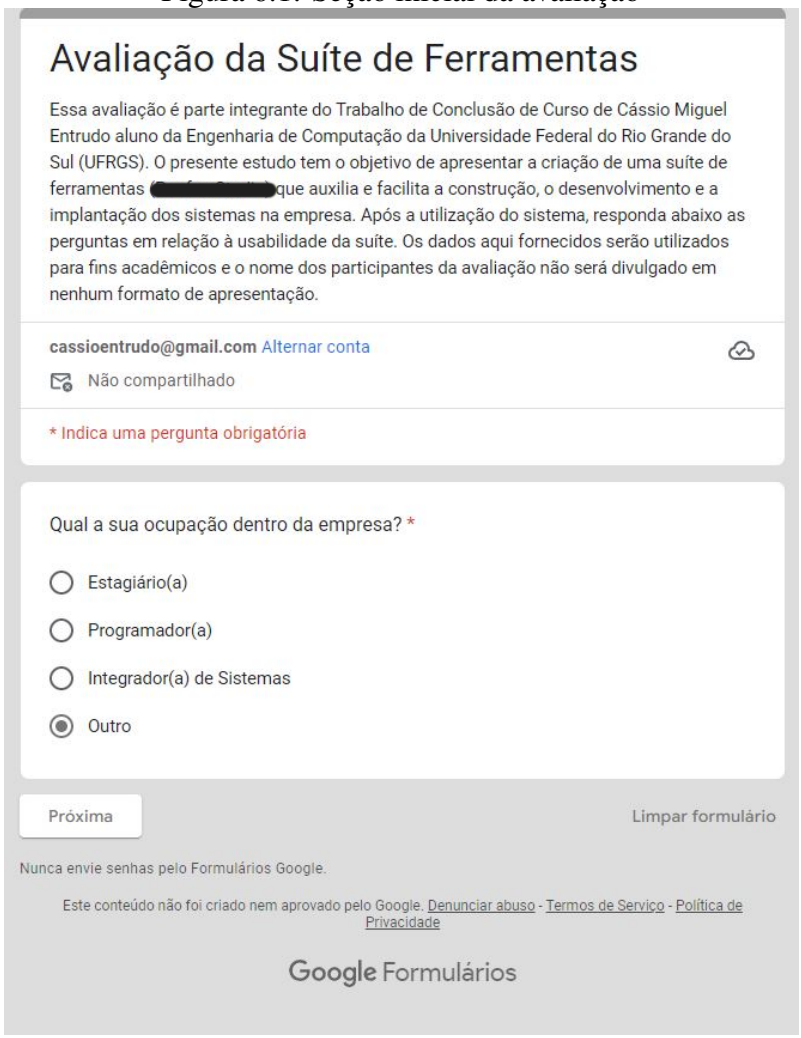
6 AVALIAÇÃO COM USUÁRIOS

Este capítulo tem o objetivo de descrever como foi realizada a avaliação do uso da suíte de ferramentas com os usuários. A avaliação foi realizada a partir de um formulário do *Google*, enviada para dez colaboradores da empresa. Nas seções a seguir, encontram-se descrito o modelo da avaliação e quais foram os resultados encontrados. Importante ressaltar que, antes da aplicação da pesquisa, a suíte de ferramentas foi apresentada em uma reunião e teve seu funcionamento demonstrado, o que possibilitou aos colaboradores um conhecimento geral da arquitetura da solução e de como as ferramentas funcionam. Após a demonstração, os colaboradores fizeram o uso da suíte em um novo projeto que estava sendo desenvolvido pela empresa. Além disso, juntamente com a gerência, criou-se um nome interno para cada uma das ferramentas. Por esta razão, as imagens aqui colocadas tiveram esta informação ofuscada.

6.1 Formulário de Pesquisa

A primeira seção da avaliação apresenta os dados do aluno pesquisador e o objetivo do trabalho, seguido do questionamento acerca do cargo do colaborador que está respondendo a pesquisa, conforme Figura 6.1.

Figura 6.1: Seção inicial da avaliação



Fonte: O Autor

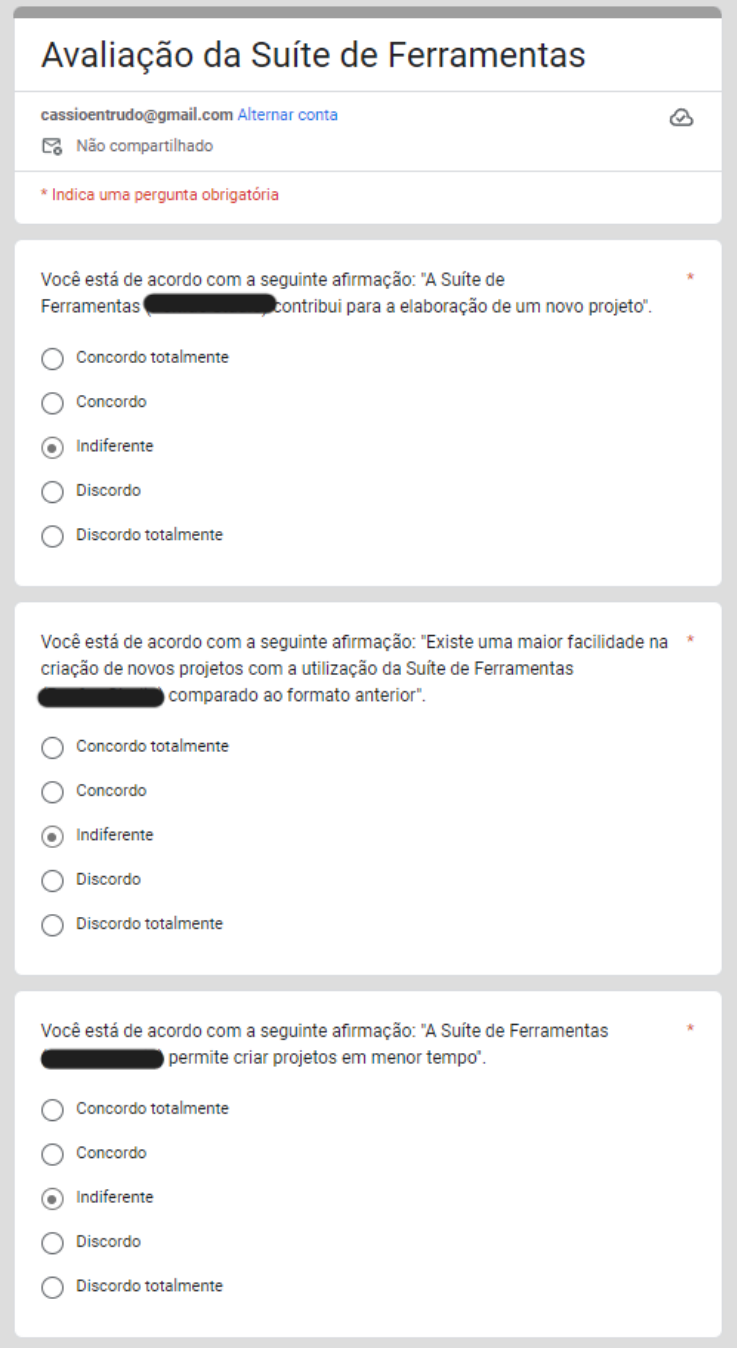
Na seção dois da pesquisa é questionado ao colaborador o tempo em que trabalha na empresa, conforme mostra a Figura 6.2.

Figura 6.2: Seção de tempo de empresa

Fonte: O Autor

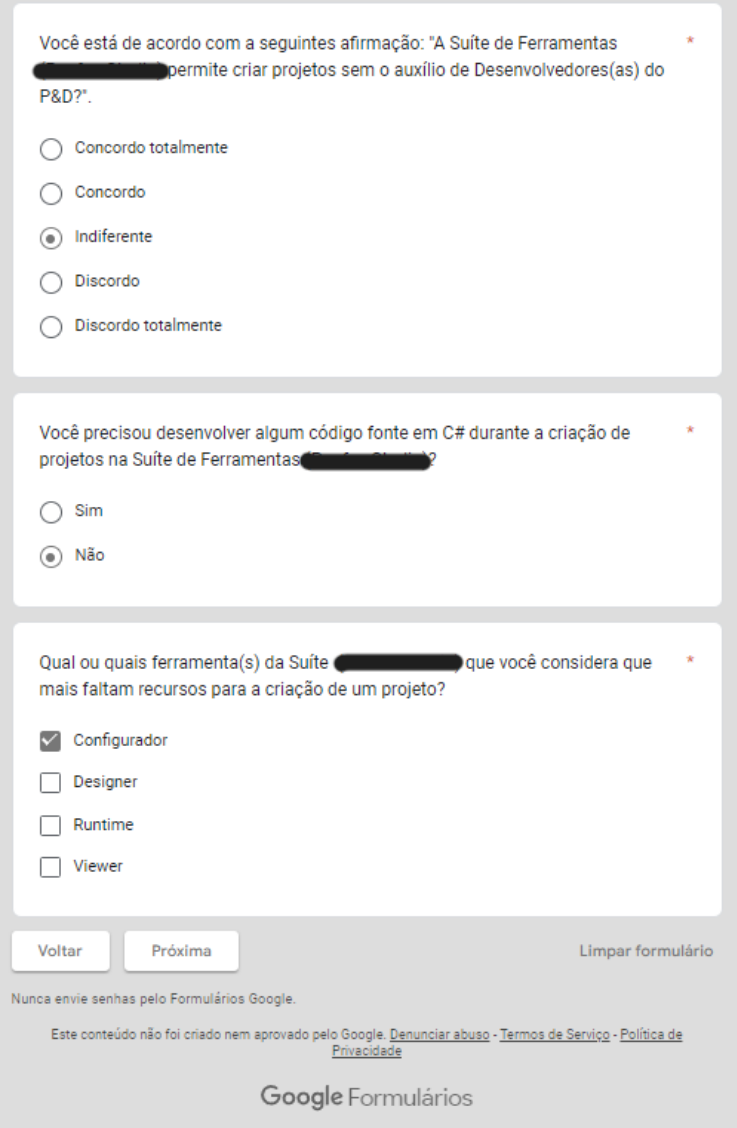
Na seção três do formulário constam as principais perguntas que avaliam a suíte de ferramentas, conforme apresentado na Figura 6.3 e na Figura 6.4.

Figura 6.3: Seção de avaliação da suíte de ferramentas parte 01



Fonte: O Autor

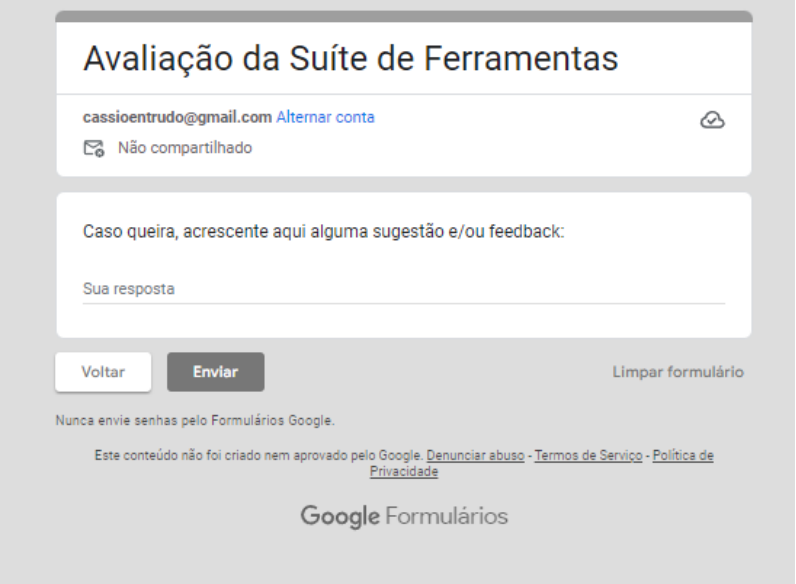
Figura 6.4: Seção de avaliação da suíte de ferramentas parte 02



Fonte: O Autor

Na última seção da pesquisa de avaliação da suíte de ferramentas foi incluído uma pergunta qualitativa para que o colaborador tivesse a oportunidade de acrescentar aquilo que não foi contemplado nas perguntas anteriores, bem como, um espaço para um feedback sobre as ferramentas, como mostrar a Figura 6.5

Figura 6.5: Seção de sugestão e feedback

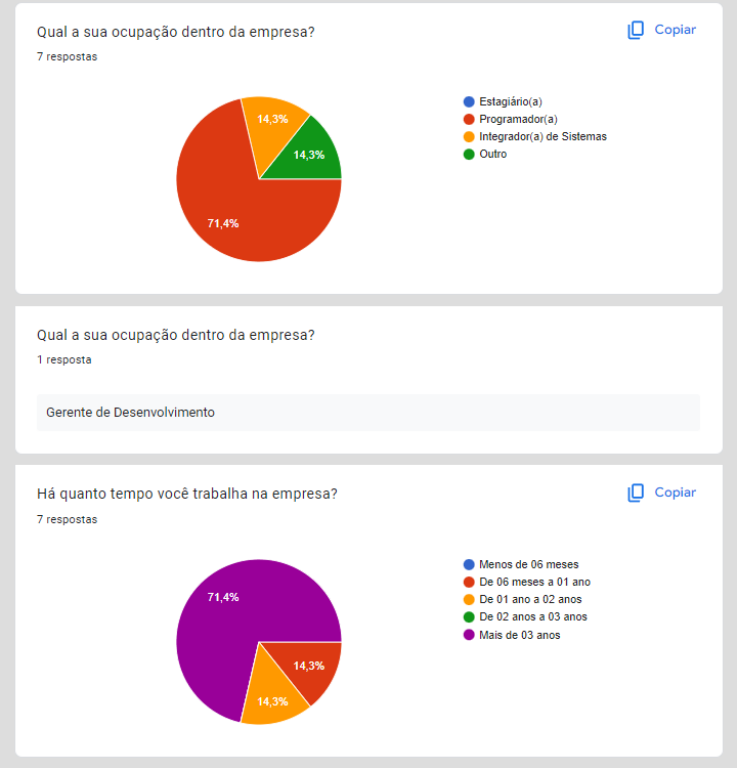


Fonte: O Autor

6.2 Resultados do Formulário de Pesquisa

Nesta seção, encontram-se os resultados obtidos na pesquisa realizada com os colaboradores da empresa. Dos dez convites enviados, somente sete colaboradores responderam a pesquisa, sendo estes cinco Programadores, um Integrador de Sistemas e um Gerente de Desenvolvimento. Em relação ao tempo de empresa, a maioria, cinco colaboradores, estão há mais de três anos na empresa, seguidos de um colaborador de um ano a dois anos de empresa e um colaborador de seis meses a um ano, conforme apresenta a Figura 6.6. Cabe ressaltar que, embora o número de participantes seja pequeno, buscou-se a avaliação de participantes especialistas no domínio da aplicação.

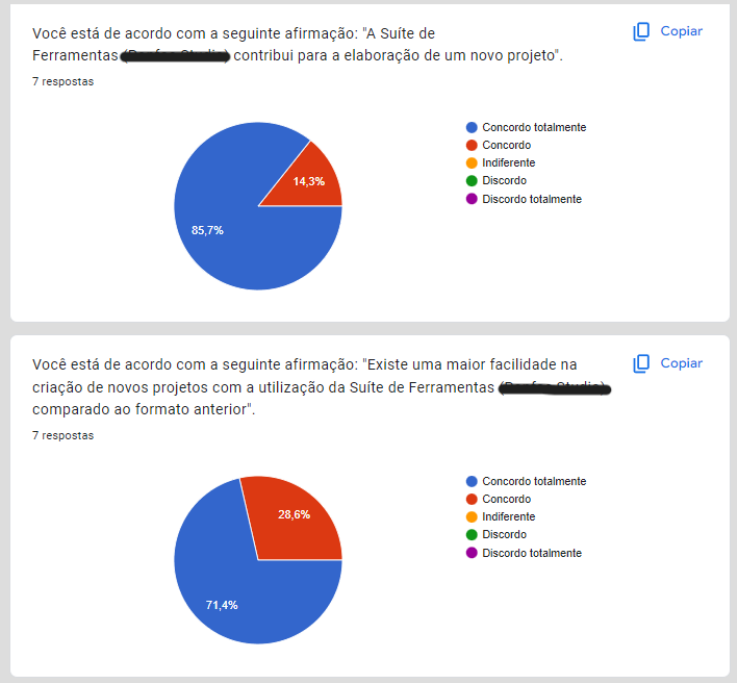
Figura 6.6: Resultados dos dados dos pesquisados



Fonte: O Autor

A seguir, apresento os resultados das quatro afirmações realizadas aos colaboradores acerca da suíte de ferramentas. Na Figura 6.7 é possível identificar que os entrevistados de modo geral "Concordam totalmente"ou "Concordam"que a suíte de ferramentas contribui para a elaboração de um novo projeto na empresa e torna este processo mais fácil se comparado ao modelo anterior.

Figura 6.7: Resultados da seção de avaliação da suíte de ferramentas parte 01



Fonte: O Autor

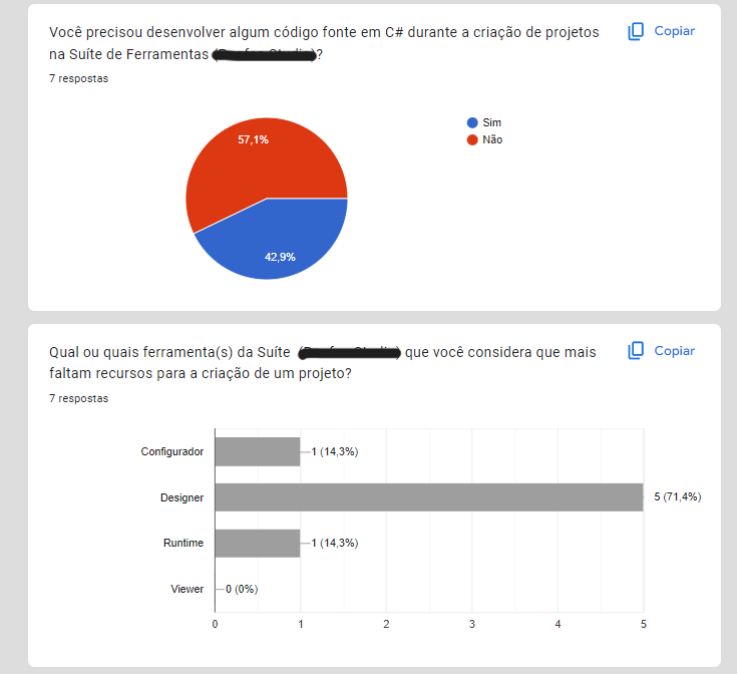
Na Figura 6.8, os colaboradores também "Concordam totalmente"ou "Concordam"que a suíte de ferramentas permite ao usuário a criação de um projeto em menor tempo. Ainda na Figura 6.8, observa-se que a afirmação referente a necessidade de criação de projetos sem auxílio de um desenvolver(a), apresentou uma maior variedade no resultado das respostas. Mesmo que a maioria "Concorde totalmente"ou "Concorde"que não é necessário o auxílio de um desenvolvedor(a), há colaboradores que afirmam que isso não é possível. Este cenário deve-se ao fato da complexidade de alguns projetos e a necessidade da inclusão de funcionalidades que tornem possível a resolução destes casos.

Figura 6.8: Resultados da seção de avaliação da suíte de ferramentas parte 02

Fonte: O Autor

A última parte da seção da avaliação da suíte de ferramentas apresenta um questionamento que é possível avaliar um dos objetivos do trabalho, pois verifica se ao criar um projeto foi necessário desenvolver algum código fonte em C#. Além disso, também identifica quais as ferramentas precisam de mais recursos para a criação de um projeto, conforme aponta a Figura 6.9.

Figura 6.9: Resultados da seção de avaliação da suíte de ferramentas parte 03



Fonte: O Autor

A primeira pergunta, que avalia se o colaborador conseguiu criar um novo projeto sem necessitar desenvolver um código fonte em C#, apontou que a maioria, quatro colaboradores, não necessitou, enquanto três colaboradores ainda precisaram utilizar algum código fonte em C#. Acredita-se que essa diferença possa estar relacionada novamente com a questão da complexidade dos projetos. Muitos projetos possuem características específicas que dificultam que a suíte de ferramentas seja 100% genérica, necessitando a adição de funcionalidades constantemente.

A segunda pergunta, que avalia qual ou quais ferramenta(s) o colaborador considera que mais faltam recursos para criação de novos projetos possibilitava múltiplas respostas. Identificou-se que com 71,4% a ferramenta que mais necessita de melhorias é o *Designer*. Este alto índice ocorre devido ao fato do *Designer* ser a ferramenta de desenvolvimento dos algoritmos de processamento de imagens e necessitar com frequência de novas técnicas e soluções para a resolução de problemas. A pergunta também permite analisar que o *Viewer* foi a ferramenta que se destacou por não necessitar de melhorias, isso porque este funciona apenas como um visualizador de resultados e imagens, e não depende diretamente da complexidade dos projetos.

A última pergunta da pesquisa possibilitou que o colaborador pudesse dissertar um feedback e/ou até mesmo uma sugestão para a suíte de ferramentas. Apenas um colaborador utilizou este espaço para escrever, no entanto, acredita-se que sua colocação corrobore com o que já foi destacado nas respostas anteriores: "*O [...] é uma ótima suíte para desenvolvimento de projetos, desde a comunicação com o hardware de integração até a elaboração de receitas de processamentos de imagens e ou inteligência artificial. Entretanto, em minha opinião, considero uma suíte que depende de auxílio para a utilização, visto a complexidade e diversidade das ferramentas que são abordadas*". Com base na colocação do colaborador que aponta para a necessidade de auxílio para a utilização da suíte de ferramentas, confirma-se que devido à complexidade dos projetos e à arquitetura da suíte de ferramentas é fundamental a execução de treinamentos e a elaboração de manuais de instrução que auxiliem os usuários.

7 CONCLUSÃO

Este trabalho descreveu o processo de criação de uma suíte de ferramentas para alterar a forma como uma empresa do Sul do Brasil, que desenvolve soluções completas em sistemas para inspeção visual automatizada, desenvolve os seus projetos. Construiu-se assim um produto no qual profissionais da área de integração de sistemas, que não possuem conhecimento em desenvolvimento de software, podem realizar os projetos de forma autônoma.

Através de uma pesquisa realizada, após a apresentação e utilização da suíte de ferramentas para os colaboradores, analisou-se que a mesma corrobora para a criação de projetos, facilitando a sua criação, configuração e instalação de maneira rápida e fácil, se comparada com a forma anterior de desenvolvimento dos projetos realizados pela empresa.

Além disso, com os resultados obtidos através da pesquisa e ao longo do desenvolvimento da suíte de ferramentas, identificaram-se pontos que necessitam de melhorias, tais como a necessidade de treinamentos para os usuários, a adição de funcionalidades e a criação de manuais de instrução. Outro ponto, que pode contribuir para a arquitetura da solução, seria executar o *Runtime* em um sistema operacional embarcado com um *hardware* dedicado, ao invés de utilizá-lo em ambiente *Windows*, tornando-o em uma unidade de processamento de imagens mais robusta e de tempo real.

A continuidade do trabalho se dará através da utilização da suíte de ferramentas no maior número de projetos possíveis dentro da empresa, incorporando nela funcionalidades específicas encontradas ao longo do desenvolvimento, tornando-a mais completa e genérica ao longo do tempo.

REFERÊNCIAS

ALMEIDA, T. C. de; ANDRADE, J. A. B. de. Benefícios e desafios da indústria 4.0 e o impacto durante a pandemia. **Revista Ibero-Americana de Humanidades, Ciências e Educação**, v. 9, p. 245–259, 2023. ISSN 2675–3375.

ANTONIO, D. S. et al. A indústria 4.0 e seus impactos na sociedade. **Revista Pesquisa e Ação**, v. 4, 2018. ISSN 2447-0627.

BARBOSA, R. E.; GASPAROTTO, A. M. S. Desenvolvimento de um software para gerenciamento da manutenção de acordo com o método tpm. 2015.

CORAL, E.; SELIG, P. M. Custos de qualidade: sua definição e aplicação. **Anais do Congresso Brasileiro de Custos-ABC**, 1994.

COSTA, D. V. da. Soluções configuráveis para inspeção visual automática. 2021.

ECMA-404. **The JSON Data Interchange Standard.** 2017. <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>. [Online; acessado 19 de Março de 2023].

GARCIA, J. L. Programação no-code no ensino superior: possibilidades de avaliação e aprendizagem ativas. **Revista Valore**, v. 8, p. 78–92, 2023.

GIT. **Git**. 2022. <https://git-scm.com/>. [Online; acessado 18 de Março de 2023].

MEDEIROS, P. H. de O.; JúNIOR, I. de S. Q. Software para análise e dimensionamento de um sistema fotovoltaico conectado à rede. Universidade Federal Rural do Semi-Árido Mossoró, 2020.

MICROSOFT. **O que é o Windows Communication Foundation - WCF | Microsoft Learn**. 2023. <https://learn.microsoft.com/pt-br/dotnet/framework/wcf/whats-wcf>. [Online; acessado 19 de Março de 2023].

MICROSOFT. **O que é o Windows Forms - Windows Forms .NET | Microsoft Learn.** 2023. <https://learn.microsoft.com/pt-br/dotnet/desktop/winforms/overview/?view=netdesktop-7.0>. [Online; acessado 18 de Março de 2023].

MICROSOFT. **Visual Studio: IDE e Editor de Código para Desenvolvedores de Software e Teams.** 2023. <https://visualstudio.microsoft.com/pt-br/>. [Online; acessado 18 de Março de 2023].

MICROSOFT. **Visão geral do XAML - WPF .NET | Microsfot Learn**. 2023. <https://learn.microsoft.com/pt-br/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0>. [Online; acessado 19 de Março de 2023].

MODICON INC., I. A. S. Modicon modbus protocol reference guide. 1996.

NETO, O. C. de O. Um relato de experiência sobre a inserção de práticas de qualidade de software no ambiente de desenvolvimento da startup proj4me. 2020.

NOCODE.TECH. **What is NoCode?** 2022. Disponível em <https://www.nocode.tech/lessons/what-is-nocode.> [Online; acessado 25 de Agosto de 2023].

OLIVEIRA, R. V. de et al. Qualidade em serviços no desenvolvimento de software: proposta de um instrumento de avaliação. 2012.

ROGGIA, L.; FUENTES, R. C. **Automação Industrial**. [S.l.]: Universidade Federal de Santa Maria, Colégio Técnico Industrial de Santa Maria, Rede e-Tec Brasil, 2016.

SAKURAI, R.; ZUCHI, J. D. As revoluções industriais até a indústria 4.0. **Revista Interface Tecnológica**, v. 15, p. 480–491, 2018.

SHARP, J.; JAGGER, J. **Microsfot Visual C# .NET Step by Step**. [S.l.]: Microsoft Press, 2003.

SQLITE. **What Is SQLite?** 2022. <https://sqlite.org/index.html>. [Online; acessado 19 de Março de 2023].

TAYLOR, W.; MELLOY, B.; DHARWADA, P. The effects of static multiple sources of noise on the visual search component of human inspection. **International Journal of Industrial Ergonomics**, p. 195–207, 2004. ISSN 0169-8141.

VILELA, D. L. et al. Sistema didÁtico de reconhecimento de objetos para automaÇÃo industrial. 2008.