

Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Avaliação de plataformas *low code* e *no code* como estratégia para aumento da produtividade no desenvolvimento de software

Juciê Xavier da Silva

Teresina-PI, 14 de Maio de 2024

Juciê Xavier da Silva

Avaliação de plataformas *low code* e *no code* como estratégia para aumento da produtividade no desenvolvimento de software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Prof. Dr. Guilherme Amaral Avelino

Teresina-PI

14 de Maio de 2024

FICHA CATALOGRÁFICA
Universidade Federal do Piauí
Sistema de Bibliotecas UFPI - SIBi/UFPI
Biblioteca Setorial do CCN

S586a Silva, Juciê Xavier da.
Avaliação de plataformas *low code* e *no code* como estratégia para aumento da produtividade no desenvolvimento de software / Juciê Xavier da Silva. -- 2024.
77 f. : il. color.

Dissertação (Mestrado) - Universidade Federal do Piauí. Centro de Ciências da Natureza. Programa de Pós-Graduação em Ciência da Computação, Teresina, 2024.
"Orientador: Prof. Dr. Guilherme Amaral Avelino."

1. Plataformas de desaenvolvimento. 2. *Low code Outsystems*. 3. *No code Bubble*. I. Avelino, Guilherme Amaral. II. Titulo.

CDD 005.43

Bibliotecária: Caryne Maria da Silva Gomes - CRB3/1461

Juciê Xavier da Silva

Avaliação de plataformas *low code* e *no code* como estratégia para aumento da produtividade no desenvolvimento de software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Documento assinado digitalmente
gov.br GUILHERME AMARAL AVELINO
Data: 19/06/2024 15:50:11-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Guilherme Amaral Avelino
Orientador

Documento assinado digitalmente
gov.br RAIMUNDO SANTOS MOURA
Data: 14/06/2024 11:02:20-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Raimundo Santos Moura
Docente Interno ao Programa

Documento assinado digitalmente
gov.br PEDRO DE ALCANTARA DOS SANTOS NETO
Data: 18/06/2024 14:45:19-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Pedro de Alcantara dos Santos Neto
Docente Interno ao Programa

Documento assinado digitalmente
gov.br DAVI VIANA DOS SANTOS
Data: 14/06/2024 19:26:08-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Davi Viana dos Santos
Docente Externo ao Programa

Teresina-PI
14 de Maio de 2024

Ao meu DEUS o senhor dos céus e da terra, que permite os estudos, o conhecimento, a pesquisa científica, e caminhos de justiça, que levam a sua salvação, a ele a honra a glória e louvor. Tudo foi feito por ele. Arrependei-vos de vossos pecados, e ele te salvará.

Agradecimentos

Primeiramente, elevo minha mais profunda gratidão ao meu Deus, meu refúgio e fortaleza, em quem sempre busquei socorro. Foi Ele que alimentou minha fé e me deu forças para continuar, mesmo quando sofri um infarto durante este mestrado, ameaçando minha jornada e minha vida. Em um momento em que a sobrevivência parecia incerta, lembrei-me d'Ele e, em sua infinita misericórdia, fui socorrido, salvo e presenteado com uma nova chance de viver. A Ele, dedico este mestrado, toda honra, glória e louvor.

Aos meus queridos pais, Por cada sacrifício, por cada palavra de incentivo, por cada oração em meu favor. Por cada lágrima de alegria compartilhada e por cada obstáculo que enfrentamos juntos. Dedico este trabalho a vocês, que foram a força silenciosa por trás de cada conquista, sempre com a benção e a graça do Deus Todo Poderoso.

À minha amada esposa, meu eterno agradecimento. Em meio à tempestade, sua presença foi uma luz que me guiou, e sua fé inabalável fortaleceu meu espírito, fazendo-me acreditar que um milagre estava à mão. Seu amor e dedicação foram fundamentais, e por isso, sou eternamente grato.

Agradeço também ao meu amado filho, que esteve ao meu lado em cada desafio, sempre pronto para me apoiar. Sua curiosidade em acompanhar minhas notas e progresso neste mestrado trouxe-me alegria e motivação, relembrando-me da importância do aprendizado contínuo e do impacto que temos nas gerações futuras.

Agradeço também a Deus pela vida do meu orientador, que com tanta dedicação e prontidão, guiou-me neste mestrado. Sua paixão e conhecimento não apenas iluminaram meu caminho acadêmico, mas tornaram-se uma fonte de inspiração que carregarei comigo por toda a minha vida.

"Há uma maneira de fazer melhor – encontre-a."— (Autor Thomas A. Edison)

Resumo

Desde o início da era digital, a indústria de software tem enfrentado uma crescente e constante demanda por soluções tecnológicas sofisticadas. Essa demanda acelerada intensificou a busca por aprimoramentos na produtividade e na eficiência do desenvolvimento. Ao longo dos anos, várias estratégias e inovações foram propostas, desde a adoção da programação orientada a objetos até a implementação de Ambientes de Desenvolvimento Integrado (IDEs) e a incorporação de metodologias ágeis. Entre as propostas mais recentes e potencialmente transformadoras, destacaram-se as plataformas de desenvolvimento *low code* e *no code*. Diante desse cenário, o estudo analisou essas plataformas emergentes, com foco especial nas alegações de melhoria substancial na produtividade. Para esse estudo, foi adotada uma metodologia dividida em duas etapas. A primeira etapa consistiu em uma revisão de literatura multivocal, abrangendo tanto fontes acadêmicas tradicionais a chamada literatura branca, quanto fontes informais e contemporâneas também conhecidas como literatura cinza. O objetivo era extrair o máximo de informações sobre os benefícios e desafios dessas plataformas, enriquecendo os achados literários. Na segunda etapa, foram analisados na prática os achados da revisão de literatura por meio de um estudo de caso empírico que envolveu o desenvolvimento de um protótipo para promover a coleta seletiva de lixo, composto por três perfis (Coletor, Produtor e Reciclador) e desenvolvido utilizando as plataformas *OutSystems* e *Bubble*. O objetivo era correlacionar teoria e prática e proporcionar uma análise abrangente. Como resultados do estudo, foi possível observar uma série de vantagens e desafios associados ao uso dessas plataformas. Entre as vantagens, destacaram-se a abstração de código, a automatização de tarefas repetitivas, a reutilização de componentes e a simplificação da colaboração entre os membros da equipe de desenvolvimento. Também foram identificados desafios, como a dependência de plataforma, requisitos de conhecimento técnico, funcionalidades limitadas e preocupações com segurança de dados. Como contribuições, esta pesquisa oferece uma avaliação empírica da plataformas *low code Outsystems* e *no code Bubble*, fornecendo avaliações sobre seu impacto na produtividade do desenvolvimento de software.

Palavras-chaves: *low code*, *no code*, *desafios*, *benefícios*, *protótipo. Bubble*, *Outsystems.*

Abstract

Since the beginning of the digital era, the software industry has faced a growing and constant demand for sophisticated technological solutions. This accelerated demand has intensified the search for improvements in productivity and development efficiency. Over the years, several strategies and innovations have been proposed, from the adoption of object-oriented programming to the implementation of Integrated Development Environments (IDEs) and the incorporation of agile methodologies. Among the most recent and potentially transformative proposals, low-code and no-code development platforms stood out. Against this background, the study analyzed these emerging platforms, with a special focus on claims of substantial improvements in productivity. For this study, a methodology divided into two stages was adopted. The first stage consisted of a multivocal literature review, covering both traditional academic sources, known as white literature, and informal and contemporary sources also known as gray literature. The objective was to extract as much information as possible about the benefits and challenges of these platforms, enriching the literary findings. In the second stage, the findings of the literature review were analyzed in practice through an empirical case study that involved the development of a prototype to promote selective waste collection, consisting of three profiles (Collector, Producer and Recycler) and developed using the OutSystems and Bubble platforms. The aim was to correlate theory and practice and provide a comprehensive analysis. As results of the study, it was possible to observe a series of advantages and challenges associated with the use of these platforms. Advantages include code abstraction, automation of repetitive tasks, reuse of components and simplification of collaboration between members of the development team. Challenges were also identified, such as platform dependency, technical expertise requirements, limited functionality, and data security concerns. As contributions, this research offers an empirical evaluation of the *low code Outsystems* and *no code Bubble* platforms, providing assessments on their impact on software development productivity.

Keywords: *low code*, *no code*, *challenges*, *benefits*, *prototype*, *Bubble*, *Outsystems*.

Lista de ilustrações

Figura 1 – Metodologia Revisão da Literatura 22
Figura 2 – Diagrama de Caso de Uso 32
Figura 3 – Base de Dados Outsystems 55
Figura 4 – Base de Dados Bubble 56
Figura 5 – Fluxo de trabalho outsystems 57
Figura 6 – Fluxo de Trabalho Bubble 59

Lista de tabelas

Tabela 1 – Resumo da Seleção de Fontes - critérios inclusão. 40
Tabela 2 – Resumo da Seleção de Fontes - critérios exclusão. 41
Tabela 3 – Ferramentas *low code* e *no code* identificadas. 43
Tabela 4 – Benefícios das ferramentas *low code* e *no code*. 43
Tabela 5 – Limitação das ferramentas *low code* e *no code*. 44
Tabela 6 – Retorno: Literatura Branca 45
Tabela 7 – Resultados: Artigos Categorizados 46
Tabela 8 – Resumo da Seleção de Artigos - critérios inclusão. 47
Tabela 9 – Resumo da Seleção de Artigos - critérios exclusão. 48
Tabela 10 – Plataformas *low code* e *no code* identificadas. 49
Tabela 11 – Benefícios das ferramentas *low code* e *no code*. 50
Tabela 12 – limitações das ferramentas *low code* e *no code*. 51
Tabela 13 – Resultados Outsystems 53
Tabela 14 – Resultados Bubble 53

Lista de abreviaturas e siglas

API	- Interface de Programação de Aplicativos (API, do inglês: Application Programming Interface)
CSS	- Folhas de Estilo em Cascata (CSS, do inglês: Cascading Style Sheets)
DSL	- Linguagem de Domínio Específico (DSL, do inglês: Domain-Specific Language)
ES	- Engenharia de Software
GPL	- Linguagem de Propósito Geral (GPL, do inglês: General Purpose Language)
HTML	- Linguagem de Marcação de Hipertexto (HTML, do inglês: HyperText Markup Language)
IDE	- Ambiente de Desenvolvimento Integrado (IDE, do inglês: Integrated Development Environment)
OOP	- Programação Orientada a Abjetos (OOP, do inglês: Object Oriented Programming)
QPE	- Questão de Pesquisa em Estudo de Caso
QPL	- Questão de Pesquisa em Literatura
SQL	- Linguagem de Consulta Estruturada (SQL, do inglês: Structured Query Language)
UML	- Linguagem de Modelagem Unificada (UML, do inglês: Unified Modeling Language)
XML	- Linguagem de Marcação Extensível (XML, do inglês: eXtensible Markup Language)

Sumário

I INTRODUÇÃO 1

1 INTRODUÇÃO . . . 3
1.1 Contexto e motivação . . . 3
1.2 Definição do problema . . . 6
1.3 Objetivos . . . 7
1.4 Estrutura do trabalho . . . 8

II PREPARAÇÃO DA PESQUISA 9

2 ESTADO DA ARTE . . . 11
2.1 Fundamentação Teórica . . . 11
2.1.1 Linguagens de Domínio Especifico . . . 11
2.1.2 IDEs . . . 12
2.1.3 Low code/No code . . . 13
2.2 Trabalhos Relacionados . . . 15

III METODOLOGIA 19

3 METODOLOGIA . . . 21
3.1 Revisão da Literatura . . . 21
3.1.1 Planejamento . . . 23
3.1.2 Busca e Seleção . . . 24
3.1.2.1 Seleção - Literatura Cinza . . . 24
3.1.2.2 Seleção - Literatura Branca . . . 24
3.1.3 Critérios de inclusão e exclusão . . . 25
3.1.3.1 Literatura Cinza . . . 25
3.1.3.2 Literatura branca . . . 26
3.1.4 Extração de dados . . . 27
3.1.4.1 literatura Branca/Cinza . . . 27
3.2 Estudo de Caso . . . 28
3.2.1 Planejamento do Estudo de Caso . . . 30
3.2.1.1 Objetivo . . . 30
3.2.1.2 Questões de Pesquisas . . . 30
3.2.1.3 O Caso . . . 30
3.2.1.4 Coleta dos Dados . . . 32

3.2.1.5 Perfil do desenvolvedor . . . 33
3.2.1.6 O Treinamento . . . 33
3.2.1.7 O Protocolo . . . 33

IV PARTE FINAL 37

4 RESULTADOS . . . 39
4.1 Revisão da Literatura Cinza . . . 39
4.1.1 Busca e Seleção . . . 39
4.1.2 Respostas às Questões de Pesquisa . . . 41
4.2 Revisão da Literatura Branca . . . 44
4.2.1 Busca e Seleção . . . 44
4.2.2 Respostas às Questões de Pesquisa . . . 48
4.2.3 Considerações Finais . . . 51
4.3 Estudo de Caso . . . 52
4.3.1 Descrição do Protótipo . . . 52
4.3.2 Desenvolvimento . . . 53
4.3.3 Respostas às Questões de Pesquisa . . . 60
4.3.4 Considerações Finais . . . 64

5 CONCLUSÕES . . . 67

Conclusão e Trabalhos Futuros . . . 67
5.1 Síntese . . . 67
5.2 Contribuições . . . 67
5.3 Trabalhos futuros . . . 68
5.4 Publicações . . . 68
5.4.1 Agradecimentos . . . 69

REFERÊNCIAS . . . 71

APÊNDICES 75

APÊNDICE A – FORMULÁRIO DE COLETA DE DADOS . . . 77

Parte I

Introdução

1 Introdução

Neste capítulo, são apresentados os elementos de contexto e motivação que fundamentam a investigação em questão. Inicia-se com uma introdução ao contexto pertinente e às motivações do estudo. Em seguida, a problemática central é definida, e os objetivos que direcionam a pesquisa são delineados.

1.1 Contexto e motivação

Desde o início da era digital e do surgimento da indústria de *software*, o mundo tem testemunhado um crescimento exponencial na demanda por soluções tecnológicas sofisticadas e avançadas. Esse cenário tem instigado, de forma incessante, uma urgência pela ampliação da produtividade, especialmente quando observamos que a capacidade técnica atual, em diversas situações, parece não conseguir atender à velocidade e à alta demanda (SOMMERVILLE, 2018). Essa disjuntiva entre a necessidade do mercado e a capacidade produtiva tem raízes profundas e não é uma mera consequência de nosso tempo. De fato, desde os primeiros passos do desenvolvimento de *software*, inúmeros esforços foram realizados por especialistas, acadêmicos e profissionais para otimizar e aprimorar os processos de produção (PRESSMAN; MAXIM, 2019).

Nesse contexto, não podemos esquecer que a engenharia de *software* tem, ao longo das décadas, enfrentado persistentemente o desafio da produtividade. Em busca de solucionar esta questão, várias inovações foram introduzidas. Uma das mais revolucionárias foi, sem dúvida, a incursão da programação orientada a objetos (OOP, do inglês: *Object Oriented Programming*). Essa abordagem propôs uma maneira de espelhar as intrincadas nuances do mundo real através de conceitos técnico-conceituais, como classes, herança e polimorfismo. Em um cenário prático, linguagens como Java e C++ abriram portas para que os desenvolvedores pudessem criar e manipular abstrações que emulam entidades palpáveis e tangíveis, proporcionando um meio mais intuitivo e natural de programação (BROOKS, 1995).

Ao mesmo tempo, o surgimento de Ambientes de Desenvolvimento Integrado (IDE, do inglês: *Integrated Development Environment*) como o Eclipse[1] e *IntelliJ IDEA*[2], dotados de ferramentas sofisticadas, como conclusão automática de código, depuração visual e integração com robustos sistemas de controle de versão, passou a ser um caminho para acelerar e aprimorar o desenvolvimento, potencializando a produtividade dos programadores. Adicionalmente, é relevante mencionar o papel das Linguagens de Domínio Específico (DSL,

[1] https://www.eclipse.org/downloads/
[2] https://www.jetbrains.com/pt-br/idea/download/

do inglês: *Domain-Specific Language*) no contexto da engenharia de software. As DSLs são linguagens de programação desenvolvidas para resolver problemas em um domínio particular, proporcionando uma abordagem mais direcionada e eficiente para solucionar questões específicas. Um exemplo notável de DSL é a Linguagem de Consulta Estruturada (SQL, do inglês: *Structured Query Language*), que é amplamente utilizado para consultar e manipular dados em bancos de dados relacionais, oferecendo uma sintaxe estruturada e específica para operações como inserção, consulta, atualização e exclusão de dados.

Além disso, uma tendência inovadora começou a ganhar força e influenciar profundamente o setor de desenvolvimento de software. Trata-se da ascensão das metodologias ágeis. Contrapondo-se às práticas tradicionais, muitas vezes engessadas e burocráticas, as metodologias ágeis propuseram uma dinâmica fluida, adaptável e intensamente focada no valor percebido pelo cliente final. O manifesto ágil, lançado em 2001, tornou-se o estandarte dessa mudança paradigmática, enfatizando uma abordagem mais humana, colaborativa e centrada nas necessidades reais dos usuários (PADUA, 2019).

Neste contexto, as práticas ágeis abraçaram a "entrega contínua", onde ciclos curtos de *feedback* e adaptabilidade contínua se tornaram a norma. Esse modo de operação, combinado com um ambiente de trabalho colaborativo e uma cultura de aprendizado constante, tem sido fundamental para impulsionar a produtividade e garantir um alto grau de satisfação dos clientes na indústria.

Entretanto, mesmo com tantas inovações e avanços tecnológicos, a produtividade ainda parece ser distante. Uma possível explicação para essa aparente contradição é que, independentemente da evolução das ferramentas e metodologias, o desenvolvimento de *software* ainda depende intrinsecamente do capital humano. A expertise técnica, aliada ao tempo e à preparação, permanece sendo um pilar insubstituível no processo (SOMMERVILLE, 2018). Assim, por mais avançada que seja a tecnologia, ela continua sendo apenas uma ferramenta nas mãos de profissionais qualificados, exigindo, por consequência, um aporte significativo de esforço, dedicação e, sobretudo, paixão.

Em resposta a essas preocupações e ao aparente hiato produtivo, observa-se, mais recentemente, o advento e consolidação de plataformas de desenvolvimento classificadas como *low code* e *no code*. Estas plataformas emergem como um pilar na arquitetura do desenvolvimento de *software*, propondo-se a democratizar a criação de soluções tecnológicas e, simultaneamente, agilizar os ciclos de desenvolvimento (LUO et al., 2021).

Assim, plataformas *low code* como o próprio nome sugere requerem uma quantidade mínima de codificação manual, permitindo que os desenvolvedores construam aplicações robustas e personalizadas através de interfaces gráficas intuitivas, arrastando e soltando componentes pré-construídos. Já as plataformas *no code* levam este paradigma um passo adiante, eliminando a necessidade de codificação, tornando o desenvolvimento de *software* acessível até mesmo para indivíduos sem formação técnica em programação (KRISHNARAJ

et al., 2022).

Estas abordagens oferecem agilidade no desenvolvimento. O recurso de arrastar e soltar, aliado à reutilização de componentes pré-definidos, pode reduzir o tempo necessário para construir e implementar aplicações. Essa agilidade se traduz em lançamentos mais rápidos e ciclos de *feedback* mais curtos. Além disso, ao minimizar a codificação manual, estas plataformas têm o potencial de abrir portas para um público muito mais amplo, permitindo que especialistas de domínio, mesmo sem um profundo conhecimento em programação, possam desenvolver soluções funcionais específicas para seus campos. Essa democratização pode, por sua vez, levar a custos potencialmente reduzidos, visto que os ciclos de desenvolvimento são mais curtos e a necessidade de equipes de desenvolvimento altamente especializadas é diminuída (KäSS; STRAHRINGER; WESTNER, 2023).

Adicionalmente, muitas dessas plataformas ainda oferecem a capacidade de integrar código personalizado quando necessário, garantindo que os requisitos específicos sejam atendidos. Este equilíbrio entre simplicidade e personalização é complementado por uma manutenção simplificada, onde atualizações e alterações podem ser implementadas com facilidade através da própria interface gráfica (ALAMIN et al., 2021).

A promessa central dessas plataformas é, sem dúvida, aumento de produtividade. Elas aspiram maximizar a eficiência, possibilitando que organizações desenvolvam e implementem soluções em uma fração do tempo tradicionalmente necessário. Contudo, é essencial reconhecer que, embora as plataformas *low code* e *no code* apresentem um potencial transformador, elas também possuem suas limitações. Sua adequação deve ser avaliada cuidadosamente em função do contexto e das necessidades específicas de cada projeto (PRINZ; RENTROP; HUBER, 2021).

Nesse contexto, um aspecto crítico permanece em discussão no cerne da comunidade acadêmica e industrial: a autenticidade das alegações dessas plataformas de desenvolvimento *low code* e *no code*. Enquanto a retórica em torno dessas tecnologias é inegavelmente atraente, um exame mais rigoroso é fundamental para discernir até que ponto essas plataformas cumprem realmente suas promessas.

Historicamente, muitas tecnologias emergentes na indústria do *software* têm sido recebidas com entusiasmo inicial, mas após uma avaliação mais profunda, certas limitações ou desvantagens se tornem aparentes (SOMMERVILLE, 2018). Por exemplo, questões como escalabilidade, segurança e integração com sistemas legados podem ser desafios inesperados para soluções criadas em plataformas *low code* e *no code*. Além disso, a dependência de tais plataformas pode levar a problemas de portabilidade ou interoperabilidade com outros sistemas.

Adicionalmente, enquanto essas ferramentas se posicionam como soluções que democratizam o desenvolvimento, é necessário investigar se elas também não introduzem

uma nova forma de "caixa preta", onde os usuários podem ser alheios ao funcionamento interno e às implicações das soluções que estão construindo. Isto poderia, em última instância, resultar em aplicações com vulnerabilidades inadvertidas.

É, portanto, imperativo que a comunidade científica e de desenvolvimento adote uma postura proativa, conduzindo estudos empíricos, *benchmarks*, estudos de casos e avaliações de campo para determinar a eficácia real dessas plataformas. Somente através de uma análise metódica e baseada em evidências podemos afirmar, com confiança, se essas tecnologias não apenas atendem às expectativas, mas também se são adequadas para uma ampla gama de aplicações e contextos no mundo real.

1.2 Definição do problema

Conforme comentado na seção anterior, é relevante para a comunidade científica e de desenvolvimento compreender se realmente as plataformas de desenvolvimento *low code* e *no code* proporcionam ganho de produtividade, conforme relatado na literatura. A ascensão destas plataformas foi acompanhada por reivindicações ousadas sobre sua capacidade de acelerar o desenvolvimento, reduzir custos e tornar o processo de criação de software mais inclusivo (KRISHNARAJ et al., 2022). No entanto, como em qualquer avanço tecnológico, é imperativo que tais reivindicações sejam submetidas a uma avaliação rigoroso.

A avaliação dessas plataformas vai além de simplesmente medir seu desempenho em comparação com métodos tradicionais. Trata-se de entender os retornos inerentes ao uso dessas ferramentas (LUO et al., 2021). Por exemplo, enquanto o tempo de desenvolvimento é reduzido, e isso apresentado como um benefício, quais são os elementos que permitem tal redução no tempo de desenvolvimento? Além disso, a longo prazo, qual é o custo de manter soluções desenvolvidas em plataformas *low code* e *no code* em comparação com aquelas desenvolvidas tradicionalmente?

Dessa forma, uma avaliação aprofunda destas plataformas é essencial por várias razões. Primeiramente, muitas organizações estão considerando fazer investimentos significativos nessa tecnologia. Para tomar decisões informadas, precisam de dados concretos e análises imparciais sobre os reais benefícios e possíveis desvantagens (PRINZ; RENTROP; HUBER, 2021). Além disso, o campo da engenharia de software tem uma longa tradição de inovação; contudo, cada inovação carrega consigo o potencial de novos desafios e complicações. Ao avaliar novas ferramentas e metodologias, a comunidade pode se antecipar a possíveis problemas, desenvolvendo melhores práticas e orientações para a adoção efetiva dessas tecnologias (PRESSMAN; MAXIM, 2019).

Em última análise, o objetivo não é simplesmente adotar novas tecnologias porque são novas ou estão na moda, mas sim entender como elas se encaixam no ecossistema

existente de desenvolvimento de software, qual é seu valor real e como podem ser usadas para melhorar verdadeiramente os processos e resultados (KäSS; STRAHRINGER; WESTNER, 2023). Esta é a razão pela qual a avaliação das plataformas de desenvolvimento *low code* e *no code* não é apenas desejável, mas essencial para a integridade e evolução contínua da disciplina de engenharia de software.

1.3 Objetivos

O objetivo geral deste estudo é conduzir uma investigação empírica sobre plataformas de desenvolvimento *low code* e *no code* para avaliar se elas proporcionam um aumento na produtividade no desenvolvimento de aplicações.

O objetivo geral pode ser dividido nos seguintes objetivos específicos:

- Objetivo 1: Elucidar os benefícios e caracterizar as limitações inerentes às plataformas de desenvolvimento *low code* e *no code*.

  Metodologia Proposta: Conduzir uma revisão abrangente de literatura, através de trabalhos informais, documentos técnicos, industriais e acadêmicos. Esta revisão será orientada tanto para destacar os benefícios intrínsecos das plataformas quanto para identificar os desafios e restrições associados à sua adoção e implementação.

- Objetivo 2: Categorizar os tipos de aplicações que podem ser concebidas utilizando plataformas *low code* e *no code*.

  Metodologia Proposta: Investigar a literatura acadêmica e cinza para discernir as categorias de aplicações (*Web*, *mobile* ou *desktop*) que são viáveis e comuns para desenvolvimento nessas plataformas, e posteriormente sintetizá-las em um compêndio abrangente.

- Objetivo 3: Desenvolver um protótipo que permita validar empiricamente os achados desta pesquisa no contexto prático.

  Metodologia proposta: A metodologia para este objetivo é baseada no modelo de estudo de caso proposto por (RUNESON; HÖST, 2009), amplamente adotado no campo da engenharia de software. O uso deste modelo é importante para obter uma compreensão detalhada dos processos e práticas associados ao desenvolvimento de software. A abordagem de estudo de caso permite a análise aprofundada e contextualizada, contribuindo para um entendimento mais rico e informativo das diversas questões relacionadas ao desenvolvimento e à engenharia de software.

1.4 Estrutura do trabalho

Este estudo é estruturado em cinco capítulos, cada um desempenhando uma função específica. O Capítulo 1 constitui a introdução, delineando o problema de pesquisa e oferecendo uma visão abrangente do contexto e das questões que impulsionam o estudo. O Capítulo 2 aborda a preparação da pesquisa, apresentando a fundamentação teórica pertinente e uma análise crítica dos trabalhos relacionados. No Capítulo 3, é apresentada a proposta de pesquisa, evidenciando a abordagem metodológica adotada, a revisão de literatura multivocal que embasa o estudo e uma descrição minuciosa do estudo de caso. O Capítulo 4 constitui a essência do estudo, fornecendo uma análise detalhada dos resultados obtidos através da pesquisa realizada, incluindo a apresentação dos dados coletados, as análises conduzidas e as conclusões decorrentes desses resultados. No Capítulo 5, são apresentadas as conclusões alcançadas com base nos resultados obtidos, destacando as descobertas mais relevantes, seus benefícios e limitações. Por fim, são delineadas sugestões para pesquisas futuras, apontando direções para estudos subsequentes e contribuições para o progresso do conhecimento na área em questão.

Parte II

Preparação da pesquisa

2 Estado da Arte

O conceito de "estado da arte"na pesquisa científica refere-se ao compêndio dos mais recentes e significativos resultados sobre um tema específico, espelhando o crescimento e a inovação em determinada área de estudo (SOMMERVILLE, 2018). Na engenharia de software, a relevância do estado da arte é multifacetada. Como (SOMMERVILLE, 2018) e (PRESSMAN; MAXIM, 2019) argumentam, essa compreensão permite discernir desafios e lacunas atuais no campo, ao mesmo tempo que sinaliza potenciais oportunidades e tendências emergentes.

Este capítulo é estruturado em duas seções principais. A primeira seção, "Fundamentação Teórica", tem o objetivo de contextualizar e oferecer ao leitor os conceitos fundamentais para um entendimento aprofundado do tema em questão. Já a segunda seção, "Trabalhos Relacionados", busca situar a pesquisa atual dentro do panorama mais amplo da área, fornecendo uma visão abrangente do estado da arte e como diferentes estudos têm abordado questões similares.

2.1 Fundamentação Teórica

Esta seção concentra-se na fundamentação teórica e nas bases acadêmicas que norteiam e embasam esta pesquisa.

2.1.1 Linguagens de Domínio Especifico

Uma linguagem de domínio específico (DSL, do inglês: *Domain-Specific Language*) é uma linguagem criada para apoiar um conjunto particular de tarefas, conforme elas são realizadas em um domínio específico. Elas oferecem ganhos substanciais em expressividade e facilidade de uso em comparação com as linguagens de programação de propósito geral (LPG, do inglês: General Purpose Language) em seu domínio de aplicação (MERNIK; HEERING; SLOANE, 2005).

Uma DSL pode ser útil para a engenharia de software por vários motivos. Primeiro, ela pode melhorar a comunicação com os especialistas do domínio, pois usa conceitos e terminologia familiares para eles. Segundo, ela pode aumentar o foco e a produtividade dos desenvolvedores, pois permite que eles expressem soluções em um nível mais alto e abstrato, evitando detalhes desnecessários ou irrelevantes. Terceiro, ela pode facilitar a verificação, validação e manutenção do software, pois reduz a complexidade e o tamanho do código-fonte e aumenta a consistência e a modularidade (TOMASSETTI, 2017).

Para criar uma DSL, é preciso seguir um processo que envolve as fases de decisão, análise, design e implementação. Na fase de decisão, deve-se avaliar se vale a pena desenvolver uma DSL para o domínio em questão, considerando os benefícios esperados, os custos envolvidos e os riscos associados. Na fase de análise, deve-se estudar o domínio e identificar os conceitos, as relações, as restrições e as operações relevantes para ele. Na fase de design, deve-se definir a sintaxe, a semântica e a notação da DSL, escolhendo entre diferentes abordagens possíveis (por exemplo, textual ou gráfica, externa ou interna, declarativa ou imperativa). Na fase de implementação, deve-se construir a DSL usando ferramentas adequadas (por exemplo, geradores de analisadores léxicos e sintáticos, *frameworks* de modelagem gráfica ou editores projetivos) e testá-la com casos de uso reais (MERNIK; HEERING; SLOANE, 2005).

Existem muitos exemplos de DSLs usadas na engenharia de software, como SQL para consultas a bancos de dados, HTML para layout web, CSS para estilo, XML para codificação de dados, UML para modelagem visual, VHDL para design de hardware, ANTLR para definições de analisadores léxicos e sintáticos, *Make* para sistemas de construção e *Latex* para layout de documentos (TOMASSETTI, 2017). Essas linguagens mostram como as LDEs podem facilitar o desenvolvimento de software em diferentes domínios e contextos.

2.1.2 IDEs

Os Ambientes de Desenvolvimento Integrado (IDE, do inglês *Integrated Development Environments*) representam uma evolução marcante no ecossistema de desenvolvimento de software. Eles são plataformas ou *suites* de software que combinam ferramentas comuns necessárias para escrever, testar e depurar código-fonte.

Segundo (FOWLER, 2004), um IDE é uma ferramenta que fornece uma variedade de recursos para facilitar o processo de desenvolvimento de software para programadores. Estes recursos podem abranger desde edição de código-fonte, compilação automática, depuração, até integração com sistemas de controle de versão. Tudo isso em uma única interface, proporcionando um fluxo de trabalho contínuo e otimizado para o desenvolvedor.

A produtividade no desenvolvimento de software é uma preocupação central para as organizações e equipes de desenvolvimento. O surgimento e adoção de IDEs demonstram um esforço contínuo para simplificar e acelerar o processo de desenvolvimento. Segundo (BERGIN et al., 2005), a utilização de IDEs pode aumentar a produtividade do desenvolvedor ao reduzir o tempo gasto em tarefas repetitivas, fornecendo sugestões inteligentes e oferecendo ferramentas avançadas para depuração e análise de código.

Segue alguns exemplos de IDEs com Recursos Produtivos:

- Visual Studio Code (VSCode): Originado da *Microsoft*, é um IDE leve, mas poderoso.

Uma das suas características mais impressionantes é sua extensibilidade, com um vasto mercado de extensões que abrangem linguagens, *frameworks* e ferramentas. Além disso, seu recurso *"IntelliSense"* fornece autocompletar inteligente com base na análise de código, definições de tipo e documentação (MICROSOFT, 2019).

- *IntelliJ IDEA*: Desenvolvido pela *JetBrains*, é conhecido por sua eficiência em desenvolvimento Java, embora suporte várias outras linguagens. Ele possui uma série de recursos para aumentar a produtividade, como análise de código em tempo real e uma ampla gama de refatorações automatizadas (JETBRAINS, 2020).

- Eclipse: Originalmente criado pela IBM e agora mantido pela *Eclipse Foundation*, é um IDE popular, particularmente no desenvolvimento Java. Possui um robusto sistema de *plugins*, permitindo sua expansão para suportar uma multitude de linguagens e ferramentas. Além disso, fornece uma rica seleção de ferramentas de perfilamento e depuração (FOUNDATION, 2018).

  Os IDEs representam um avanço significativo no panorama de ferramentas de desenvolvimento de software, consolidando-se como elementos essenciais para o aumento da produtividade e eficiência dos desenvolvedores.

2.1.3 Low code/No code

Os termos *low code* e *no code* se referem a metodologias de desenvolvimento de software que reduzem ou eliminam a necessidade de escrever código-fonte tradicional, utilizando interfaces gráficas e componentes prontos para criar aplicações. Essas metodologias surgiram como alternativas ao modo tradicional de programação, que demanda tempo, recursos e conhecimento técnico específico. O objetivo é agilizar o processo de desenvolvimento, permitindo que profissionais de diferentes áreas e níveis de experiência possam criar soluções digitais para seus problemas e necessidades (CHEN; BABAR; BABAR, 2020).

O termo *low code* foi criado em 2014, de acordo com (Clay Richardson, John Rymer, 2014) para designar plataformas que tinham interfaces de desenvolvimento baseadas em gráficos (GUI, do inglês: *Graphical User Interface*). Assim, elas permitem um trabalho de codificação tradicional sem a necessidade de conhecer explicitamente a linguagem de programação em si. O *low code* oferece componentes prontos para o desenvolvedor utilizar em sua aplicação, mas também permite a customização e a criação de novos elementos se necessário. O *low code* é destinado principalmente a grandes corporações e médias empresas que têm alta demanda no desenvolvimento de soluções e equipes de TI para agilizar os processos (CHEN; BABAR; BABAR, 2020; KULKARNI et al., 2020).

O termo *no code* surgiu um pouco depois do *low code*, como uma evolução da mesma ideia. O *no code* oferece uma redução completa da programação, ou seja, o profissional não precisa escrever uma linha de código sequer, criando softwares por meio de uma

interface com modelos que reúnem várias ações. O *no code* é destinado principalmente a profissionais e empresas que não têm experiência e conhecimento em códigos, com foco específico nos micro e pequenos empreendedores, para reduzir demandas de seus setores de TI (RAHMANIAN; KHORASANI; BAGHERI, 2020).

Essas plataformas de desenvolvimento *low code* e *no code* são ferramentas que permitem a criação de aplicativos, softwares e sites com pouca ou nenhuma codificação. Elas usam interfaces gráficas, blocos lógicos, modelos prontos e interface de programação de aplicativos (API, do inglês: Application Programming Interface) para facilitar e agilizar o processo de desenvolvimento. Essas plataformas têm se tornado cada vez mais populares e importantes para a engenharia de software, pois oferecem diversas vantagens (MONDAY.COM, 2022), como:

- Redução de custos e tempo de desenvolvimento, pois eliminam ou diminuem a necessidade de contratar desenvolvedores especializados e permitem que profissionais de outras áreas possam criar soluções digitais.

- Aumento da produtividade e da inovação, pois permitem que os usuários testem e validem suas ideias rapidamente, sem depender de processos burocráticos ou complexos.

- Melhoria da qualidade e da manutenção dos produtos, pois seguem padrões e boas práticas de desenvolvimento, além de facilitar a integração com outras ferramentas e sistemas.

Existem diversos exemplos de utilização das plataformas *low code* e *no code* na indústria, no comércio e em outros setores (BRASIL, 2021). Algumas delas são:

- monday.com: uma plataforma *no code* que permite criar e gerenciar fluxos de trabalho, projetos, processos e colaboração entre equipes. Ela oferece diversos aplicativos personalizáveis e integráveis com outras ferramentas externas.

- Webflow: uma plataforma *no code* que permite criar e projetar sites intuitivos e funcionais sem a necessidade de código. Ela oferece um sistema de desenvolvimento visual, em que o usuário pode ver o resultado do seu site em tempo real.

- Objective: uma plataforma *low code* que permite criar aplicativos corporativos com foco em gestão ágil, transformação digital e experiência do usuário. Ela oferece uma estrutura de codificação simplificada e integrada com outras tecnologias.

2.2 Trabalhos Relacionados

O desenvolvimento de software passou por uma significativa transformação, principalmente devido à ascensão das plataformas de desenvolvimento *low code* e *no code*. Estas plataformas têm sido o foco de muitas pesquisas e estudos, cada uma destacando diferentes aspectos de seus benefícios e desafios. Ao mergulharmos profundamente na literatura disponível, encontramos padrões e percepções que oferecem uma visão clara da situação atual destas plataformas.

Em um estudo empírico conduzido por (ALAMIN et al., 2021), os autores se aprofundam nas discussões dos desenvolvedores para entender os desafios associados às plataformas de desenvolvimento *low code*. Através de um exame das conversas reais e opiniões dos desenvolvedores, emerge uma imagem contrastante das plataformas *low code*. Por um lado, os desenvolvedores reconhecem e valorizam os benefícios evidentes dessas plataformas. Elas fornecem uma maneira mais rápida de desenvolver software, reduzindo a necessidade de codificação extensiva e, portanto, acelerando o ciclo de desenvolvimento. Esse aceleramento não só permite uma entrega mais rápida mas também abre o mundo do desenvolvimento de software para aqueles que podem não ter habilidades de codificação avançadas.

No entanto, o estudo também revela uma série de desafios e reservas expressas por desenvolvedores experientes. Muitos sentem que as plataformas *low code* podem ser restritivas, limitando a personalização e a flexibilidade, aspectos essenciais para desenvolver soluções que atendam a necessidades específicas e complexas. Além disso, enquanto a curva de aprendizado inicial pode ser menor, alguns desenvolvedores apontam que, para dominar e extrair o máximo dessas plataformas, especialmente as mais sofisticadas, ainda é necessária uma quantidade significativa de treinamento e aprendizado.

A escalabilidade também é uma área de preocupação. Enquanto as soluções *low code* podem ser perfeitamente adequadas para aplicações menores e menos críticas, há dúvidas sobre sua capacidade de escalar e atender às demandas de aplicativos empresariais robustos.

Por outro lado, (KHORRAM; MOTTU; SUNYÉ, 2020) conduzem uma análise criteriosa sobre o panorama de testes dentro do ambiente das plataformas de desenvolvimento *low code*. Ao analisar este domínio emergente, fica claro que, assim como o desenvolvimento *low code* revolucionou a maneira como concebemos e construímos aplicações, ele também trouxe à tona novos paradigmas para a fase de teste dessas aplicações. O principal benefício identificado é a simplificação do processo de teste. Muitas plataformas *low code* vêm equipadas com ferramentas integradas de teste que visam facilitar e automatizar a validação de aplicações. Essas ferramentas permitem uma rápida identificação e resolução de *bugs*, reduzindo assim o tempo tradicionalmente necessário para testes manuais extensos.

Esta abordagem integrada promete alinhar-se com o ritmo acelerado de desenvolvimento que essas plataformas promovem, proporcionando uma entrega mais ágil e eficiente.

Entretanto, essa revolução não está isenta de desafios. O primeiro e mais notável é a profundidade dos testes automatizados fornecidos por plataformas *low code*. Enquanto a automação pode cobrir casos de teste padrão, ela pode não ser suficientemente abrangente para garantir a robustez de aplicações complexas ou críticas para o negócio. Outro desafio é a portabilidade dos testes. Em um ambiente em que a dependência do fornecedor já é uma preocupação, garantir que os testes e as metodologias de teste possam ser transferidos ou adaptados entre diferentes plataformas *low code* é fundamental.

(LUO et al., 2021) nos oferece um olhar valioso sobre os profissionais que lidam diariamente com essas plataformas. Os autores se propõem a explorar o mundo do desenvolvimento *low code* através das lentes daqueles que estão no cerne da indústria - os praticantes. Ao fazê-lo, proporciona-se uma visão das realidade vivenciada por aqueles que adotam e trabalham com plataformas *low code* diariamente.

Dentro das características elogiadas, a capacidade de prototipagem rápida se destaca proeminentemente. Praticantes valorizam o modo como as plataformas *low code* permitem uma resposta ágil e adaptativa às demandas em constante evolução do ambiente empresarial. Há também uma apreciação substancial pela inclusividade que tais plataformas oferecem, ao permitir que indivíduos sem uma profunda formação técnica em programação participem ativamente do processo de desenvolvimento. Isso, para muitos, representa uma democratização significativa do desenvolvimento de software.

Entretanto, essa perspectiva enraizada dos praticantes também revela desafios intrincados. Um ponto crítico de discussão é a segurança. Em um mundo cada vez mais digital, as preocupações com a segurança dos dados e a integridade das aplicações estão no auge. Muitos praticantes sentem que, enquanto o desenvolvimento *low code* pode ser rápido e eficaz, pode não estar à altura das rigorosas demandas de segurança de algumas organizações, especialmente em setores altamente regulamentados. Além disso, a questão da sustentabilidade e manutenção a longo prazo das soluções desenvolvidas em plataformas *low code* surge como um desafio palpável. Muitos praticantes estão cautelosamente preocupados se tais soluções podem resistir ao teste do tempo e evoluir conforme necessário.

Em (PRINZ; RENTROP; HUBER, 2021), é apresentado uma revisão literária profunda sobre plataformas de desenvolvimento *low code*, o panorama atual de ferramentas projetadas para simplificar o desenvolvimento de software é examinado minuciosamente. O potencial destas plataformas é evidente: elas democratizam o processo, abrindo portas para participantes que não possuem uma formação técnica tradicional e prometendo tempos de desenvolvimento mais rápidos. A grande vantagem dessas plataformas reside na sua capacidade de permitir a prototipagem e implementação rápidas, respondendo de maneira ágil às necessidades do mercado e dos usuários.

No entanto, o estudo também destaca desafios significativos associados a essa abordagem. Uma preocupação predominante é a dependência de fornecedores, uma vez que muitas destas plataformas operam dentro de seus próprios ecossistemas fechados. Essa natureza proprietária pode dificultar a migração ou integração com outras soluções e tecnologias. A qualidade do software produzido através destas plataformas também é um tópico de debate. Enquanto a rapidez é uma vantagem, há um reconhecimento da necessidade de garantir que essa velocidade não comprometa a robustez, a segurança ou a eficiência do software gerado.

O (KRISHNARAJ et al., 2022) amplia nossa compreensão sobre a eficácia na integração dos processos de negócios. Os autores conduzem um exame rigoroso das diversas plataformas de Gestão de Processos de Negócios (BPM) baseadas em *low code* disponíveis no mercado contemporâneo. Esta análise comparativa oferece uma visão panorâmica, porém detalhada, das ferramentas em questão, posicionando-se como um recurso inestimável para instituições e indivíduos que navegam neste domínio em crescimento.

Entre os benefícios destacados, a capacidade de implementação rápida e ágil é unanimemente reconhecida. Plataformas de BPM *low code*, de acordo com o estudo, proporcionam uma integração mais suave, adaptabilidade às mudanças organizacionais e a capacidade de desenvolver e otimizar processos de negócios com velocidade sem precedentes.

No entanto, essa análise comparativa não se esquiva dos desafios associados. A questão da personalização emerge como uma preocupação principal. Enquanto plataformas de BPM *low code* são projetadas para serem amplamente aplicáveis a uma variedade de cenários, elas podem enfrentar limitações quando confrontadas com necessidades de negócios altamente específicas ou nichadas. Além disso, a dependência e a escalabilidade são abordadas. As empresas estão cautelosamente cientes dos riscos de se tornarem excessivamente dependentes de uma única solução, e há questionamentos sobre se tais plataformas podem escalar adequadamente à medida que uma organização cresce e seus processos se tornam mais complexos.

Por último, (KäSS; STRAHRINGER; WESTNER, 2023) os autores empreendem uma investigação aprofundada sobre as perspectivas dos profissionais que estão na linha de frente da adoção e implementação de plataformas de desenvolvimento *low code*. Este estudo revelador emerge como um reflexo genuíno das realidades operacionais e estratégicas enfrentadas por esses praticantes.

Inicialmente, os benefícios das plataformas *low code* são ressaltados com clareza. Os praticantes amplamente reconhecem a promessa dessas ferramentas em acelerar o ciclo de desenvolvimento. Há um apreço pela eficiência com que as soluções podem ser prototipadas, desenvolvidas e iteradas, proporcionando uma agilidade comercial valiosa. Além disso, a inclusividade destas plataformas, ao permitir que indivíduos sem formação técnica profunda contribuam para o desenvolvimento, é vista como uma democratização

revolucionária da criação de software.

Dentre os estudos apresentados, esta pesquisa se insere no contexto dos seis artigos analisados ao explorar questões relacionadas às plataformas de desenvolvimento *low code* e *no code*. Além disso, adotou uma abordagem abrangente e metodologicamente sólida para investigar tais plataformas, com foco na otimização da produtividade no processo de desenvolvimento de software. Ao comparar com os artigos, nota-se que esta pesquisa se distingue por sua metodologia multivocal, que integra uma revisão de literatura branca e cinza com a realização de um estudo de caso. Seu principal objetivo está centrado na avaliação das alegações de melhoria substancial na produtividade promovidas pelas plataformas *low code* e *no code*.

Vale ressaltar que, enquanto os autores (ALAMIN et al., 2021) conduziram um estudo empírico focado nas discussões dos desenvolvedores para compreender os desafios associados às plataformas *low code*, (PRINZ; RENTROP; HUBER, 2021) apresentam uma revisão literária abrangente sobre essas plataformas, explorando o panorama das ferramentas destinadas a simplificar o desenvolvimento de software.

Esta pesquisa, assim como os artigos analisados, identifica uma série de vantagens, como abstração de código e automação de tarefas, e desafios, como dependência de plataforma e preocupações com segurança de dados. No entanto, ela complementa as descobertas dos artigos ao apresentar resultados de uma avaliação empírica dessas plataformas, fornecendo uma visão mais ampla sobre os aspectos positivos e negativos das plataformas *low code* e *no code*.

Quanto às conclusões e sugestões para trabalhos futuros, esta pesquisa destaca a importância de avaliações empíricas mais abrangentes e a inclusão de outras plataformas *low code* e *no code*. Esse enfoque reflete o desejo de compreender mais profundamente o impacto dessas plataformas na produtividade e qualidade do software produzido.

Em síntese, esta pesquisa se alinha e complementa os temas e descobertas discutidos nos seis artigos analisados, oferecendo uma contribuição valiosa para o entendimento dessas tecnologias emergentes no desenvolvimento de software, especialmente em relação à produtividade.

Parte III

Metodologia

3 Metodologia

O propósito primordial deste estudo é investigar as plataformas de desenvolvimento *low code* e *no code*, focando especificamente nas suas alegações referentes à produtividade. Para alcançar uma análise abrangente, a pesquisa foi estruturada em duas etapas distintas:

Inicialmente, foi proposto a realização de uma revisão de literatura multivocal para avaliar os benefícios e desafios associados a essas plataformas. Essa revisão englobou tanto a literatura branca, que se refere a documentos disseminados por meios formais como editoras, periódicos e bases de dados acadêmicas, quanto a literatura cinza, que engloba fontes mais informais, como postagens de blogs e documentos técnicos. Este espectro diversificado de fontes visa oferecer um panorama holístico do atual estado da arte e das práticas predominantes nessas plataformas.

Na sequência, a pesquisa adotou uma abordagem empírica por meio de um estudo de caso envolvendo plataformas *low code* e *no code*. Esse estudo envolveu a criação de um aplicativo específico com o propósito de avaliar e confrontar os achados da revisão literária, determinando se tais observações preliminares serão corroboradas ou se encontrarão contraposições na prática.

3.1 Revisão da Literatura

A utilização da Literatura Cinza (LC) como fonte de dados em pesquisas em Engenharia de Software (SE) tem crescido nos últimos anos devido ao aumento do uso de canais de comunicação online por engenheiros de software. No entanto, ainda há uma compreensão limitada de como a pesquisa em SE está aproveitando a LC (KAMEI et al., 2021). Uma Revisão de Literatura Multivocal (RLM) é uma forma de Revisão Sistemática da Literatura (RSL) que inclui a literatura cinza (fontes em blogs e documentos técnicos) além da literatura formal (artigos de periódicos e conferências). As RLMs são úteis tanto para pesquisadores quanto para profissionais, pois fornecem resumos tanto do estado da arte quanto da prática em uma determinada área (GAROUSI; FELDERER; MÄNTYLÄ, 2019).

Sua adoção é justificada, uma vez que a literatura acadêmica sobre esse tema ainda é escassa, pois se trata de uma tendência recente e em rápida evolução. Assim, estender as buscas para literatura cinza permite incluir uma diversidade de vozes e perspectivas sobre o tema em estudo, incluindo fontes de informação que não são publicadas em canais tradicionais.

Essa abordagem pode ser útil para ampliar o escopo e a diversidade das evidências

disponíveis, bem como para identificar lacunas e contradições no conhecimento existente, aumentando a relevância e a representatividade (GAROUSI; FELDERER; MÄNTYLÄ, 2019).

Para a condução da revisão de literatura multivocal foi adotada a metodologia baseada na proposta por (GAROUSI; FELDERER; MÄNTYLÄ, 2019), que apresenta diretrizes para RLM. Essa metodologia, adapta algumas das diretrizes apresentadas no trabalho de (PETERSEN et al., 2008), que expõe diretrizes para Revisão Sistemática de Literatura.

Com base na metodologia proposta por (GAROUSI; FELDERER; MÄNTYLÄ, 2019) estruturamos o nosso processo de condução de revisão de literatura multivocal, conforme apresentado na imagem 1.

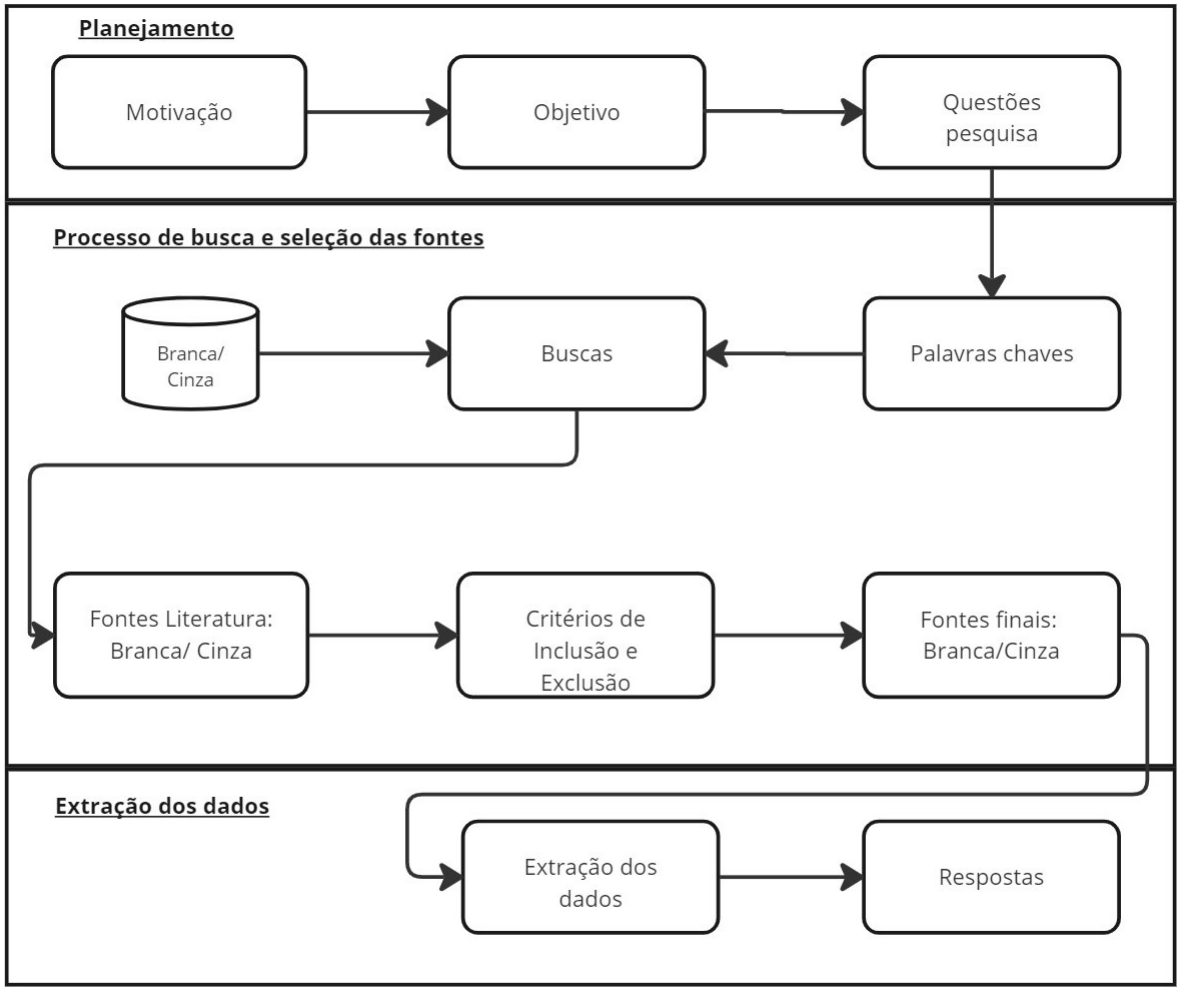


Figura 1 – Metodologia Revisão da Literatura

Nas próximas subseções, são detalhadas as etapas que compõe a metodologia. No segmento de planejamento, delineiam-se a motivação, os objetivos e as questões de investigação científica. No que tange à etapa de busca e seleção, descrevem-se metodologias

adotadas para a construção da *string* de busca, a escolha das bases de dados e os critérios estabelecidos para inclusão e exclusão de fontes. No contexto da extração de dados, elucidam-se os procedimentos metodológicos adotados com o intuito de responder às indagações da pesquisa.

3.1.1 Planejamento

No estágio inicial de planejamento deste estudo, observou-se que o tema é muito ligada a industria de software, e por isso, é preciso ouvir vozes dessa população através de outras estratégias. Portanto, optou-se por conduzir uma revisão de literatura multivocal com o intuito primordial de investigar plataformas de desenvolvimento *low code* e *no code*. O objetivo central foi avaliar o potencial dessas plataformas em promover ganhos de produtividade, identificando simultaneamente seus principais desafios e benefícios. Adicionalmente, buscou-se compreender o estado da arte relacionado ao desenvolvimento e à utilização dessas plataformas no contexto atual.

Para atingir esse objetivo foram levantadas as seguintes Questões de Pesquisa relacionadas à revisão da Literatura (QPL) a serem exploradas:

- ***(QPL1)*** *Quais ferramentas low code e no code são mais populares?*
  Motivação:

  A identificação das ferramentas *low code* e *no code* mais utilizadas pela comunidade de desenvolvimento de softwares é um passo essencial para a realização de estudos mais aprofundados sobre o estado da arte no desenvolvimento e uso de tais ferramentas. Dessa forma, esse questionamento foi levantado tendo como base prover um conhecimento base essencial para o restante do estudo.

- ***(QPL2)*** *Como ferramentas low code e no code são utilizadas?*
  Motivação: O uso de ferramentas *low code* e *no code* vem ganhando popularidade, entretanto é preciso compreender melhor como e em que contexto tais ferramentas são utilizadas. Nessa questão de pesquisa buscamos compreender melhor o contexto de uso das ferramentas *low code* e *no code*.

- ***(QPL3)*** *Quais são os principais benefícios das ferramentas low code e no code?*
  Motivação: Ferramentas *low code* e *no code* prometem facilitar e promover ganhos de desempenho significativos no desenvolvimento de software (LUO et al., 2021; ALAMIN et al., 2021). Com essa questão de pesquisa buscamos identificar se tais promessas são realmente percebidas como verdadeiras pela comunidade de usuários de tais ferramentas, bem como identificar que outros benefícios podem ser acrescidos a essa lista.

- ***(QPL4)** Quais são as limitações das ferramentas low code e no code?*
  Motivação: Limitações das ferramentas representam oportunidades de aprimoramento (LUO et al., 2021; PRINZ; RENTROP; HUBER, 2021). Nessa questão buscamos identificar que limitações são percebidas pela comunidade de desenvolvedores e usuários de ferramentas *low code* e *no code* objetivando identificar oportunidades para o desenvolvimento de pesquisas que auxiliem no aprimoramento das ferramentas existentes.

3.1.2 Busca e Seleção

Esta fase do processo metodológico destinou-se a definição de palavras chaves, mecanismos de buscas, obtenção das fontes e aplicação de critérios de inclusão e exclusão, objetivando garantir fontes de qualidade que respondam adequadamente as questões de pesquisa. Sendo descritas a seguir.

Busca:

Inicialmente, foram identificados os termos que deveriam compor a *string* de pesquisa. Nessa etapa, diferentes combinações e formas similares de escrita das palavras chaves foram propostas e avaliadas. Especificamente foi usado os seguintes termos: *no-code, nocode, no code, low-code, lowcode, low code, challenges, benefits, best practices, pros, cons, limitations, expressiveness, report, case study, tool.*

Como resultado, a seguinte *string* de busca foi utilizada:

(no-code **OR** nocode **OR** “no code” **OR** low-code **OR** lowcode **OR** “low code”) **AND** (challenges **OR** benefits **OR** “best practices” **OR** pros **OR** cons **OR** limitations **OR** expressiveness OR report **OR** “case study” **OR** tool)

3.1.2.1 Seleção - Literatura Cinza

O *Google* foi selecionado como mecanismo de pesquisa. Ele é ideal para realizar as buscas de fontes para uma revisão de literatura cinza, pois abrange uma grande variedade de fontes, incluindo aquelas que não são indexadas em bases de dados tradicionais, como a *Scopus* ou a *Web of Science*. Isso amplia as possibilidades de encontrar documentos relevantes, que muitas vezes são ignorados ou subestimados pelos pesquisadores. Por esses motivos, o *Google* é uma ferramenta indispensável para realizar as buscas de fontes para uma revisão de literatura cinza.

3.1.2.2 Seleção - Literatura Branca

Para pesquisas por literatura branca, procurou-se bases de dados confiáveis e atualizadas, que reúnam as principais fontes de informações científicas na área. Nesse

sentido, as bases de dados IEEE Explore, ACM e Web of Science são ideais, pois cobrem as principais fontes de informação na área de engenharia de software e oferecem recursos que facilitam a busca, a seleção e a avaliação dos documentos. Apresentando conferências, fóruns e eventos relevantes nesse tema como: *International Conference on Low-Code Development Platforms* (LCDP); *International Workshop on No Code Software Development* (NCSD).

3.1.3 Critérios de inclusão e exclusão

A aplicação rigorosa de critérios de inclusão e exclusão desempenha um papel fundamental para assegurar a relevância e a qualidade das fontes selecionadas. Estes critérios são estabelecidos com base em características objetivas e bem definidas, servindo como um filtro que permite aos pesquisadores identificar e selecionar, dentre um vasto universo de fontes disponíveis, aquelas que são mais pertinentes e confiáveis para o propósito do estudo em questão.

Esta seção estabeleceu critérios de inclusão e exclusão visando contribuir para minimizar o risco de viés, garantindo que a seleção não seja arbitrária ou influenciada por fatores externos indesejados. Além disso, ao estabelecer limites claros para a seleção de fontes, os critérios de inclusão e exclusão proporcionam maior replicabilidade ao estudo, uma vez que outros pesquisadores podem, com base nas mesmas diretrizes, realizar revisões ou meta-análises consistentes.

3.1.3.1 Literatura Cinza

Critérios de inclusão:

- Fontes selecionados têm como data mínima o ano de 2017.

  Justificativa: A ideia de construir software sem a necessidade de escrever código não é nova. O desenvolvimento de linguagens de auto nível, ferramentas visuais de programação, ou até mesmo aplicações como planilhas eletrônicas, são passos nesse sentido. Ao definimos uma data miníma, temos como objetivo focarmos nas publicações que se referem aos últimos avanços no desenvolvimento e uso de ferramentas *low code* e *no code*.

- Fontes que apresentem referências claras a temática ou as questões de pesquisas nos títulos.

  Justificativa: O critério de inclusão que prioriza fontes com referências explícitas à temática ou às questões de pesquisa nos títulos é de suma importância para garantir relevância e pertinência no processo de coleta de dados. A presença destas referências nos títulos indica, em geral, uma centralidade do tema na abordagem do documento, assegurando que o conteúdo seja diretamente alinhado aos objetivos desta pesquisa.

Critérios de exclusão:

- Fontes que são apenas propagandas de ferramentas, com cunho comercial.

  Justificativa: por essas ferramentas *low code* e *no code* estarem em evidência, é natural existirem muitas Fontes que são apenas propaganda objetivando vendê-las. Essas fontes por serem publicadas pelos desenvolvedores da ferramenta com o objetivo de promover e divulgar seu uso, podem apresentar um conteúdo enviesado.

- Fontes com conteúdos insuficientes e pouco direcionado às questões de pesquisa.

  Justificativa: muitas fontes tratam superficialmente alguns assuntos sobre ferramentas *low code* e *no code*, não provendo conteúdo que auxilie a responder os questionamentos levantados.

3.1.3.2 Literatura branca

Critérios de Inclusão:

- Artigos selecionados tem como data mínima 2017

  Justificativa: Plataformas de desenvolvimento *low code* e *no code* são relativamente novas, e recentemente tem sido objeto de estudo na academia. Aplicar esse critério evita que se utilize fontes desatualizadas ou obsoletas, que podem conter informações incorretas, incompletas ou superadas pelo avanço do conhecimento científico, também garante-se estar a par das novidades e dos debates mais recentes sobre o tema.

- Artigos cujos títulos apresentem referências claras aos termos *low code* e *no code* .

  Justificativa: A leitura preliminar dos títulos dos artigos permite avaliar se o título do estudo indica que ele aborda o tema, ajudando a evitar viés de seleção, facilitando a replicabilidade e contribuindo para qualidade e a confiabilidade da revisão, pois os estudos incluídos são mais prováveis de responder à questão de pesquisa de forma adequada e rigorosa.

Critérios de Exclusão:

- Síntese

  Justificativa: Artigos que não possuam uma relação direta com as abordagens de desenvolvimento *low code* e *no code* serão excluídos. Isso é essencial para manter o foco e a relevância da análise, garantindo que os achados e conclusões extraídos sejam pertinentes especificamente a esse domínio.

- Escopo

Justificativa: Dada a versatilidade das plataformas de desenvolvimento *low code* e *no code*, observa-se sua aplicação em diversas áreas. No entanto, para assegurar a precisão e qualidade da pesquisa, será imperativo excluir artigos que abordem plataformas de propósito específico não diretamente alinhadas com os princípios fundamentais do desenvolvimento *low code* e *no code*. Tal critério de exclusão visa evitar ambiguidades e garantir que os resultados e discussões estejam intrinsecamente ligados ao *core* da temática em estudo.

3.1.4 Extração de dados

O processo de extração de dados, especialmente em estudos que demandam uma abordagem sistemática, é uma etapa crucial na pesquisa científica. A importância dessa fase reside em sua capacidade de consolidar informações oriundas das fontes selecionadas, garantindo que os dados extraídos sejam precisos, relevantes e representativos do escopo do estudo. Primeiramente, a extração correta de dados assegura que os resultados obtidos na pesquisa sejam baseados em informações fidedignas, minimizando o risco de conclusões imprecisas ou distorcidas. Erros ou inconsistências na extração podem comprometer significativamente a validade dos achados e, por consequência, a contribuição científica do trabalho (ANDRADE, 2017).

Além disso, uma extração bem estruturada facilita a comparação e síntese de dados entre diferentes estudos, possibilitando meta-análises mais robustas e conclusões generalizáveis. Por outro lado, a documentação adequada do processo de extração garante a replicabilidade do estudo, um pilar fundamental da ciência, permitindo que outros pesquisadores validem ou refutem os resultados obtidos. Assim, o ato de extrair dados de maneira sistemática e criteriosa não só eleva o rigor metodológico do trabalho científico, mas também fortalece a integridade e confiabilidade dos resultados apresentados (WAZLAWICK, 2020).

3.1.4.1 literatura Branca/Cinza

Nesta subseção, unifica-se os procedimentos de extração de dados oriundos tanto da literatura branca quanto da cinza. Tal unificação justifica-se pela inexistência de diferenciações notáveis nos métodos de extração entre essas duas categorias de literatura.

1. **Seleção de Fontes**: Uma avaliação rigorosa e criteriosa foi realizada em fontes previamente identificadas, assegurando a relevância e alinhamento com as Questões de Pesquisa de literatura (QPL) estabelecidas.

2. **Triagem e Catalogação**: Fontes que se alinhem às QPLs foram submetidas a uma análise detalhada. Elementos centrais, como título e trechos significativos, foram catalogados e inseridos em uma matriz estruturada para posterior análise.

3. **Leitura Integral e Releitura**: A fim de garantir uma análise exaustiva, cada fonte associada a uma QPL passou por uma leitura completa. Quando ambiguidades ou nuances emergirem, uma releitura concentrada foi conduzida para assegurar a precisão da informação extraída.

3.2 Estudo de Caso

A segunda fase desta proposta consistiu na realização de um estudo de caso focado no desenvolvimento de uma aplicação utilizando plataformas *low code* e *no code*. O objetivo foi avaliar e confrontar os achados da revisão literária, determinando se as observações preliminares serão corroboradas ou se encontrarão contraposições na prática.

No contexto da engenharia de software, a metodologia de estudo de caso é amplamente defendida por (RUNESON; HÖST, 2009), destacando sua importância na compreensão aprofundada dos processos e práticas de desenvolvimento de software. Além disso, (YIN, 2018) afirma que o estudo de caso é uma abordagem essencial na pesquisa empírica, permitindo uma investigação detalhada de fenômenos específicos em contextos reais.

No contexto da avaliação de plataformas de desenvolvimento *low code* e *no code*, a metodologia de estudo de caso desempenha um papel crucial na análise das características, funcionalidades e eficácia dessas plataformas. Como afirmado por (SEAMAN, 1999), o estudo de caso é particularmente adequado para investigar contextos complexos e dinâmicos, como o desenvolvimento de software.

Uma das principais razões para a relevância da metodologia de estudo de caso nesse contexto é sua capacidade de capturar nuances e complexidades que podem não ser adequadamente abordadas por métodos de pesquisa mais quantitativos. Como argumentado por (STAKE, 1995), os estudos de caso permitem uma compreensão profunda dos fenômenos estudados em seu contexto natural, possibilitando achados valiosos sobre os processos e práticas envolvidos.

Além disso, os estudos de caso permitem uma análise comparativa detalhada entre diferentes plataformas de desenvolvimento *low code* e *no code*, destacando suas distintas abordagens, pontos fortes e limitações. Essa compreensão aprofundada é essencial para os tomadores de decisão na seleção da plataforma mais adequada às necessidades e requisitos específicos de suas organizações.

Em suma, a metodologia de estudo de caso oferece uma abordagem robusta e pragmática para avaliar as plataformas de desenvolvimento *low code* e *no code* em um contexto do mundo real, fornecendo achados valiosos para a pesquisa acadêmica, bem como para a prática da engenharia de software e a tomada de decisões estratégicas nas

organizações.

Para condução deste estudo de caso, foi utilizada a metodologia proposta por (RUNESON; HÖST, 2009), que descreve o processo de pesquisa de estudo de caso em cinto etapas:

1. **Desenho do Estudo de Caso**: Nesta etapa, os pesquisadores definem o planejamento do estudo de caso, identificam os objetivos da pesquisa, o caso a ser estudado, perguntas de pesquisa e métodos de coleta. Isso inclui a seleção do caso a ser investigado, a definição de questões de pesquisa e a elaboração de um protocolo de pesquisa.

2. **Preparação para a Coleta de Dados**: Antes de iniciar a coleta de dados, os pesquisadores devem realizar uma preparação cuidadosa. Isso pode incluir a identificação e o recrutamento de participantes, o desenvolvimento de instrumentos de coleta de dados, como entrevistas e questionários, e a definição de procedimentos para garantir a validade e confiabilidade dos dados coletados.

3. **Coleta de Evidências**: Nesta etapa, os pesquisadores coletam dados relevantes para responder às questões de pesquisa definidas anteriormente. Isso pode envolver a realização de entrevistas, observações no local, análise de documentos e outras técnicas de coleta de dados, dependendo do caso específico e dos objetivos da pesquisa.

4. **Análise dos Dados Coletados**: Após a coleta de dados, os pesquisadores realizam uma análise detalhada e sistemática dos dados coletados. Isso pode incluir a identificação de padrões, tendências e temas emergentes, bem como a interpretação dos resultados à luz das questões de pesquisa e do contexto do estudo.

5. **Relatório**: Finalmente, os pesquisadores relatam os resultados do estudo de caso de forma clara e transparente. Isso envolve a apresentação dos dados coletados, a discussão das principais descobertas, a interpretação dos resultados à luz da teoria existente e a conclusão com implicações práticas e recomendações para futuras pesquisas ou práticas.

Na condução deste estudo de caso, é imprescindível destacar a relevância do planejamento, visto que este representa um alicerce essencial para a obtenção de resultados robustos e confiáveis. Nesse contexto, a fase inicial desta metodologia desempenha um papel crucial, ao delinear os diversos elementos primordiais que devem ser considerados para a elaboração de um plano de estudo de caso eficiente.

3.2.1 Planejamento do Estudo de Caso

Este processo de planejamento prossegue com a definição do Objetivo (o que se pretende alcançar?), das Questões de Pesquisa (o que se deseja saber?), da caracterização do caso (o que está sendo estudado?), da escolha do método (como os dados serão coletados?), O perfil do desenvolvedor (Quem desenvolverá?), treinamento (ambiente?) e do protocolo a ser seguido.

3.2.1.1 Objetivo

O objetivo deste estudo de caso é investigar se os benefícios e desafios das plataformas de desenvolvimento *low code* e *no code* se confirmam na prática. Especificamente, busca-se apresentar evidências de ganho de produtividade no desenvolvimento de software ao utilizar essas plataformas. Para alcançar esse objetivo foi elaborado algumas questões de pesquisa:

3.2.1.2 Questões de Pesquisas

- QP1 - Como os benefícios e desafios identificados na literatura em relação às plataformas de desenvolvimento *low code* e *no code* se manifestam na prática?
- QP2 - Os benefícios e desafios associados às plataformas de desenvolvimento *low code* e *no code*, e sua aplicabilidade varia conforme a plataforma utilizada?
- QP3 - De que maneira as constatações dos benefícios e desafios das plataformas *low code* e *no code* sustentam o ganho de produtividade no processo de desenvolvimento de software ao utilizar essas plataformas?

3.2.1.3 O Caso

Para confrontar os achados literários e validar empiricamente sua aplicabilidade, foi desenvolvido um protótipo de aplicação abrangendo a **utilização de banco de dados, interfaces e lógica de negócio**. Este processo visou examinar se os benefícios identificados na literatura, tais como facilidade de uso, prototipagem e aumento de produtividade, bem como os desafios, complexidades na lógica de negócio e requisitos de conhecimento, entre outros aspectos, efetivamente contribuem ou não para o ganho de produtividade no desenvolvimento de aplicações.

Nesse contexto, a seleção das plataformas para desenvolvimento do protótipo do estudo de caso é embasada na análise dos achados literários desta pesquisa, abrangendo tanto fontes de literatura cinza quanto branca. Observa-se que as plataformas de desenvolvimento mais proeminentes compreendem *OutSystems (low code)* e *Bubble (no code)*. Destaca-se, especialmente, a predominância da *OutSystems*, que se posiciona como a mais

mencionada e amplamente utilizada. É crucial ressaltar que características como abordagem de desenvolvimento, escopo do projeto e integração exerceram uma forte influência sobre os autores ao optarem por essas plataformas, conforme evidenciado pelos resultados desta pesquisa, apresentados na Seção 4.

Assim, este estudo de caso foi planejado para averiguar as características das plataformas de desenvolvimento *low code* e *no code*. Como resultado, foi desenvolvida uma aplicação protótipo denominada "Coleta Seletiva de Lixo". Uma descrição do protótipo, histórias de usuários e diagrama de caso de uso, são apresentadas detalhadamente a seguir, fornecendo uma visão abrangente das funcionalidades e objetivos da aplicação.

O sistema descrito envolve três principais atores: **Produtores**, **Coletores** e **Recicladores**. Os **Produtores** são usuários do aplicativo que geram resíduos recicláveis e desejam descartá-los em troca de alguma recompensa. Os **Coletores** são responsáveis por coletar os resíduos dos produtores, registrar as quantidades recolhidas no aplicativo e encaminhar os materiais aos locais de reciclagem. Por fim, os **Recicladores** representam os centros de reciclagem que compram os materiais dos coletores e registram essas transações no aplicativo. Após essa descrição, é apresentada as histórias de usuários, que consistem em descrições curtas e precisas das funcionalidades desejadas do sistema do ponto de vista do usuário final. Além disso, e apresentado o diagrama de caso de uso, representando visualmente as interações entre os diferentes atores e os casos de uso do sistema, oferecendo uma compreensão abrangente das funcionalidades e dos fluxos de trabalho do sistema em questão.

1. Eu, como Produtor, desejo me cadastrar no sistema;
2. Eu, como Produtor, quando tiver uma quantidade de resíduos recicláveis, desejo solicitar, via App, o recolhimento dos meus resíduos, informando quantidades e tipos de resíduos, bem como o coletor;
3. Eu, como Produtor, desejo acompanhar e confirmar minha entrega com os coletores;
4. Eu, como Produtor, quero poder acompanhar meus históricos de entregas;
5. Eu, como Produtor, desejo visualiza o que tenho de ganhos em minha conta do App;
6. Eu, como Coletor, desejo me cadastrar no sistema;
7. Eu, como Coletor, desejo visualizar solicitações para coleta de materiais recicláveis, com suas quantidades e tipos de materiais;
8. Eu, como Coletor, quero poder registrar a entrega a recicladora para Produtor e sua recompensa;

9. Eu, como Coletor, desejo confirmar as quantidades no momento de dar baixa na entrega final junto com o Reciclador;
10. Eu, como Reciclador, desejo me cadastrar no sistema;
11. Eu, como Reciclador, desejo registrar uma entrega feita pelo próprio Coletor;



Figura 2 – Diagrama de Caso de Uso

3.2.1.4 Coleta dos Dados

Como parte essencial do estudo de caso, é crucial definir os dados relevantes e estabelecer o método de coleta. Portanto, foram delineadas as seguintes métricas:

1. **Tempo de Desenvolvimento**: É uma métrica crucial que avalia a eficiência do processo de desenvolvimento de software. Medido em horas, dias ou semanas, o Tempo de Desenvolvimento reflete o período de tempo necessário para concluir uma determinada tarefa de desenvolvimento, desde a concepção até a implementação.
2. **Complexidade do Modelo de Dados**: Métrica essencial, especialmente em projetos que envolvem bancos de dados e estruturas de dados complexas. Medida em unidades arbitrárias, como o número de tabelas, relacionamentos ou entidades, a Complexidade do Modelo de Dados indica o grau de complexidade e abstração presentes no modelo de dados utilizado no desenvolvimento do software. Uma maior complexidade pode indicar desafios adicionais na manutenção e escalabilidade do sistema.

3. **Número de Erros ou Problemas Encontrados**: É uma métrica crítica que reflete a qualidade do software desenvolvido. Medido em termos absolutos ou como uma taxa por mil linhas de código, essa métrica indica quantos erros ou problemas foram identificados durante o desenvolvimento ou após a implementação do software. Um baixo número de erros sugere uma maior qualidade do software, enquanto um número elevado pode indicar a necessidade de revisão e melhorias no processo de desenvolvimento.

3.2.1.5 Perfil do desenvolvedor

O estudo de caso proposto demanda participantes específicos para sua condução eficaz. Os requisitos incluem uma combinação de habilidades técnicas e acadêmicas, bem como experiência prática relevante. Com tal objetivo, foram selecionados dois: o autor deste estudo, atualmente matriculado como mestrando nesta instituição e professor do Instituto Federal do Piauí (IFPI), cuja experiência de uma década inclui disciplinas afins, e que será mencionado como **Dev01**; e um discente no nono período do curso de Ciência da Computação na Universidade Federal do Piauí, que detém experiência como estagiário júnior em Java, com foco principal em atividades de desenvolvimento backend, e será mencionado como **Dev02**.

3.2.1.6 O Treinamento

Como forma de nivelamento dos participantes do estudo de caso, foi promovido um treinamento preliminar antes de iniciar o desenvolvimento, o qual foi planejado e coordenado pelo autor deste estudo. Este treinamento foi estruturado em consonância com a documentação disponibilizada nos próprios sítios das plataformas em questão, englobando diversos recursos como textos explicativos, vídeos instrucionais e trilhas de aprendizado. Desta forma, o treinamento foi conduzido fazendo uso das próprias plataformas de desenvolvimento *low code* e *no-code*, com uma abordagem inicial focada na *low code*, priorizando o desenvolvimento do protótipo. Durante este processo, foram exploradas as funcionalidades relacionadas à manipulação em bases de dados, à criação de interfaces e à lógica de negócios pertinentes ao contexto do estudo.

3.2.1.7 O Protocolo

O presente protocolo teve como objetivo primordial assegurar a padronização mediante a definição clara das etapas, procedimentos e critérios a serem adotados na condução do estudo de caso, e possibilitando que outros pesquisadores repliquem o referido estudo. Além disso, buscou-se contribuir para a manutenção da consistência na coleta, análise e interpretação dos dados. Em última análise, o protocolo se destina a organizar de

forma sistemática o processo de estudo de caso, desde a seleção do caso até a apresentação dos resultados.

A coleta de dados foi realizada por meio de formulário específico desenvolvido para esta finalidade (Apêndice A), com o propósito de armazená-los de maneira adequada, e seguir rigorosamente as métricas definidas na subseção 3.2.1.4 e possibilitar sua análise e interpretação dos dados.

As plataformas utilizadas para o desenvolvimento foram a OutSystems (*low code*) e a Bubble (*no code*), conforme descrito na seção 3.2.1.3. O protótipo delineado na mesma seção, foi inicialmente desenvolvido na plataforma *low code*, com o propósito de avaliar a eventual necessidade de incorporar programação tradicional. Posteriormente, o protótipo foi replicado na plataforma *no code*.

O protótipo, foi desenvolvido seguindo o que foi definido na Seção 3.2.1.3. Dessa forma, os participantes implementaram as histórias de usuários, caso de uso e demais direcionamentos, que foi prática essencial para o desenvolvimento deste projeto. Essa condução foi realizada em etapas, conforme segue:

- Na primeira etapa do desenvolvimento do protótipo, uma reunião foi conduzida para delinear os objetivos do projeto, resultando na decisão de priorizar a construção das bases de dados, alinhando-as às necessidades dos usuários identificados. Consequentemente, as tabelas essenciais para o protótipo foram concebidas, enquanto o tempo de desenvolvimento, a complexidade da estrutura das bases de dados e quaisquer problemas ou erros encontrados foram registrados. Ao término desta etapa, uma reunião de revisão foi conduzida entre os participantes, visando esclarecer dúvidas e avaliar a conformidade das bases de dados com o estudo de caso.
- Posteriormente, como segunda etapa, uma nova reunião foi organizada para determinar a continuidade do desenvolvimento, resultando na decisão de elaborar a interface gráfica e integrá-la às bases de dados. Em seguida, uma reunião adicional foi realizada para deliberar sobre a melhor abordagem para aproveitar os recursos de fluxo de trabalho oferecidos pelas plataformas, especialmente considerando as nuances da plataforma *low code*, que oferecem múltiplas direções a seguir. Com a conclusão desta etapa, uma nova reunião de avaliação foi conduzida.
- Como etapa final, uma reunião foi realizada para definir os perfis de usuário para o protótipo. Os perfis identificados no estudo de caso (Produtor, Coletor e Reciclador) foram implementados conforme necessário, e as respectivas regras de acesso foram atribuídas as interfaces. Por fim, uma reunião de avaliação e teste do protótipo foi conduzida para averiguar suas funcionalidade.

Instrumentos de pesquisas

Durante o processo de desenvolvimento do protótipo, foi de suma importância conduzir pesquisas destinadas à resolução de incertezas, preferencialmente utilizando os recursos disponíveis nos próprios sites das plataformas de desenvolvimento selecionadas. Para o ambiente *Bubble*, pode-se acessar o manual disponível em: documentation bubble.io. Para o ambiente Outsystems, recomendou-se consultar a documentação disponível em: documentation Outsystems.

Contudo, na eventualidade de não ter ser sido possível esclarecer as dúvidas por meio dessas fontes oficiais, foi admissível realizar buscas em outras fontes não oficiais. Todas as dúvidas, tanto as solucionadas pelos meios oficiais quanto aquelas que necessitaram de outras fontes, foram devidamente registradas no Formulário de Coleta de Dados (Apêndice A).

Diretrizes para análise dos dados

1. Tempo de Desenvolvimento:

• Analisar os dados relativos ao tempo total gasto em cada fase do desenvolvimento da aplicação, incluindo design da interface, lógica de negócios e modelagem de dados.

2. Complexidade do Modelo de Dados:

• Avaliar a estrutura do modelo de dados por meio de métricas como o número de entidades, atributos e relacionamentos.

• Utilizar análise quantitativa para determinar a complexidade dos relacionamentos entre os componentes do modelo de dados.

3. Problemas Encontrados:

• Analisar os problemas documentados durante o desenvolvimento da aplicação, categorizando-os de acordo com sua natureza e impacto.

• Realizar uma análise de causa raiz para entender as origens dos problemas encontrados e determinar se são específicos das plataformas *low code/no code* ou se também seriam encontrados no desenvolvimento tradicional.

4. Dúvidas durante o Desenvolvimento:

• Analisar os registros de dúvidas levantadas durante o desenvolvimento da aplicação, avaliando sua natureza e complexidade.

Parte IV

Parte Final

4 Resultados

Este capítulo é dedicado à apresentação e análise dos resultados obtidos ao longo desta pesquisa. As subseções seguintes apresentam os achados da revisão de literatura cinza, revisão de literatura branca e o estudo de caso.

4.1 Revisão da Literatura Cinza

Este capítulo é dedicado à apresentação e análise dos resultados obtidos ao longo desta pesquisa. As subseções seguintes apresentam os achados da revisão de literatura cinza, revisão de literatura branca e o estudo de caso.

4.1.1 Busca e Seleção

Conforme delineado na Seção 3.1.2, intitulada "Busca e Seleção", procedeu-se à investigação empregando os termos especificados na *string* de busca. Para a constituição do *dataset* preliminar, priorizou-se a inclusão dos 150 primeiros resultados obtidos. A plataforma de pesquisa adotada para esta etapa foi o *Google*, amplamente reconhecida na comunidade acadêmica por sua abrangência e credibilidade em literatura científica e cinza. Os resultados colhidos, após uma análise preliminar, foram sistematicamente organizados e exportados para uma planilha, a qual detalha metadados cruciais, incluindo o link da fonte original, o autor e o ano de publicação. É imperativo destacar que tais procedimentos de pesquisa foram conduzidos no intervalo dos meses de junho a julho de 2022, contemplando literaturas nos idiomas inglês e português.

A rigorosidade na adoção das diretrizes estabelecidas na Seção 3.1.2, e a escolha do *Google* como mecanismo de busca asseguram a qualidade e relevância dos dados coletados. A decisão de selecionar os 150 primeiros resultados visa capturar um volume de dados gerenciável, enquanto ainda mantém uma representação significativa da literatura disponível. A posterior organização dos dados em uma planilha estruturada facilita a análise subsequente e a transparência do processo, garantindo, assim, que a pesquisa esteja alinhada aos padrões da metodologia científica e contribua de maneira significativa para o campo de estudo em questão.

Após a coleta, deu-se início ao processo de seleção, onde os critérios de inclusão foram aplicados em um esforço contínuo para garantir a relevância e atualidade das fontes selecionadas através do mecanismo de busca *Google*, foi estabelecido um processo rigoroso para avaliar e filtrar o conjunto inicial de 150 fontes. Esse processo foi regido por critérios de inclusão específicos:

- 1. Data de Publicação: Para garantir a atualidade das fontes, foi definido que apenas os documentos publicados a partir do ano de 2017 seriam considerados relevantes para esta seleção. Após a aplicação deste critério, foram mantidos 124 documentos, resultando na exclusão de 26 fontes que não atendiam a essa especificação.

- 2. Relevância do Título ao Tema de Pesquisa: Dado o vasto número de publicações disponíveis, foi essencial identificar fontes que, claramente, faziam referência à temática ou às questões de pesquisa em seus títulos. Esse critério adicional assegurou que os documentos selecionados fossem altamente pertinentes ao objetivo da pesquisa. Depois de aplicado este critério, o conjunto foi reduzido para 93 fontes, com a exclusão de 31 fontes cujos títulos não demonstravam clara alusão ao tema.

Tabela 1 – Resumo da Seleção de Fontes - critérios inclusão.

Critério de Inclusão	**Fontes Restantes**	**Fontes Excluídas**
Total inicial de fontes	150	-
Publicação a partir de 2017	124	26
Títulos com referências claras à temática de pesquisa	93	31

Este processo de filtragem garante que as fontes selecionadas para a pesquisa sejam não só atuais, mas também altamente relevantes para a temática em estudo. A adoção desses critérios visa maximizar a qualidade e precisão das informações coletadas e, por consequência, dos resultados obtidos a partir desta pesquisa.

Após a implementação dos critérios de inclusão, onde um conjunto inicial de fontes foi refinado para garantir sua atualidade e relevância direta ao tema da pesquisa, foi necessário seguir com a etapa subsequente do processo de seleção. Esta etapa envolveu a aplicação de critérios de exclusão com o objetivo de refinar o conjunto inicial e garantir a qualidade e pertinência das informações a serem estudadas. A partir de um conjunto inicial de 93 fontes identificadas, foi realizada uma avaliação detalhada para assegurar a conformidade destas fontes com os padrões estabelecidos. Esse processo foi regido por critérios de exclusão específicos:

- 1. Natureza Comercial da Fonte: O primeiro critério visou eliminar quaisquer fontes que tivessem uma orientação estritamente comercial, ou seja, fontes que se limitavam a ser propagandas de ferramentas ou possuíam cunho primariamente comercial. Através da aplicação deste critério, 26 fontes foram identificadas e excluídas, resultando em um total de 67 fontes remanescentes.

- 2. Qualidade e Relevância do Conteúdo: Posteriormente, foi avaliada a substância e direcionamento do conteúdo de cada fonte em relação às questões de pesquisa. Fontes que apresentaram conteúdos insuficientes ou que não estavam alinhadas diretamente

ao tema da pesquisa foram descartadas. Após esta análise, 17 fontes adicionais foram removidas, deixando 50 fontes qualificadas para o estudo.

Tabela 2 – Resumo da Seleção de Fontes - critérios exclusão.

Critério de Exclusão	**Fontes Restantes**	**Fontes Excluídas**
Total inicial de fontes	93	-
Fontes com natureza comercial	67	26
Conteúdos insuficientes ou não alinhados à pesquisa	50	17

Em resumo, a aplicação desses critérios de exclusão resultou na eliminação de 43 fontes do conjunto inicial. Esta seleção visa garantir que o conteúdo estudado seja relevante, atual e de alta qualidade, proporcionando assim uma base sólida para os resultados da pesquisa.

A aplicação dos critérios de inclusão e exclusão fortaleceu significativamente a qualidade e relevância das fontes selecionadas para esta pesquisa. Ao enfocar na atualidade e na pertinência direta ao tema, e posteriormente eliminar fontes de natureza comercial e aquelas de conteúdo insuficiente, foi possível criar uma base sólida e focada para a análise. Estes achados, rigorosamente curados, são cruciais para responder às questões de pesquisa com precisão e profundidade. Através deste processo, foi garantido não apenas a validade das informações utilizadas, mas também a integridade e robustez dos resultados e conclusões derivadas desta investigação. A lista completa de documentos selecionadas e utilizadas no estudo está disponível em planilha online[1].

4.1.2 Respostas às Questões de Pesquisa

***QPL1*: Quais são as ferramentas *low code* e *no code* mais populares/adotadas?**

Para responder a QPL1, foi identificado manualmente as ferramentas que foram citadas nas fontes selecionadas. Durante o processo de identificação, se uma ferramenta foi mencionada várias vezes em uma mesma postagem, ela foi contada apenas uma vez. Adicionalmente, vale mencionar que a ferramenta App Maker da Google, apesar de ter sido mencionada quatro vezes, foi descartada da contagem por ter sido descontinuada[2].

As ferramentas mais mencionadas foram **OutSystem**, com 14 ocorrências, **Mendix** e **PowerApps**, com 12 menções cada, seguidas por **Bubble**, **Salesforce** e **Appian**, com nove referências. A listagem das ferramentas mais citadas (pelo menos três vezes) pode ser observada na Tabela 3.

***QPL2*: Como ferramentas *low code* e *no code* são utilizadas?**

[1] <https://docs.google.com/spreadsheets/d/1ejv_q-vzFurJF-uZIXF989O7IFxBiuWaA2kB2HqEay0/edit?usp=sharing.>

[2] <https://workspaceupdates.googleblog.com/2020/01/app-maker-update.html>

Para responder a QPL2, foi realizada uma busca adicional por informações no site das próprias ferramentas e fóruns de discussão, objetivando identificar como essas são utilizadas, tendo como base seu propósito (ferramenta *low code* ou *no code*, se a ferramentar é distribuída *on-premise* ou na *cloud*, aplicação (desenvolvimento de sistemas *mobile*, *desktop* ou *web*) e a linguagens de programação adotadas. Esses resultados são apresentados na Tabela 3.

É possível observar que grande parte das ferramentas concentram seu foco no desenvolvimento de apps **web** (15) e **mobile** (11). O desenvolvimento de aplicações **desktop** é permitido por apenas uma das ferramentas. Embora muitas sejam ferramentas **no code** (6), permitindo o desenvolvimento de aplicações sem a necessidade de escrita de código, a maior parte das ferramentas são **low code** (12).

Olhando para as ferramentas *low code* é possível observar que diversas são as linguagens de programação adotadas, porém se destaca a linguagem **Javascript** (7 de 12 ferramentas *low code*) como a linguagem mais adotada.

Por fim, foi investigada a estratégia de distribuição das aplicações desenvolvidas em cada uma das ferramentas. Em outras palavras, se as aplicações podem ser executadas em ambiente de servidores privado do usuário (*on-premise*) ou são obrigados a utilizarem a infraestrutura de nuvem (*cloud*) da ferramenta.

O modelo em *cloud* é adotado por todas as ferramentas, demonstrando ser o padrão para o desenvolvimento de aplicações que utilizam ferramentas *low code* e *no code*. Entretanto, algumas ferramentas disponibilizam, ainda que em alguns casos apenas em versões pagas, o modelo *on-promise* (8 ferramentas) permitindo a possibilidade de executar as aplicações desenvolvidas na infraestrutura do próprio cliente.

***QPL3*: Quais são os principais benefícios das ferramentas *low-code* e *no-code*?**

Para responder a QPL3, foram feitas leituras nas fonte, objetivando identificar como a comunidade de programadores, profissionais de negócios e usuários iniciantes, percebem os benefícios apresentados por essas ferramentas, sob o ponto de vista de utilização no desenvolvimento de suas aplicações. Das 50 fontes iniciais foi identificado menções a benefícios das ferramentas *low code* e *no code* em 30. Os benefícios identificados, que foram mencionados mais de 3 vezes, e em postagens distintas, foram sumarizados e são apresentados na Tabela 4.

É possível perceber que facilidade de uso, economia de recursos, prototipagem e aumento de produtividade são os benefícios mais mencionados. Esse resultado podem ser compreendido pois essas são as primeiras percepção dos usuários ao utilizarem ferramentas *low code* e *no code*. Por exemplo, *hackernoon* (brocoders, 2021) menciona, “com o *low code*, você pode economizar seu tempo e os recursos da empresa fazendo as coisas mais rapidamente”.

Tabela 3 – Ferramentas *low code* e *no code* identificadas.

Ferramenta	**F**requência	**A**plicação	**T**ipo	**L**inguagens	**D**istribuída
OutSystems	14	Web, Mobile	low-code	C#	on-premise e cloud
Mendix	12	Mobile	low-code	Java, Javascript	on-premise e cloud
PowerApps	12	Web, Mobile	low-code	PowerFX	on-premise e cloud
bubble	09	Web	no-code	no-code	cloud
Salesforce	09	Web, Mobile	low-code	Apex	cloud
Appian	09	Web, Mobile	low-code	Java	cloud
ServiceNow	07	Mobile	low-code	Javascript	on-premise e cloud
QuickBase	06	Web	no-code	no-code	on-premise e cloud
ZohoCreator	06	Web, Mobile	low-code	Java, Javascript	cloud
TrackVia	05	Web, Mobile	no-code	no-code	cloud
Airtable	05	Web	low-code	Javascript	cloud
BettyBlocks	04	Web, Mobile	no-code	no-code	on-premise e cloud
OracleApex	04	Web	low-code	Javascript, Sql	cloud
Thunkable	03	Mobile	no-code	no-code	cloud
Caspio	03	Web	low-code	Javascript, Sql	cloud
AgilePointInc	03	JWeb, Mobile, Desktop	low-code	Java, C, Python, Javascript	cloud
Webflow	03	Web	no-code	no-code	on-premise e cloud
WordPress	03	Web	low-code	PHP, Sql	on-premise e cloud

Tabela 4 – Benefícios das ferramentas *low code* e *no code*.

Benefícios	**Frequência**
Facilidade de uso	30
Economia de recursos	30
Prototipagem	30
Aumento de produtividade	30
Manutenções reduzidas	15
Citezen develop	8
Integração de dados	4

Percebe-se que o benefício integração de dados apresenta a menor frequência, isso não significa que é menos importante, apenas será mais evidente ao usuário quando sua aplicação necessitar de evoluções maiores que impliquem em integrar dados de outras fontes. Segundo *Teleport* (Calin Abaiesitei, 2022), "ferramentas *low code* e *no code* permite a integração de sistemas legados de mainframe".

***QPL4*: Quais são as limitações das ferramentas *low code* e *no code*?**

Para responder a QPL4, foram feitas leituras nas fontes, com o objetivo de observar que desafios a comunidade em geral de utilizadores enfrentam, com um olhar voltado não apenas para o momento em que se adota uma ferramenta para desenvolver uma aplicação, mas na continuidade desse desenvolvimento, por entender-se que, somente com a evolução da aplicação essas limitações começam a ser aparentes. Das 50 fontes, 27 foram selecionas, por apresentarem conteúdos relevantes. As limitações identificadas, mencionadas mais de 3 vezes, e em postagens distintas, foram sumarizados.

Os resultados foram compilados e apresentados na tabela 5

Tabela 5 – Limitação das ferramentas *low code* e *no code*.

Limitações	**F**requência
Dependência de plataforma	27
Requisito de conhecimento	27
Funcionalidade limitada	27
Lógica de negócio	20
Segurança dos dados	15
Dívida técnica	15
Velocidade e escalabilidade	12
Governança	4

Percebe-se que em uma análise geral os resultados com as maiores frequências, com destaque para dependência de plataforma e funcionalidade limitada, expressão as limitações percebidas na fase inicial de adoção dessas ferramentas, sendo perceptível que ao iniciar uma aplicação você estará dependente dessas ferramentas. Segundo *Integrate* (Donal Tobin, 2021), “Os componentes usados para sua aplicação são exclusivos dessas ferramentas e é improvável que você possa migrá-los para outro lugar”.

As de frequências menores, com observação para lógica de negócio, representam a percepção do usuário na continuidade do desenvolvimento da aplicação, essa limitação está relacionada ao quanto sua ferramenta possibilitará o crescimento, a evolução de sua aplicação. Segundo *hackernoon* (brocoders, 2021), “uma vez que o estágio inicial de prototipagem é passado, a lógica de negócios torna-se mais complexa eventualmente. Para desenvolver um projeto ainda maior, você precisará de uma equipe especializada”.

4.2 Revisão da Literatura Branca

Nesta seção, descreve-se os resultados obtidos a partir do processo de busca e seleção, juntamente com os relacionados às questões de pesquisa vinculadas à literatura branca. Na subseção "Busca e Seleção", destaca-se a metodologia da busca, a quantidade de fontes escolhidas e aplicação das diretrizes de inclusão e exclusão. Por sua vez, na subseção "Respostas às Questões de Pesquisa", os principais achados são apresentados.

4.2.1 Busca e Seleção

Conforme delineado na Seção 3.1.2, intitulada "Busca e Seleção", procedeu-se a realização das buscas utilizando a *string* de busca, nas bases de dados IEEE Explore, ACM e Web of Science. Após realização das buscas, foram selecionados todos os resultados, e feita a exportação para arquivo *Comma Separated Values*(CSV), por facilitar a importação em planilha, para compor o *dataset*. A utilização da planilha é justificada para seguir o rigor proposto nesta pesquisa, incluindo os seguintes campos: (título, autor(es), ano, resumo,

tipo de publicação, URL, Publication Title e Abstract). As buscas foram executadas entre junho e julho de 2023, no idioma inglês. Apesar da estratégia de busca ter sido baseada em palavras-chave em inglês, é importante salientar que os resultados obtidos nas respectivas bases de dados abrangiam artigos de diversos idiomas. Dentre estes, publicações em língua portuguesa, quando identificadas, também foram consideradas. Portanto, a análise restringiu-se aos artigos escritos em inglês e português.

Ao fim desse processo obteve-se os seguintes resultados apresentados na tabela 6.

Tabela 6 – Retorno: Literatura Branca

Base	**Quantidade**
IEEE Xplore	138
Web of Science	88
ACM	44

Diante de uma quantidade significativa de artigos, a categorização é fundamental para identificar tendências, facilitar buscas e apoiar revisões bibliográficas. Para tal, foi analisado e categorizado os 270 títulos de artigos para determinar seus temas centrais. A abordagem utilizada é uma adaptação da metodologia proposta por (BRAUN; CLARKE, 2006) para análise temática, que é uma das mais citadas e amplamente reconhecidas no campo da pesquisa qualitativa. É uma metodologia flexível de pesquisa qualitativa que visa identificar, analisar e relatar padrões (temas) dentro dos dados. O processo envolve seis fases, desde a familiarização com os dados até a produção do relatório final, com foco constante na reavaliação e interpretação dos temas emergentes. Entretanto, para esta pesquisa foi necessário uma adaptação, concentrando-se em identificar padrões imediatos nos títulos, utilizando palavras-chave e temas recorrentes para formar categorias. Esta abordagem modificada visa uma análise mais direcionada aos títulos, enquanto mantém a essência da metodologia original.

Abordagem

1. **Análise Inicial**: Inicialmente, procedeu-se com uma leitura rápida de todos os títulos para obter uma compreensão geral dos temas abordados.

2. **Identificação de Palavras-Chave e Temas Recorrentes nos títulos**: Identificou-se palavras-chave e frases que apareciam com frequência, como "*Low Code*", "*IoT*", "*Blockchain*", entre outras. Esta etapa foi crucial para determinar as categorias preliminares.

3. **Criação de Categorias Preliminares**: Com base nas palavras-chave e temas recorrentes, estabeleceu-se categorias preliminares. Por exemplo, títulos contendo a palavra "*Low Code*" foram preliminarmente agrupados sob o tema "Desenvolvimento *Low Code*".

4. **Revisão e Refinamento das Categorias**: Após a categorização preliminar, revisou-se cada título dentro de sua categoria para assegurar que realmente pertencia àquele agrupamento. Títulos ambíguos ou que poderiam se encaixar em mais de uma categoria foram analisados com maior detalhe para determinar sua melhor colocação.

Após a aplicação da abordagem proposta, chegou-se a um total de oito categorias distintas, que refletem as tendências e focos temáticos das amostras analisadas, apresentada na tabela 7

Tabela 7 – Resultados: Artigos Categorizados

Categoria	**Contagem**
Conceitos e Plataformas de Desenvolvimento Low-Code	103
IoT e Sistemas Inteligentes	12
Análise de Código e Estratégias	28
Blockchain e Contratos	7
IA e Machine Learning	14
Desenvolvimento e Engenharia Orientados a Modelos	57
Desenvolvimento No-Code	12
Outros	37

A lista completa dos artigos selecionadas e utilizadas no estudo está disponível em online[3].

Essa categorização desempenha um papel essencial ao proporcionar uma análise aprofundada sobre como os artigos se agrupam em áreas temáticas específicas, o que, por sua vez, possibilita uma avaliação mais precisa da distribuição e relevância dos estudos em diversos campos de pesquisa. Este enfoque não apenas auxilia na identificação de tendências e lacunas no conhecimento, mas também desempenha um papel crítico na otimização da seleção de trabalhos durante o processo de revisão bibliográfica.

Ademais, a categorização representa uma etapa preliminar para a aplicação dos critérios de inclusão e exclusão dos trabalhos selecionados, aprimorando significativamente a eficiência e rigor do processo de seleção de artigos relevantes.

Após a coleta e categorização, deu-se início ao processo de seleção dos artigos, onde os critérios de inclusão e exclusão foram aplicados em um esforço contínuo para garantir a relevância e atualidade das fontes selecionadas, esse processo foi regido por diretrizes delineadas na subseção 3.1.3 intitulada "Critérios de inclusão e exclusão". Iniciado-se pelos critérios de inclusão, conforme segue:

- 1. Para assegurar a contemporaneidade das fontes utilizadas, estabeleceu-se como critério a inclusão dos artigos publicados a partir do ano de 2017. Essa medida visa garantir que os artigos selecionados estejam alinhados com os desenvolvimentos mais

[3] <https://1drv.ms/x/s!AoQQ6gRdN6qshL1xXBqiY195A00ywA?e=6sEhut>

recentes na literatura científica. Após a aplicação deste critério, foram mantidos 243 artigos, resultando na retirada de 27 que não atendiam a essa especificação.

- 2. Relevância do Título ao Tema de Pesquisa: Dado o vasto número de publicações disponíveis, foi essencial identificar fontes que, claramente, faziam referência à temática ou às questões de pesquisa em seus títulos. Embora muitos títulos não incluíssem explicitamente os termos-chave *'low code'* e *'no code'*, alguns faziam menção a plataformas específicas, como Mendix, OutSystems, entre outras. Além disso, alguns artigos apresentavam palavras-chave definidas na estratégia de busca, como 'desafios' e 'oportunidades', que se relacionam com diversas áreas temáticas, incluindo Análise de Código e Estratégias, Blockchain e Contratos, Desenvolvimento e Engenharia Orientados a Modelos, entre outras. Embora todos os artigos recuperados estejam associados às palavras-chave empregadas na estratégia de busca, verifica-se que nem todos correspondem diretamente às questões de pesquisa propostas. Nota-se que a área temática 'Desenvolvimento e Engenharia Orientados a Modelos', que se dedica à metodologia de desenvolvimento de software com foco na criação e utilização de modelos de domínio, apresenta um distanciamento em relação às questões de pesquisa estabelecidas. Esta observação é igualmente aplicável às áreas de '*Blockchain* e Contratos' e 'Análise de Código e Estratégias'. Em consequência, decidiu-se excluir os artigos enquadrados nessas categorias temáticas. Após aplicação desse critério, o conjunto de artigos foi reduzido para 151, implicando na retirada de 92. Conforme apresentado na tabela: 8

Tabela 8 – Resumo da Seleção de Artigos - critérios inclusão.

Critério de Inclusão	**Fontes Restantes**	**Fontes Excluídas**
Total inicial de fontes	270	-
Artigos como data mínima 2017	243	27
Artigos cujos títulos apresentem referências claras aos termos low code e no code	151	92

Após implementar os critérios de inclusão, que refinaram um conjunto inicial de fontes para garantir sua atualidade e relevância direta ao tema da pesquisa, prosseguiu-se com a etapa seguinte do processo de seleção. Essa etapa consistiu-se na aplicação de critérios de exclusão com a finalidade de aprimorar o conjunto inicial e assegurar a qualidade e a pertinência das informações a serem analisadas. De um conjunto inicial de 151 fontes identificadas, fez-se uma avaliação para verificar a conformidade dessas fontes com as diretrizes definidas nos critérios de exclusão, conforme segue:

- 1. Síntese: O método de exclusão adotado consistiu na remoção de artigos que não mantivessem uma conexão direta com as abordagens de desenvolvimento *low code* e *no code* apresentando suas característica, benefícios ou desafios. Dessa maneira,

mediante uma análise dos 151 artigos inicialmente considerados, 42 foram descartados por não atenderem a esse critério, culminando, portanto, em uma amostra final composta por 109 artigos.

- 2. Escopo: Diante da abrangente aplicação das plataformas de desenvolvimento *low code* e *no code*, nota-se sua utilização em diversas áreas. No entanto, com o propósito de assegurar a precisão e a qualidade da pesquisa em foco, tornou-se imperativo proceder à exclusão de artigos que abordassem plataformas de propósito específico, não alinhadas diretamente aos princípios fundamentais do desenvolvimento low code e no code, e que expressassem suas características, benefícios ou desafios. Este processo de exclusão foi aplicado a um conjunto inicial de 109 artigos, culminando na remoção de 47 fontes que não satisfizeram as diretrizes estabelecidas. Após essa etapa, subsiste um conjunto composto por 62 artigos, os quais serão utilizados para responder às questões de pesquisa propostas. A tabela 9 apresenta o resultado da execução dos critérios de exclusão.

Tabela 9 – Resumo da Seleção de Artigos - critérios exclusão.

Critério de Exclusão	**Fontes Restantes**	**Fontes Excluídas**
Total inicial de fontes	151	-
Síntese	109	42
Escopo low code e no code	62	47

4.2.2 Respostas às Questões de Pesquisa

***QPL1*: Quais são as ferramentas *low code* e *no code* mais populares/adotadas?**

Para responder a QPL1, foi iniciado um processo de leitura e catalogação das plataformas de desenvolvimento *low code* e *no code* mais utilizadas nos artigos. culminando em um conjunto de 11 plataformas.

Os achados da presente análise evidenciam que as três plataformas frequentemente mais utilizadas são *OutSystems, Mendix* e *powerApps*, com 11, 9 e 8 respectivamente. É relevante destacar que a plataforma *Google AppSheet* destaca-se como uma das mais adotadas , assumindo o papel de alternativa à plataforma *App Maker*, que foi descontinuada pela própria *Google*, e citada neste pesquisa na seção 4.1.2 intitulada "Resposta as Questões de Pesquisa" investigadas na literatura cinza.

Vale ressaltar, adicionalmente, que essas plataformas demonstraram aplicabilidade em variados domínios de negócio, como saúde, educação e indústria. A integralidade dos resultados encontra-se detalhada na Tabela 10.

***QPL2*: Como as ferramentas *low code* e *no code* são utilizadas?**

Para responder à QPL2, foi realizado uma busca complementar por informações nos sites das plataformas, uma vez que os artigos científicos não investigam todas essas funcionalidades. O objetivo foi identificar a utilização dessas plataformas considerando sua proposta (*low code* e *no code*), seu modo de distribuição (*on-premise* ou *cloud*), seu tipo de aplicação (*mobile, desktop* ou *web*) e suas linguagens de programação. Os resultados são apresentados na tabela 10.

É evidente que todas as plataformas oferecem a capacidade de desenvolver aplicações Web, destacando a necessidade imperativa dessas plataformas no cenário do desenvolvimento Web. Outro cenário que frequentemente se destaca é o *mobile*, com (8) das (11) plataformas analisadas. Esse dado evidencia um notável crescimento e uma significativa adoção por parte dessas plataformas no cenário de desenvolvimento móvel.

Um dado notável, revelado na tabela no campo 'tipo', destaca que a proposta predominante entre as plataformas *low code* e *no code* é ser *low code*. Embora muitos artigos mencionem ambas as propostas, a adoção acaba sendo predominantemente *low code*, visto que, das (11) plataformas, apenas (3) são classificadas como *no code*. As linguagens de programação adotadas por plataformas *low code* estão intimamente relacionadas às mais utilizadas no desenvolvimento tradicional, com (*Java* e *JavaScript*) se destacando como predominantes. Como última análise dos resultados da tabela, observa-se que todas as (11) plataformas adotam a distribuição em nuvem (*Cloud*). No entanto, é interessante notar que (4) das (11) plataformas oferecem um modelo híbrido, proporcionando aos usuários a flexibilidade de escolher entre *Cloud* e *on-premise*.

Tabela 10 – Plataformas *low code* e *no code* identificadas.

Plataformas	**F**requência	**A**plicação	**T**ipo	**L**inguagens	**D**istribuída
OutSystems	11	Web, Mobile	low-code	C#	on-premise e cloud
Mendix	09	Mobile, Web	low-code	Java, Javascript	on-premise e cloud
PowerApps	08	Web, Mobile	low-code	PowerFX	on-premise e cloud
bubble	08	Web	no-code	no-code	cloud
Appian	07	Web, Mobile	low-code	Java	cloud
Google AppSheet	06	Web, Mobile	Low-code	AppSheet Expression	cloud
Salesforce	06	Web, Mobile	low-code	Apex	cloud
QuickBase	05	Web	no-code	no-code	on-premise e cloud
ZohoCreator	03	Web, Mobile	low-code	Java, Javascript	cloud
Oracle Apex	03	Web	low-code	Javascript, Sql	cloud
App gyve	02	Web, Mobile	no-code	no-code	cloud

***QPL3*: Quais são os principais benefícios das ferramentas *low code* e *no code* ?**

Com o propósito de abordar a QPL3, realizou-se análises aprofundadas nos artigos, visando identificar a percepção dos autores em relação aos benefícios proporcionados por essas plataformas. Ressalta-se a importância de salientar que a síntese desses benefícios foi conduzida com a observação das formas de menção e sua frequência. Os resultados podem ser observados na tabela 11

Tabela 11 – Benefícios das ferramentas *low code* e *no code*.

Benefícios	**Frequência**
Tempo de desenvolvimento reduzido	31
Produtividade aumentada	29
Acessibilidade a um público mais amplo	18
Redução de custos	12
Flexibilidade	6
Segurança	5
Personalização	4

Conforme evidenciado na tabela, destaca-se que o benefício frequentemente mais mencionado é a redução do tempo de desenvolvimento. As plataformas de desenvolvimento *low code* e *no code* proporcionam aos desenvolvedores a capacidade de criar aplicativos de maneira ágil e descomplicada, demandando pouco ou nenhum domínio em codificação (MARTINS et al., 2020). Tal dinâmica resulta em uma considerável diminuição no período dedicado ao desenvolvimento. Em segundo plano, figura o aumento da produtividade como benefício destacado. Essas plataformas automatizam tarefas repetitivas e disponibilizam ferramentas e recursos que simplificam o processo de criação de aplicativos (ARORA; GHOSH; MONDAL, 2020).

O terceiro benefício mais citado refere-se à ampliação da acessibilidade a um público mais abrangente. As plataformas *low code* e *no code* propiciam uma democratização do desenvolvimento de aplicativos, tornando-o mais acessível, inclusive, a indivíduos não familiarizados com programação (JUHáS et al., 2022).

Outros benefícios comumente apontados são a redução de custos, flexibilidade, segurança e personalização. Cabe salientar que a natureza e a magnitude desses benefícios podem variar em conformidade com a plataforma escolhida e as exigências específicas do projeto em questão.

***QPL4*: Quais são as limitações das ferramentas *low code* e *no code* ?**

Com o intuito de explorar a QPL4, realizou-se análises abrangentes nos artigos, buscando identificar a perspectiva dos autores em relação às limitações ou possíveis dificuldades associadas a essas plataformas. Embora a maioria dos artigos se dedique à investigação da aplicação dessas plataformas em suas respectivas questões de pesquisa, priorizando a avaliação da aplicabilidade em detrimento da abordagem de desafios ou possíveis limitações, foi possível extrair, por meio dessas análises, algumas considerações sobre as limitações. Os resultados podem ser observados na tabela 12

As plataformas de *low code* e *no code* oferecem benefícios notáveis em tempo de desenvolvimento, produtividade, acessibilidade etc. mas enfrentam desafios que impactam a qualidade, segurança e sustentabilidade das soluções (KASS; STRAHRINGER; WESTNER, 2022). Desafios incluem altos custos, especialmente para pequenas e médias empresas,

Tabela 12 – limitações das ferramentas *low code* e *no code*.

Limitações	**Frequência**
Dependência do fornecedor	11
Alto custo	7
Complexidade de integração	7
Limitação de recursos	5
Limitação de dados	4
Baixa personalização	4
Baixa escalabilidade	3

complexidade de integração com outras ferramentas, dependência do fornecedor, baixa personalização, limitações de dados, recursos insuficientes para soluções complexas e baixa escalabilidade.

O custo elevado de muitas dessas plataformas pode inviabilizar projetos com orçamento limitado, enquanto a complexidade de integração pode demandar esforços adicionais de programação ou contratação de terceiros (GOMES; BRITO, 2022). A dependência do fornecedor é agravada pela falta de capacidade de exportação ou migração das soluções desenvolvidas. A baixa personalização e as limitações de dados podem restringir a criatividade, enquanto recursos insuficientes e baixa escalabilidade podem ser obstáculos para soluções mais complexas.

4.2.3 Considerações Finais

As respostas obtidas às questões de pesquisa, provenientes da literatura cinza e branca, oferecem uma visão abrangente sobre as plataformas de desenvolvimento *low code* e *no code*. Ao explorar as ferramentas mais populares, destaca-se a *OutSystems, Mendix* e *PowerApps* como líderes, corroborando tanto a perspectiva de blogs e sites quanto a de artigos científicos.

Quanto à utilização dessas ferramentas, observa-se uma concentração no desenvolvimento de aplicações web e móveis, com uma ênfase predominante em ferramentas *low code*. A linguagem *JavaScript* emerge como uma escolha comum, enquanto a distribuição em nuvem é praticamente ubíqua, refletindo o padrão para o desenvolvimento de aplicações nesta categoria. Destaca-se ainda a disponibilidade de modelos híbridos, permitindo aos usuários escolherem entre ambientes *on-premise* e em nuvem.

Os benefícios identificados nas fontes abordam a redução do tempo de desenvolvimento, aumento da produtividade e ampliação da acessibilidade, enfatizando o impacto positivo dessas plataformas na eficiência e na democratização do desenvolvimento de software. No entanto, é crucial notar que a percepção desses benefícios pode variar de acordo com a plataforma escolhida e as especificidades do projeto.

Por outro lado, as limitações apontadas incluem desafios financeiros, complexidade

de integração, dependência do fornecedor, baixa personalização e questões de escalabilidade. Esses desafios podem influenciar a qualidade, segurança e sustentabilidade das soluções desenvolvidas, sendo importante considerá-los durante o processo de escolha e implementação dessas ferramentas.

Em síntese, as plataformas de desenvolvimento *low code* e *no code* têm se destacado como ferramentas poderosas, acelerando o ciclo de desenvolvimento e democratizando o acesso à criação de aplicações. No entanto, é essencial que os desenvolvedores e as organizações estejam cientes das limitações associadas a essas ferramentas, buscando equilibrar os benefícios com os desafios específicos de cada contexto.

Estas considerações finais proporcionam uma visão integrada das respostas obtidas, unindo as perspectivas da literatura cinza e branca, e fornecem uma base sólida para a compreensão do panorama atual das plataformas *low code* e *no code* no desenvolvimento de software.

4.3 Estudo de Caso

Nesta seção, apresenta-se uma breve descrição sobre o protótipo, o processo de desenvolvimento adotado, as respostas às questões de pesquisa elaboradas e as considerações finais obtidas a partir dessa análise. O objetivo é fornecer uma visão abrangente das etapas realizadas neste estudo, destacando os principais pontos de investigação e conclusões alcançadas.

4.3.1 Descrição do Protótipo

No processo de desenvolvimento do protótipo, foram conduzidas três etapas distintas. Na primeira fase,focou-se na definição das bases de dados necessárias para o desenvolvimento de protótipo, resultando na criação das tabelas e seus relacionamentos e na identificação de eventuais problemas. Em seguida, a segunda fase concentrou-se na elaboração das interfaces gráficas, seguida por uma discussão sobre o melhor uso do fluxo de trabalho oferecido pelas plataformas. A terceira e última fase foi direcionada à implementação dos perfis de usuário, refletindo os papéis identificados no estudo de caso e culminando em uma avaliação e teste do protótipo. Durante todas as etapas, foram realizada a coleta do tempo de desenvolvimento, a avaliação da complexidade da estrutura das bases de dados e fluxo de trabalho, também registrou-se quaisquer problemas ou erros encontrados para análise posterior. Além disso, buscou-se constantemente analisar e confrontar os benefícios e desafios encontrados na literatura especializada, objetivando averiguar se na prática eles se confirmam e sustentam o ganho de produtividade no desenvolvimento de software.

4.3.2 Desenvolvimento

Nas Tabelas 13 e 14, são apresentados os dados coletados a partir do formulário mencionado na Seção 3.2.1.7. A análise desses dados é detalhada em três etapas que correspondem ao desenvolvimento do protótipo. Embora o protótipo tenha sido implementado em duas plataformas distintas, optou-se por apresentar os resultados de ambas em cada etapa, para permitir uma melhor compreensão dos benefícios e limitações dessas plataformas em cada um dos contextos trabalhados.

Tabela 13 – Resultados Outsystems

RESULTADOS OUTSYSTEMS				
DEV01				
	Tempo de Des.	**Complexidade**	**Dúvidas**	**Erros/Probleblas**
Base de Dados	1H 30min	7 Entidades 6 relacionamentos	Sim	Sim
Interface	30min	Baixa	Sim	Sim
Lógica de Negócios	5h	Alta	Sim	Sim
DEV02				
Base de Dados	3H	5 Entidades 4 relacionamentos	Sim	Sim
Interface	1H	Baixa	Sim	Sim
Lógica de Negócios	4h	Alta	Sim	Sim

Tabela 14 – Resultados Bubble

RESULTADOS BUBBLE				
DEV01				
	Tempo de Des.	**Complexidade**	**Dúvidas**	**Erros/Probleblas**
Base de Dados	1H	4 Entidades 1 Option Set 5 relacionamentos	Sim	Não
Interface	3H	Baixa	Sim	Não
Lógica de Negócios	1H 20min	Baixa	Sim	Não
DEV02				
Base de Dados	1:30min	7 Entidades 3 option Set 11 relacionamentos	Sim	Não
Interface	4H	Baixa	Sim	Não
Lógica de Negócios	1H 30min	Baixa	Sim	Não

Primeira etapa - (Bases de dados)

Durante a etapa inicial do desenvolvimento do protótipo, decidiu-se por estruturar a base de dados, suas tabelas e relacionamentos. Contudo, dado o contexto de utilização de plataformas *low code* e *no code*, a expectativa era de uma otimização ou aceleração nesse processo, talvez com algum recurso visual facilitador. No entanto, ao construirmos as tabelas, notamos que o método utilizado foi essencialmente o mesmo empregado em ferramentas como o *MySQL* com o *WorkBench* da *Oracle*, não apresentando inovações que

pudessem impactar positivamente a produtividade. Durante essa fase, foram realizadas reuniões no início e fim dessa etapa para discutir novos encaminhamentos e garantir a eficácia do processo de estruturação da base de dados.

Iniciou-se pela plataforma *OutSystems*, onde trabalhar com bancos de dados é intuitivo para aqueles familiarizados com Sistemas de Gerenciamento de Banco de Dados (SGBDs). Nesse contexto, um ponto positivo é a presença de uma ferramenta visual de modelagem lógica dos dados, o que permite um planejamento mais estruturado desde o início. Adicionalmente, destaca-se a facilidade em importar dados de planilhas eletrônicas, com apenas um clique, abre-se uma janela para importação e uma tabela com os dados é gerada automaticamente pela plataforma.

Globalmente, os desenvolvedores não enfrentaram dificuldades significativas na criação das bases de dados do protótipo na plataforma *OutSystems*. No entanto, é relevante destacar que não estiveram isentos de consultar a documentação e vídeos de trilhas fornecidos pela plataforma. Além disso, a necessidade de familiarização com a plataforma é independente do nível de conhecimento dos desenvolvedores, como evidenciado nos achados literários e sumarizados na tabela 5, especificamente a limitação: Requisito de Conhecimento.

Durante esse processo, uma questão crucial surgiu como fator limitante para os desenvolvedores. A tabela de usuários, apresenta fortes restrições na versão gratuita, impedindo sua manipulação e relacionamentos com outras. Isso resultou na necessidade de redesenhar a estrutura da base de dados. Para o Dev01, com mais experiência na plataforma, foram geradas 7 entidades e 6 relacionamentos, incluindo a modelagem. Apesar desse número ser maior ao do Dev02, a justificativa reside na implementação adicional de duas tabelas para gerenciar melhor os tipos de materiais a serem descartados e os tipos de coletores, resultando respectivamente nas tabelas(tipodescarte e tipo), conforme pede ser observado na figura 3, e culminado em 1H 30 minutos de trabalho. Para o Dev02, foram necessárias 5 entidades e 4 relacionamentos, demandando um tempo total de 3H, incluindo a modelagem lógica e sua implementação, conforme apresentado na tabela 13. Além disso, a *OutSystems* oferece excelente rastreabilidade de problemas encontrados durante o desenvolvimento, fornecendo correções passo a passo.

Dando continuidade ao desenvolvimento na plataforma *Bubble*, surgiram diversas questões relacionadas à utilização do banco de dados. Por exemplo, a plataforma emprega o conceito de *Option Set* para o que tradicionalmente denominamos de tabelas estáticas, estabelecendo uma dependência em relação a determinados componentes da interface gráfica, como o *DropBox*, e apresenta uma abstração referente a como as tabelas se relacionam. Esse aspecto foi percebido como facilidade de uso pelos desenvolvedores, porém implicou em uma necessidade de investimento adicional de tempo em pesquisas para esclarecer dúvidas que surgiram quanto ao uso dessa abordagem. Apesar de a documentação

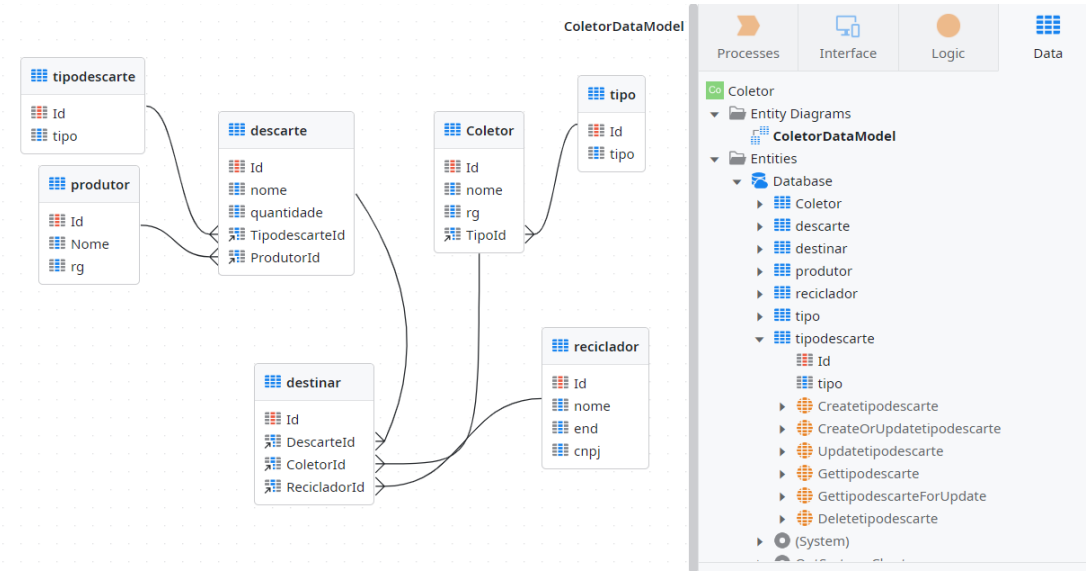


Figura 3 – Base de Dados Outsystems

fornecida pela plataforma *Bubble* ser rica e detalhada, incluindo vídeos curtos de duração variando entre 0:50 a 1:30 minutos sobre cada componente da plataforma, é importante ressaltar que quando se trata da interconexão desses componentes, a documentação pode não ser tão abrangente, demandando a busca por fontes secundárias, como vídeos no *YouTube*, fóruns e sites especializados.

Após essa etapa, a elaboração das bases de dados no *Bubble* pelo Dev01 seguiu uma abordagem de simplificação da complexidade, aproveitando os recursos disponíveis. Um exemplo disso é a utilização da tabela de perfil, que é automaticamente criada no momento da criação do aplicativo e permite a adição de campos adicionais, reduzindo assim a necessidade de entidades distintas. Isso resultou na criação de 4 entidades, 1 *Option Set* e 5 relacionamentos, com um tempo total de 1H. Por outro lado, o Dev02 optou por um caminho de maior complexidade, resultando na criação de 7 entidades, 3 *Option Sets* e 11 relacionamentos. Essa decisão foi justificada pela elaboração de três tabelas distintas para cada perfil de usuário. Como resultado, o tempo necessário para conclusão foi de 1H 30 minutos, com trabalho realizado em dias alternados. Esses resultados são apresentados na Tabela 14. Além disso, a Figura 4 apresenta o resultado das tabelas geradas pelo Dev01.

Como última análise, os desenvolvedores decidiram por aprofundar seu conhecimento nas plataformas, com o objetivo de verificar se o benefício (integração de dados) relatados na literatura, é confirmado na prática. Nesse contexto, o Sistema de Gerenciamento de Banco de Dados (SGBD) utilizado pelo *OutSystems* demonstra ser robusto e escalável, proporcionando conectividade com uma ampla gama de bancos de dados relacionais, como *SQL Server*, *Oracle* e *MySQL*, por meio de componentes pré-definidos.

Em contrapartida, a plataforma *Bubble* simplifica ainda mais o gerenciamento de dados, apresentando um SGBD proprietário, e com isso, eliminando a necessidade de lidar

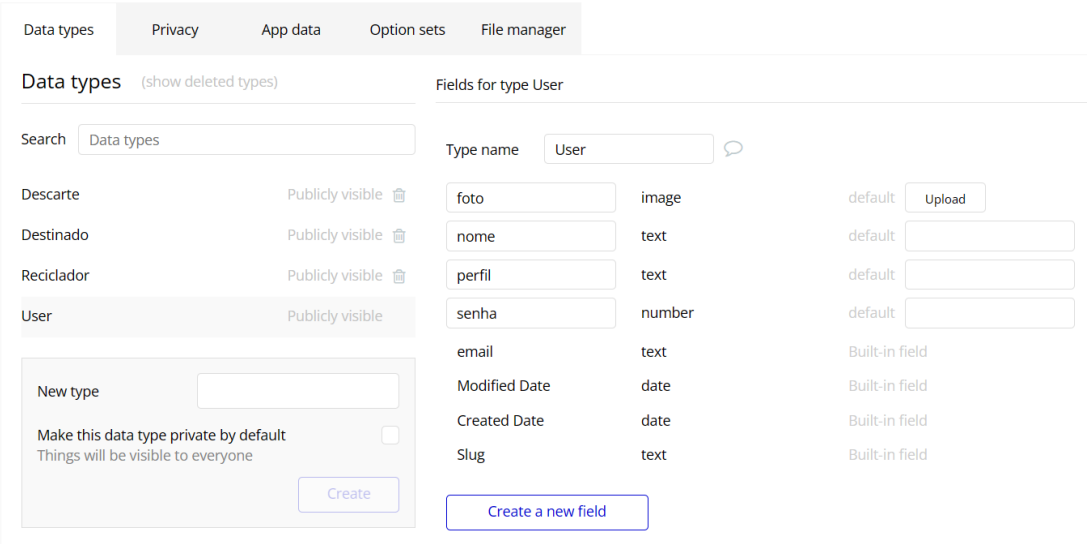


Figura 4 – Base de Dados Bubble

com os aspectos mais tradicionais dos SGBDs de mercado, como configurações de servidor, de usuários, restrições etc. Os usuários do *Bubble* podem aproveitar um banco de dados nativo da plataforma, totalmente gerenciado e escalável. Essa integração direta elimina a complexidade associada a essas configurações, tornando o processo de desenvolvimento mais acessível e rápido para uma ampla gama de usuários. Além disso, a *Bubble* oferece um conjunto rico de APIs para integração com SGBDs externos, como *MySQL*, *Oracle* e *SQL Server*, proporcionando uma maior flexibilidade para os desenvolvedores que necessitam interagir com sistemas de dados existentes, garantindo também integrações com sistemas legados.

Segunda etapa - (Interfaces/Lógica de negócios)

Na segunda etapa, a atenção foi direcionada à elaboração da interface gráfica e do fluxo de trabalho, uma decisão justificada pelo fato de que as bases de dados já estavam prontas, permitindo assim a integração do módulo de interface com a base de dados. Este processo foi seguido por uma discussão sobre a otimização do fluxo de trabalho oferecido pelas plataformas utilizadas. É importante ressaltar que a construção de interfaces fornecidas por essas plataformas é de suma importância, não apenas em termos de facilidade de uso, mas também em relação à sua produtividade. Essas interfaces demonstraram ser altamente eficientes, exigindo pouca interação por parte dos desenvolvedores, o que resultou em uma significativa economia de tempo e recursos durante o desenvolvimento do projeto. Durante essa fase, foram realizadas reuniões no início e fim dessa etapa para discutir novos encaminhamentos e garantir a eficácia do processo de construção das interfaces e fluxos de trabalho.

Durante esse processo, iniciado pela plataforma ***OutSystems***, foi perceptível um dos grandes benefícios prometidos pelas plataformas *low code* e *no code*: a geração automática de interfaces gráficas a partir de suas bases de dados. Isso significa que, ao

possuir um banco de dados, toda a interface é construída automaticamente, com operações de inserção, atualização e busca de dados, tudo com apenas um clique em um comando. Esse processo apresenta um ganho de produtividade significativo, pois a plataforma gera as interfaces para todos as tabelas, sem a necessidade de intervenção manual por parte do desenvolvedor.

Entretanto, quando há a necessidade de modificações, como adicionar um novo campo ao banco de dados e modificar a interface para integrá-lo, ou acrescentar uma operação de delete na base de dados as dificuldades surgem. É exigido do desenvolvedor um conhecimento profundo sobre todos os componentes, suas propriedades e relacionamentos. Assim, muitas tarefas precisarão ser realizadas manualmente, especialmente na interface gráfica, o que consequentemente força o usuário a trabalhar com o *workflow*, uma abstração da lógica de negócios, que gera o código necessário para salvar os dados, atualizar, cadastrar etc., e é apresentado como um fluxograma. Um exemplo de um de nossos cenários que salva dados em uma tabela pode ser observado no fluxogramas apresentado na imagem 5. Por exemplo, ao clicar em um botão "salvar dados" na interface, a codificação necessária para que isso ocorra é abstraída por um *workflow* pré-configurado. No entanto, a modificação desse processo exige do desenvolvedor um conhecimento maior do sistema e habilidades de programação. Essa característica foi percebida pelos desenvolvedores como uma limitação, que distancia usuários iniciantes dessas plataformas.

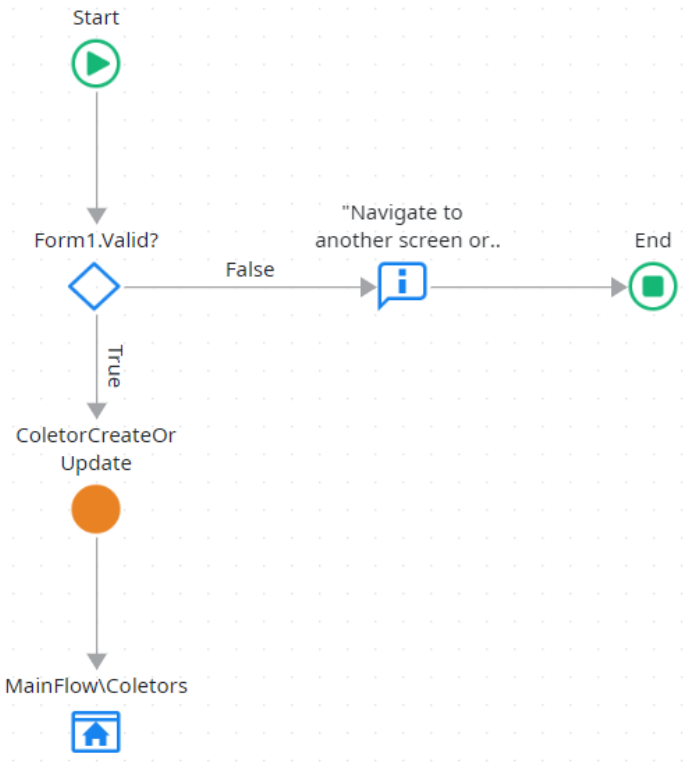


Figura 5 – Fluxo de trabalho outsystems

Assim, não havendo a necessidade de customização para esta etapa, pode-se considerar quase inexistente, uma vez que é gerada automaticamente. No caso do Dev01, algumas alterações na interface foram requeridas, sem afetar as propriedades dos componentes, apenas demandando ajustes mínimos, os quais consumiram um total de 30 minutos. Por outro lado, o Dev02 achou necessário alterar a interface, realizando apenas pequenos ajustes posicionais e dimensionais, o que exigiu 1H de trabalho. No entanto, surgiu um problema ao precisarmos adicionar a função de exclusão de dados, uma vez que modificar o fluxo de trabalho (*Workflow*) se tornou necessário, demandando um tempo significativo e experiência, conforme já relatado anteriormente. Dessa forma, para o Dev01, foram necessárias 5H, enquanto para o Dev02, o tempo total de trabalho foi de 4H, conforme apresentado na tabela 13.

Apesar de não terem sido identificados erros significativos, a exclusão de um componente como entrada de dados, ou a necessidade de adicionar uma nova funcionalidade, podem resultar em problemas. A plataforma oferece soluções tanto visuais quanto textuais, apresentando um guia passo a passo para solucionar tais questões, porém, isso requer conhecimento prévio por parte do usuário.

Por outro lado, na plataforma ***Bubble***, não há a mesma facilidade de geração automática de interfaces a partir das bases de dados encontrada na *OutSystems*. No entanto, construir e reutilizar interfaces é extremamente simples na *Bubble*, bastando arrastar e soltar componentes prontos e funcionais na tela. Uma janela *pop-up* fornece orientações passo a passo sobre como integrar esses componentes à base de dados ou utilizá-los para outras finalidades. É relevante destacar que a plataforma é altamente didática e repleta de recursos explicativos sobre cada componente, o que a torna particularmente vantajosa em comparação com a *OutSystems*, facilitando o desenvolvimento mesmo para indivíduos sem conhecimento em programação. Após uma reunião de conclusão, os desenvolvedores reconheceram essa facilidade como produtiva, uma vez que os componentes da interface são altamente independentes. Isso significa que, ao excluir um botão ou *input*, a aplicação continua a funcionar, e se surgir algum problema, a resolução é direta, ao contrário do que acontece na *OutSystems*.

Contudo, apesar de todas essas facilidades, o verdadeiro destaque de produtividade da *Bubble* é seu fluxo de trabalho. Ao contrário da *OutSystems*, onde a construção é parcial e, a partir daí, a complexidade exige um conhecimento mais aprofundado por parte do desenvolvedor, na *Bubble*, todo o fluxo de trabalho está pré-configurado, sendo abstraído em comandos simples, conforme apresentado na imagem 6. Se é necessário conectar-se a uma base de dados, basta selecionar o fluxo de conexão e pronto. Se é necessário excluir ou atualizar dados, ou fechar janelas, basta escolher o fluxo correspondente, sem a necessidade de escrever linhas de código ou possuir um conhecimento profundo em programação. Esse diferencial, de fato, proporcionou um ganho de produtividade significativo.

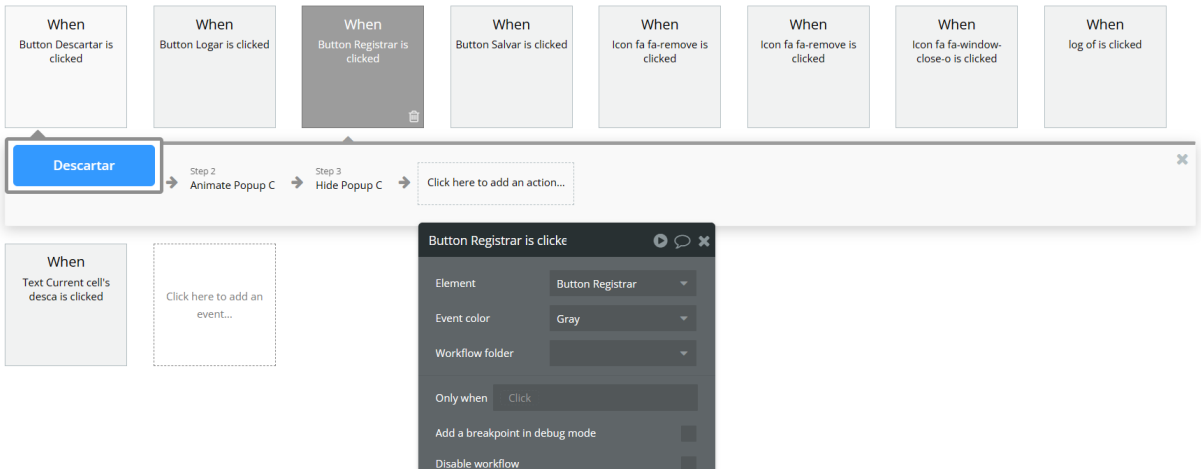


Figura 6 – Fluxo de Trabalho Bubble

Em relação ao tempo dedicado ao desenvolvimento das interfaces, o Dev01, demandou um total de 3H, distribuídas ao longo de dias alternados. Por sua vez, o Dev02, completou sua tarefa em 4H. No que diz respeito à implementação e modificação da lógica de negócios, representada pelo *workflow*, o Dev01 dedicou 80 minutos. Em contrapartida, o Dev02, implementando e modificando, concluiu sua parte em 90 minutos, conforme apresentado na tabela 14.

Terceira etapa - (Perfis de usuários)

Embora a base de dados e as interfaces, reportadas respectivamente nas etapas um e dois, tenham incluído o sistema de login, após uma reunião, optou-se por relatar essa etapa separadamente. Tal decisão foi motivada pelas distintas características identificadas em cada plataforma. Assim, tanto o tempo de desenvolvimento quanto o necessário para configurar o fluxo de trabalho já foram registrados nessas etapas.

Nesta etapa do desenvolvimento do protótipo, ao iniciar a implementação dos perfis de usuários, começando com a plataforma *Outsystems*, observou-se uma peculiaridade. Durante a concepção do projeto, o sistema de controle de usuários é automaticamente integrado a toda estrutura do aplicativo criado, sem a possibilidade de ser desmembrado ou modificado programaticamente. No entanto, é apresentado um sistema completo de controle de usuários e perfis, mas para a versão gratuita utilizada neste protótipo, fica disponível apenas o perfil de desenvolvedor, o que inviabiliza a criação de regras específicas e, assim, limita o desenvolvimento. Esta característica não foi percebida pelos desenvolvedores como uma limitação, mas como um indicativo de que esta plataforma seja mais direcionada para empresas do que para usuários finais. Surpreendentemente, a documentação oficial não faz menção a essa restrição, direcionando suas explicações à plataforma como um todo. Somente após uma extensa pesquisa, de três horas de investimento para os desenvolvedores, foram encontrados alguns comentários em fóruns que esclareceram tal restrição, revelando essa característica não trivial na manipulação deste sistema de controle.

Por outro lado, na plataforma *Bubble*, o sistema de controle de usuários é mais acessível e customizável. Os desenvolvedores desfrutam de uma liberdade substancial, podendo relacionar tabelas de perfis, adicionar campos e demais personalizações, mesmo na versão gratuita da plataforma. Este aspecto é particularmente relevante, pois permite o desenvolvimento e implementação de aplicações de forma abrangente. Apesar disso, também foi necessário um período de aprendizado para lidar com esse sistema, incluindo a consulta à documentação e a visualização de vídeos instrutivos, demandando duas horas de investimento. Após esta etapa, foi possível definir os perfis e atribuir regras às interfaces de maneira intuitiva, utilizando lógica visual, onde as condições são expressas de forma simplificada: "se perfil A, então página A". Este processo dispensa a necessidade de escrita de código, sendo realizado inteiramente de forma visual, através da manipulação de elementos por meio de operações de arrastar e soltar ou da seleção de opções pré-definidas que representam ações específicas do fluxo de trabalho.

4.3.3 Respostas às Questões de Pesquisa

***QP1*: Como os benefícios e desafios identificados na literatura em relação às plataformas de desenvolvimento *low code* e *no code* se manifestam na prática?**

As plataformas *OutSystems* (*low code*) e *Bubble* (no code) são reconhecidas por sua interface intuitiva e abordagem visual na criação de aplicativos, permitindo que até mesmo indivíduos sem conhecimento técnico profundo possam rapidamente conceber protótipos de aplicativos. Estas vantagens são sustentadas por diversos fatores observados na prática, os quais não esgotam o universo de benefícios, mas oferecem uma visão das percepções adquiridas durante o desenvolvimento do protótipo, conforme descrito:

- Abstração de Código: Tanto o *OutSystems* quanto o *Bubble* oferecem interfaces visuais e componentes pré-configurados que capacitam os usuários a criar aplicativos sem a necessidade de codificar manualmente grandes volumes de código. Essa abordagem simplifica consideravelmente o desenvolvimento, permitindo que os desenvolvedores arrastem e soltem componentes e configurem opções através de interfaces gráficas intuitivas. Esse benefício identificado é corroborado pelos achados literários desta pesquisa. Na literatura cinza, tal benefício é caracterizado como ‘Facilidade de Uso’ e ’Aumento de produtividade’, enquanto na literatura branca, é descrito como ‘Produtividade Aumentada’. Esses achados são evidenciadas, respectivamente, nas Tabelas 4 e 11

- Automatização de Tarefas Repetitivas: Ambas as plataformas frequentemente automatizam tarefas rotineiras, como configuração de conexões de banco de dados, implementação de lógica de negócios básica e geração de interfaces de usuário. Isso libera os desenvolvedores para focarem em aspectos mais complexos e criativos

do desenvolvimento, acelerando significativamente o processo como um todo. Esse benefício identificado é elencado nos achados literários desta pesquisa, os quais abrangem múltiplas faces. Na literatura cinza, conforme elucidado na resposta à questão de pesquisa 4.1.2, esse benefício é reconhecido como: 'Aumento de Produtividade', 'Redução de Manutenções' e 'Integração de Dados'. Paralelamente, na literatura branca, conforme descrito em 4.2.2, apresenta-se como: 'Tempo de Desenvolvimento Reduzido', 'Produtividade Aumentada' e 'Flexibilidade'. Estes foram sumarizados, respectivamente, nas Tabelas 4 e 11

- Reutilização de Componentes: Tanto o *OutSystems* quanto o *Bubble* oferecem bibliotecas de componentes e modelos prontos para uso, permitindo que os desenvolvedores reutilizem funcionalidades padrão e layouts de interface em seus aplicativos. Esta prática economiza tempo e esforço, evitando que os desenvolvedores reinventem a roda em cada projeto. Esse benefício, também identificado nos resultados da literatura, apresenta-se com diversas vantagens. Ao reutilizar um componente de conexão com banco de dados, é possível obter maior produtividade, prototipagem mais ágil, melhor integração de dados, maior flexibilidade, entre outros aspectos, conforme apresentado nas tabelas 4 e 11.

- Colaboração Simplificada: Ambas as plataformas simplificam a colaboração entre membros da equipe de desenvolvimento, possibilitando o trabalho simultâneo em um mesmo projeto, visualização de alterações em tempo real e fornecimento de *feedback* imediato. Tal abordagem agiliza o desenvolvimento, reduzindo os atrasos causados por comunicação e coordenação em equipes distribuídas. Apesar desse benefício não ter sido identificado na literatura, ele é fundamental para a redução de custos, aumento da produtividade e gerenciamento de projetos.

Esses benefícios combinados ressaltam as vantagens práticas das plataformas *OutSystems* e *Bubble*, as quais foram examinadas durante o desenvolvimento do protótipo, possibilitando uma criação de aplicativos mais rápida, eficiente e colaborativa. Adicionalmente, todas essas características contribuem para a minimização das necessidades de manutenção do desenvolvimento. Por outro lado, é importante observar que existem desafios ou limitações que podem impactar a liberdade no desenvolvimento, a eficácia e a aplicabilidade, incluindo:

- Dependência de Plataforma: Ambas as plataformas enfrentam o desafio da dependência do ambiente fornecido pela própria plataforma. Isso significa que os aplicativos desenvolvidos nessas plataformas podem ser difíceis de migrar para outras plataformas ou ambientes de execução, limitando assim sua portabilidade e escalabilidade. No entanto há ferramentas disponíveis no mercado que possibilitam a extração do código fonte. Essa constatação é confirmada pelos achados da literatura e amplamente

mencionada como dependência do fornecedor, conforme apresentados nas tabelas 4 e 11.

- Requisito de Conhecimento: Apesar de serem projetadas para serem acessíveis a usuários com diversos níveis de habilidade técnica, tanto o *OutSystems* quanto o *Bubble* ainda exigem um certo entendimento básico de lógica de programação e processos de negócios para aproveitar ao máximo suas funcionalidades. Isso pode representar um desafio para usuários sem experiência prévia em desenvolvimento de software. Tal limitação é sustentada pelos achados literários desta pesquisa, conforme apresentado nas tabelas 4 e 11.

- Funcionalidades Limitadas: As plataformas *OutSystems* e o*Bubble*, podem ter suas funcionalidades limitadas em comparação com abordagens tradicionais de desenvolvimento. Embora ofereçam uma variedade de componentes pré-construídos e integrações, essas plataformas podem não ser capazes de atender a todas as necessidades de desenvolvimento de aplicativos mais complexos. E amplamente discutida na literatura cinza e sumarizada na tabela 4.

- Lógica de Negócio: O desenvolvimento de lógica de negócio mais complexa pode ser um desafio nessas plataformas, especialmente na *Outsystems*. Embora ofereçam ferramentas para implementação de lógica de negócios básica, lidar com requisitos mais complexos pode exigir trabalho adicional e soluções criativas por parte dos desenvolvedores.

- Segurança dos Dados: A segurança dos dados é uma preocupação fundamental em qualquer ambiente de desenvolvimento de software, e as plataformas *OutSystems* e *Bubble* não são exceção. A dependência de componentes pré-construídos e integrações externas pode aumentar o risco de vulnerabilidades de segurança, especialmente se esses componentes não forem devidamente avaliados quanto à segurança.

Essa limitação lógica de negócio e segurança dos dados identificada na prática, é amplamente discutida nos achados literários desta pesquisa, como detalhado nas questões de pesquisa 4.1.2 e 4.2.2 e apresentadas na tabela 4 e 11.

Esses desafios ressaltam a importância de avaliar cuidadosamente as necessidades e requisitos específicos de cada projeto ao utilizar essas plataformas, garantindo que elas sejam aplicadas de maneira eficaz e segura.

***QP2*: Os benefícios e desafios associados às plataformas de desenvolvimento *low code* e *no code*, e sua aplicabilidade varia conforme a plataforma utilizada?**

Na plataforma *OutSystems*, os desenvolvedores têm a possibilidade de construir o fluxo de trabalho utilizando fluxogramas e lógica de negócios. No entanto, à medida que a complexidade do projeto aumenta, os desenvolvedores precisam adquirir um conhecimento

mais profundo sobre a plataforma. Isso ocorre porque, mesmo com a ajuda de fluxogramas e lógica de negócios, eventualmente é necessário interagir diretamente com a infraestrutura e funcionalidades da plataforma para implementar funcionalidades mais avançadas. Essa necessidade de conhecimento mais detalhado pode ser considerada uma desvantagem.

Por outro lado, na plataforma *Bubble*, o fluxo de trabalho é desenvolvido de maneira mais direta, permitindo que os desenvolvedores construam o fluxo apenas clicando onde desejam adicionar um *WorkFlow*, selecionando a ação desejada e finalizando-a utilizando comandos específicos para operações como login, salvar dados, abrir janelas e executar operações em bancos de dados (CRUD - Create, Read, Update, Delete) etc. Essa abordagem simplificada torna significativamente mais acessível o processo de desenvolvimento, tornando-o adequado para uma variedade de usuários, independentemente do nível de conhecimento técnico. Assim, essa característica da plataforma *no code* reforça um benefício extremamente importante: a capacidade de criar aplicativos de forma rápida, eficiente e com menor necessidade de conhecimento detalhado da plataforma.

Assim, a distinção fundamental que observamos em nossa prática entre as diferentes plataformas reside na abordagem adotada para o desenvolvimento do fluxo de trabalho. Essa distinção pode ter um impacto significativo na experiência e na eficiência dos desenvolvedores durante o processo de criação de aplicativos.

***QP3*: De que maneira as constatações dos benefícios e desafios das plataformas *low code* e *no code* sustentam o ganho de produtividade no processo de desenvolvimento de software ao utilizar essas plataformas?**

A observação na prática dos benefícios e desafios das plataformas *OutSystems* e *Bubble*, oferecem experiências valiosos sobre o ganho de produtividade no processo de desenvolvimento de software ao utilizar essas plataformas. Essas constatações demonstram que as vantagens proporcionadas por essas plataformas sustentam efetivamente um aumento na produtividade durante de desenvolvimento de software. E são amplamente mencionadas nos achados literários desta pesquisa, conforme às questões de pesquisa 4.1.2 e 4.2.2.

Primeiramente, os benefícios identificados, como a abstração de código, a automatização de tarefas repetitivas, a reutilização de componentes e a simplificação da colaboração, convergem para uma realidade prática onde o desenvolvimento de aplicativos se torna mais rápido, eficiente e colaborativo. Através da abstração de código e da automatização de tarefas rotineiras, os desenvolvedores são capazes de criar aplicativos sem a necessidade de codificar extensivamente, acelerando o processo de desenvolvimento. Além disso, a reutilização de componentes e a colaboração simplificada permitem uma abordagem mais modular e colaborativa, reduzindo redundâncias e acelerando o desenvolvimento através do trabalho em equipe.

No entanto, é importante reconhecer que existem desafios associados ao uso dessas

plataformas, como a dependência de plataforma, o requisito de conhecimento, funcionalidades limitadas, lógica de negócio e segurança dos dados. Embora esses desafios possam representar obstáculos, eles também proporcionam oportunidades para aprimorar e otimizar o processo de desenvolvimento. Por exemplo, a dependência de plataforma pode ser mitigada através do uso de ferramentas de extração de código fonte, permitindo uma maior flexibilidade e portabilidade dos aplicativos desenvolvidos nessas plataformas. Da mesma forma, os requisitos de conhecimento podem ser superados com programas de treinamento e capacitação, capacitando os usuários a tirar o máximo proveito das funcionalidades oferecidas pelas plataformas.

Em suma, as constatações dos benefícios das plataformas *OutSystems* e *Bubble*, corroboram o ganho de produtividade no processo de desenvolvimento de software. Ao superar os desafios e aproveitar ao máximo os benefícios oferecidos por essas plataformas, as organizações podem acelerar o desenvolvimento de aplicativos, reduzir custos e aumentar a eficiência, resultando em um processo de desenvolvimento mais ágil e produtivo.

4.3.4 Considerações Finais

Com base nas respostas às questões de pesquisa e nas constatações durante o desenvolvimento do protótipo nas plataformas *OutSystems* e *Bubble*, é possível fazer considerações finais abrangentes sobre o ganho de produtividade proporcionado por essas ferramentas de desenvolvimento de software.

Ambas as plataformas, tanto *OutSystems* quanto *Bubble*, oferecem vantagens significativas que contribuem para aumentar a produtividade no processo de desenvolvimento de software. A abstração de código, automatização de tarefas repetitivas, reutilização de componentes e simplificação da colaboração são benefícios tangíveis que aceleram o desenvolvimento de aplicativos, permitindo que os desenvolvedores foquem em aspectos mais criativos e complexos do projeto. Esses aspectos reduzem consideravelmente o tempo necessário para desenvolver aplicativos, proporcionando uma vantagem competitiva para as organizações que utilizam essas plataformas.

No entanto, é importante reconhecer que existem desafios associados ao uso dessas plataformas, como a dependência de plataforma, requisitos de conhecimento, funcionalidades limitadas e lógica de negócio. Esses desafios podem representar obstáculos para alguns projetos e requerem uma abordagem cuidadosa para serem superados. No entanto, com o apoio adequado e investimento em capacitação, muitos desses desafios podem ser mitigados.

Um aspecto crucial é a diferença fundamental na abordagem de desenvolvimento entre as plataformas *OutSystems* e *Bubble*, especialmente em relação à construção de interfaces de usuário e fluxos de trabalho.

Na plataforma *OutSystems*, os desenvolvedores têm a capacidade de construir interfaces de usuário e fluxos de trabalho utilizando uma combinação de desenvolvimento visual e programação *low code*. Eles podem criar fluxos de trabalho usando fluxogramas e lógica de negócios, o que oferece flexibilidade e controle sobre o processo de desenvolvimento. No entanto, à medida que a complexidade do projeto aumenta, os desenvolvedores podem precisar interagir diretamente com a infraestrutura da plataforma para implementar funcionalidades mais avançadas, exigindo um conhecimento mais profundo da plataforma.

Por outro lado, na plataforma *Bubble*, o processo de desenvolvimento é simplificado, permitindo que os desenvolvedores construam interfaces e fluxos de trabalho de maneira mais direta e intuitiva, sem a necessidade de codificação. Os desenvolvedores podem criar fluxos de trabalho clicando em elementos visuais e selecionando ações específicas, tornando o processo acessível mesmo para usuários sem experiência técnica profunda. Essa abordagem simplificada é especialmente benéfica para projetos que requerem desenvolvimento rápido e iteração frequente.

Essas distinções nas interfaces e fluxos de trabalho entre as plataformas têm um impacto significativo na experiência do desenvolvedor e na eficiência do processo de desenvolvimento. Enquanto o *OutSystems* oferece maior controle e flexibilidade, a *Bubble* prioriza a simplicidade e acessibilidade.

Portanto, ao considerar o ganho de produtividade proporcionado por essas plataformas, é importante levar em conta não apenas as vantagens e desafios gerais discutidos anteriormente, mas também as diferenças na abordagem de desenvolvimento, incluindo as interfaces e fluxos de trabalho oferecidos por cada plataforma. Esses aspectos adicionais podem influenciar significativamente a escolha da plataforma mais adequada e a eficácia do processo de desenvolvimento como um todo. Um outro aspecto importante a se considerar é a diferença na abordagem de desenvolvimento entre as duas plataformas. Enquanto o *OutSystems* é mais orientado para uma abordagem *low code*, permitindo maior flexibilidade e controle sobre o desenvolvimento, a *Bubble* adota uma abordagem *no code*, simplificando ainda mais o processo de desenvolvimento e tornando-o acessível até mesmo para usuários sem experiência técnica. Essa distinção influencia diretamente a experiência do desenvolvedor e a velocidade com que os aplicativos podem ser criados.

Em última análise, as plataformas *OutSystems* e *Bubble* oferecem uma maneira eficaz de aumentar a produtividade no desenvolvimento de software, cada uma com suas próprias vantagens e desafios. Ao escolher a plataforma mais adequada às necessidades do projeto e investir na capacitação dos desenvolvedores, as organizações podem colher os benefícios de um processo de desenvolvimento mais rápido, eficiente e colaborativo.

5 Conclusões

5.1 Síntese

Com base nas análises realizadas sobre as plataformas *OutSystems* e *Bubble*, é evidente que ambas oferecem vantagens distintas que podem impulsionar significativamente o ganho de produtividade no desenvolvimento de software. Tanto a abordagem de desenvolvimento *low code* da *OutSystems* quanto a abordagem *no code* da *Bubble* têm o potencial de acelerar o processo de criação de aplicativos, reduzir a necessidade de codificação manual extensiva e simplificar a colaboração entre equipes de desenvolvimento.

Ao utilizar a *OutSystems*, os desenvolvedores têm acesso a uma gama de ferramentas que permitem uma maior flexibilidade e controle sobre o desenvolvimento de aplicativos, especialmente em projetos mais complexos que exigem uma abordagem mais personalizada. Por outro lado, a *Bubble* oferece uma experiência de desenvolvimento mais simplificada e acessível, permitindo que até mesmo usuários sem experiência técnica profunda criem aplicativos de forma rápida e eficiente.

No entanto, é importante reconhecer que ambas as plataformas apresentam desafios e limitações, como dependência de plataforma, requisitos de conhecimento e funcionalidades limitadas. Superar esses desafios requer uma abordagem cuidadosa e investimento em capacitação e suporte adequados.

5.2 Contribuições

Para a Indústria de Software: esta pesquisa oferece uma avaliação empírica das plataformas *low code Outsystems* e *no code Bublble*, fornecendo avaliações sobre seu impacto na produtividade e eficiência no desenvolvimento de software. Isso permite que as empresas da indústria de software tomem decisões mais informadas sobre a adoção e implementação dessas tecnologias, visando melhorar seus processos de desenvolvimento e entrega de software.

Para os Desenvolvedores: a pesquisa identifica uma série de desafios e vantagens associados às plataformas *low code Outsystems* e *no code Bubble*, oferecendo aos desenvolvedores uma visão mais clara sobre como essas tecnologias podem melhorar seu trabalho diário. Isso permite que os desenvolvedores entendam melhor os recursos disponíveis e os possíveis obstáculos ao trabalhar com essas plataformas, facilitando a adaptação e o aproveitamento máximo de suas funcionalidades.

Para os Desenvolvedores de Plataformas *Low Code* e *No Code*: esta pesquisa fornece

feedback valioso das plataformas *Outsystems* e *Bubble*. Isso pode orientar o desenvolvimento futuro dessas plataformas, ajudando os desenvolvedores a aprimorar suas funcionalidades, resolver problemas identificados e oferecer uma experiência mais satisfatória aos usuários.

5.3 Trabalhos futuros

Como indicativo para trabalhos futuros, sugere-se a realização de avaliações empíricas com um grupo mais amplo de participantes e a inclusão de uma variedade maior de plataformas. Isso permitirá uma compreensão mais abrangente dos benefícios e desafios associados a diferentes abordagens de desenvolvimento e ajudará a informar a seleção de plataformas mais adequadas para projetos específicos. Além disso, uma análise mais aprofundada dos impactos do uso dessas plataformas na produtividade e na qualidade do software seria valiosa para aprimorar as práticas de desenvolvimento de software no futuro.

Como uma segunda recomendação para trabalhos futuros, é fundamental investigar como plataformas *low code* e *no code* se comportam no contexto de manutenção e evolução de software após a entrega. O acompanhamento contínuo desses processos oferece uma visão mais abrangente sobre a eficácia das práticas adotadas, bem como a sustentabilidade dos resultados ao longo do tempo. Além disso, permite identificar potenciais desafios ou limitações que podem surgir durante as fases posteriores ao lançamento, facilitando ajustes e melhorias contínuas.

Por fim, sugere-se conduzir estudos que explorem o uso da IA generativa em plataformas *low code* e *no code*, buscando entender como a programação visual dessas plataformas pode se beneficiar de recursos de geração de código baseados em *prompts* de IA, como o *ChatGPT*. Uma análise detalhada dessa interação pode revelar benefícios significativos, como a aceleração do desenvolvimento, a simplificação de tarefas complexas e a personalização mais rápida de aplicativos.

5.4 Publicações

O primeiro momento desta pesquisa, foi dedicado a literatura cinza, que resultou na publicação do artigo "Low-code and No-code Technologies Adoption: A Gray Literature Review" o qual foi apresentado na trilha principal do XIX Simpósio Brasileiro de Sistemas de Informação – SBSI, que ocorreu no período de 29/05 a 01/06/2023, em Maceió-AL, https://dl.acm.org/doi/10.1145/3592813.3592929.

A conclusão desta pesquisa abre caminho para uma segunda publicação, desta vez incorporando dados coletados empiricamente por meio de uma abordagem de estudo de caso. Esta adição é especialmente valiosa, pois ofereceu a oportunidade de avaliar a

aplicação prática dos achados literários, proporcionando uma perspectiva mais abrangente sobre o tema em questão.

5.4.1 Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento (403512/2021-2)

Referências

ALAMIN, M. A. A. et al. An empirical study of developer discussions on low-code software development challenges. In: IEEE. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2021. p. 46–57. Citado 4 vezes nas páginas 5, 15, 18 e 23.

ANDRADE, M. *Fundamentos de Metodologia Científica*. 8. ed. [S.l.]: Atlas, 2017. Citado na página 27.

ARORA, R.; GHOSH, N.; MONDAL, T. Sagitec software studio (s3) - a low code application development platform. In: *2020 International Conference on Industry 4.0 Technology (I4Tech)*. [S.l.]: IEEE, 2020. Citado na página 50.

BERGIN, J. et al. *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. [S.l.]: Dreamtech Press, 2005. Citado na página 12.

BRASIL, D. Explorando plataformas low code no code. *blog.duda.co*, Out 2021. Disponível em: <https://blog.duda.co/pt-br/plataformas-low-code-no-code>. Citado na página 14.

BRAUN, V.; CLARKE, V. Using thematic analysis in psychology. *Qualitative research in psychology*, Taylor & Francis, v. 3, n. 2, p. 77–101, 2006. Citado na página 45.

brocoders. *The Pros and Cons of Low-Code Development*. 2021. Disponível em: <https://hackernoon.com/the-pros-and-cons-of-low-code-development-4y2p33g9>. Acesso em: 27 de novembro 2022. Citado 2 vezes nas páginas 42 e 44.

BROOKS, F. *The Mythical Man-Month: Essays on Software Engineering*. [S.l.]: Addison-Wesley, 1995. Citado na página 3.

Calin Abaiesitei. *13 Benefits Of Low-Code Development That Improve the Way Tech People Work*. 2022. Disponível em: <https://teleporthq.io/blog/benefits-of-low-code-development>. Acesso em: 27 de novembro 2022. Citado na página 43.

CHEN, P.; BABAR, M. A.; BABAR, M. A. Low-code/no-code development: What the future holds. *IEEE Software*, IEEE, v. 37, n. 6, p. 12–16, 2020. Citado na página 13.

Clay Richardson, John Rymer. *New Development Platforms Emerge For Customer-Facing Applications*. 2014. Disponível em: <https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411>. Acesso em: 11 de Julho 2023. Citado na página 13.

Donal Tobin. *What is No-Code?* 2021. Disponível em: <https://www.integrate.io/blog/what-is-no-code/>. Acesso em: 27 de novembro 2022. Citado na página 44.

FOUNDATION, E. *Eclipse IDE for Java Developers*. 2018. Disponível em: <https://www.eclipse.org/>. Citado na página 13.

FOWLER, M. *Inversion of Control Containers and the Dependency Injection Pattern*. 2004. Disponível em: <https://martinfowler.com/articles/injection.html>. Citado na página 12.

GAROUSI, V.; FELDERER, M.; MÄNTYLÄ, M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and software technology*, Elsevier, v. 106, p. 101–121, 2019. Citado 2 vezes nas páginas 21 e 22.

GOMES, P. M.; BRITO, M. A. Low-code development platforms: A descriptive study. In: *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.]: IEEE, 2022. Citado na página 51.

JETBRAINS. *IntelliJ IDEA: The Java IDE for Professional Developers*. 2020. Disponível em: <https://www.jetbrains.com/idea/>. Citado na página 13.

JUHáS, G. et al. Low-code platforms and languages: the future of software development. In: *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. [S.l.]: IEEE, 2022. Citado na página 50.

KAMEI, F. et al. Grey literature in software engineering: A critical review. *Information and Software Technology*, v. 138, p. 106609, 2021. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584921000860>. Citado na página 21.

KASS, S.; STRAHRINGER, S.; WESTNER, M. Drivers and inhibitors of low code development platform adoption. In: *2022 IEEE 24th Conference on Business Informatics (CBI)*. [S.l.]: IEEE, 2022. Citado na página 50.

KHORRAM, F.; MOTTU, J.-M.; SUNYÉ, G. Challenges & opportunities in low-code testing. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. [S.l.: s.n.], 2020. p. 1–10. Citado na página 15.

KRISHNARAJ, N. et al. Comparative study on various low code business process management platforms. In: *2022 International Conference on Inventive Computation Technologies (ICICT)*. [S.l.: s.n.], 2022. Citado 3 vezes nas páginas 5, 6 e 17.

KULKARNI, V. et al. A brief history of low code platforms. *arXiv preprint arXiv:2001.04440*, 2020. Citado na página 13.

KäSS, S.; STRAHRINGER, S.; WESTNER, M. Practitioners' perceptions on the adoption of low code development platforms. *IEEE Access*, 2023. Citado 3 vezes nas páginas 5, 7 e 17.

LUO, Y. et al. Characteristics and challenges of low-code development: the practitioners' perspective. In: *Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*. [S.l.: s.n.], 2021. p. 1–11. Citado 5 vezes nas páginas 4, 6, 16, 23 e 24.

MARTINS, R. et al. An overview on how to develop a low-code application using outsystems. In: *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. [S.l.]: IEEE, 2020. Citado na página 50.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 37, n. 4, p. 316–344, 2005. Citado 2 vezes nas páginas 11 e 12.

MICROSOFT. *Visual Studio Code - Code Editing. Redefined*. 2019. Disponível em: <https://code.visualstudio.com/>. Citado na página 13.

MONDAY.COM, T. na. Plataformas low-code e no-code: um guia completo. *monday.com Blog*, Jan 2022. Disponível em: <https://monday.com/blog/pt/produto-pt/plataformas-low-code-e-no-code-um-guia-completo/>. Citado na página 14.

PADUA, W. *Engenharia de Software - Produtos*. [S.l.]: LTC Editora, 2019. Citado na página 4.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. [S.l.: s.n.], 2008. p. 1–10. Citado na página 22.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. [S.l.]: McGraw-Hill, 2019. Citado 3 vezes nas páginas 3, 6 e 11.

PRINZ, N.; RENTROP, C.; HUBER, M. Low-code development platforms-a literature review. In: *AMCIS*. [S.l.: s.n.], 2021. Citado 5 vezes nas páginas 5, 6, 16, 18 e 24.

RAHMANIAN, M.; KHORASANI, S.; BAGHERI, E. No-code platforms: A technological solution to the global developer shortage. *IEEE Access*, IEEE, v. 8, p. 147434–147448, 2020. Citado na página 14.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, Springer, v. 14, n. 2, p. 131–164, 2009. Citado 3 vezes nas páginas 7, 28 e 29.

SEAMAN, C. B. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, IEEE, v. 25, n. 4, p. 557–572, 1999. Citado na página 28.

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. [S.l.]: Pearson, 2018. Citado 4 vezes nas páginas 3, 4, 5 e 11.

STAKE, R. E. *The art of case study research*. [S.l.]: Sage, 1995. Citado na página 28.

TOMASSETTI, F. *The complete guide to (external) Domain Specific Languages*. 2017. Disponível em: <https://tomassetti.me/domain-specific-languages/>. Citado 2 vezes nas páginas 11 e 12.

WAZLAWICK, R. S. *Metodologia de Pesquisa para ciência da Computação*. 3. ed. [S.l.]: GEN LTC, 2020. Citado na página 27.

YIN, R. K. *Case Study Research and Applications: Design and Methods*. [S.l.]: Sage Publications, 2018. Citado na página 28.

Apêndices

APÊNDICE A – Formulário de Coleta de Dados

Executor dos Testes: responsável pela coleta de dados.

Momento da Coleta: Os dados serão coletados durante o desenvolvimento do protótipo. Formulário de Coleta de Dados:

Nome do Desenvolvedor:

Data e Hora:

Plataforma Utilizada (OutSystems ou No Code Bubble):

Funcionalidade Desenvolvida(Login / Base de Dados / Interfaces):

Tempo de Desenvolvimento: na funcionalidade em desenvolvimento

Complexidade do Modelo de Dados: DB(Quantas Entidades/ relacionamentos)

Problemas Encontrados (se houver, mesmo que resolvido, tem que ser relatado):